# Preface

In July 2006 the participants at the 17th Australasian Workshop on Combinatorial Algorithms (AWOCA) met to consider future directions for a gathering that, from a humble beginning in 1989, had taken on an important role among Australian (and other) computer scientists and mathematicians working in combinatorial areas. The decision was taken to — in the Aussie phrase — go for it: to upgrade the academic profile of the workshop, to arrange for publication of the Proceedings by a world-class scientific publisher (College Publications), and to internationalise, changing the name to IWOCA (International Workshop on Combinatorial Algorithms). At the same time, the meeting urged that the traditional problem-oriented (enquiry-oriented) nature of AWOCA be preserved; this decision has led to the prominent scheduling of problem sessions in the IWOCA program, and to the establishment of a permanent combinatorial problem webpage, accessible from the permanent IWOCA website at

`http://www.iwoca.org`

IWOCA 2007 (arguably either the 18th or the first) was held 5–9 November 2007 at Rafferty's Resort on Lake MacQuarrie, a picturesque location close to Newcastle in the Australian state of New South Wales. The meeting was sponsored by The University of Newcastle and hosted by its School of Electrical Engineering & Computer Science. Calls for papers were distributed round the world, using LISTSERVE and other e-mail lists, resulting in 42 submissions, of which 24 were accepted after review by at least two (generally three) referees, making use of the EasyChair system. In the conference, 20 of the accepted submissions were actually presented, and so included in these Proceedings. In addition, IWOCA 2007 featured talks by 8 invited speakers. Altogether, the authors at IWOCA 2007 represented universities from 16 territories: Australia, Canada, Chile, China, Cuba, Czech Republic, Greece, Indonesia, Israel, Japan, the Philippines, Poland, Russia, Spain, Taiwan, and the UK.

We gratefully acknowledge the fine work done by the members of the Program Committee and the Organising Committee in ensuring the success of the first IWOCA and thus assisting in the establishment of IWOCA as a forum for algorithmic research on combinatorial objects.

For details of the meeting, including the complete program and a list of participants, please access

`http://www.eng.newcastle.edu.au/ ~ iwoca2007/`

November 2007

Ljiljana Brankovic
Yuqing Lin
William F. Smyth

# Conference Organization

## Steering Committee

| | |
|---|---|
| Costas Iliopoulos | King's College London, UK |
| Mirka Miller | University of Ballarat, Australia |
| | University of West Bohemia, Czech Republic |
| Bill Smyth | McMaster University, Canada |
| | Curtin University, Australia |

## Programme Chairs

| | |
|---|---|
| Ljiljana Brankovic | The University of Newcastle, Australia |
| Bill Smyth | McMaster University, Canada |
| | Curtin University, Australia |

## Programme Committee

| | |
|---|---|
| Amihood Amir | Bar-Ilan University, Israel |
| Martin Baca | Technical University of Kosice, Slovakia |
| Edy Tri Baskoro | Institut Teknologi Bandung, Indonesia |
| Hajo Broersma | Durham University, UK |
| Pino Caballero-Gil | University of La Laguna, Spain |
| Kun-Mao Chao | National Taiwan University, Taiwan |
| Arbee L. P. Chen | National Chengchi University, Taiwan |
| Francis Y.L. Chin | Hong Kong University, Hong Kong |
| Charlie Colbourn | Arizona State University, USA |
| Derek Corneil | University of Toronto, Canada |
| Jackie Daykin | Royal Holloway College, UK |
| Frank Dehne | Carleton University, Canada |
| Diane Donovan | University of Queensland, Australia |
| Xiaodong Hu | Chinese Academy of Sciences, China |
| Andrei Kelarev | University of Tasmania, Australia |
| Jan Kratochvil | Charles University, Czech Republic |
| Selda Küçükçifçi | Koç University, Turkey |
| Gad M. Landau | University of Haifa, Israel |
| Thierry Lecroq | University of Rouen, France |
| Jimmy Lee | Hong Kong University, Hong Kong |
| Paulette Lieby | NICTA, Australian National University, Australia |
| Yuqing Lin | The University of Newcastle, Australia |
| Stefano Lonardi | University of California Riverside, USA |

| | |
|---|---|
| Prabhu Manyem | University of Ballarat, Australia |
| Laurent Mouchard | University of Rouen, France |
| Gonzalo Navarro | University of Chile, Chile |
| Takao Nishizeki | Tohoku University, Japan |
| Kunsoo Park | Seoul National University, Korea |
| Andrzej Proskurowski | Oregon State University, USA |
| Bharati Rajan | Loyola College, India |
| Rajeev Raman | University of Leicester, UK |
| Joe Ryan | University of Ballarat, Australia |
| Zdenek Ryjacek | University of West Boehmia, Czech Republic |
| Wojciech Rytter | Warsaw University, Poland |
| Jamie Simpson | Curtin University, Australia |
| Jozef Sirán | Open University, UK |
| Michiel Smid | Carleton University, Canada |
| Kathleen Steinhöfel | King's College London, UK |
| Anne Street | University of Queensland, Australia |
| Athanasios Tsakalidis | University of Patras, Greece |
| Gabriel Valiente | Technical University of Catalonia, Spain |
| Jito Vanualailai | The University of the South Pacific, Fiji |
| Koichi Wada | Nagoya Institute of Technology, Japan |
| Sue Whitesides | McGill University, Canada |
| Zhiyou Wu | University of Ballarat, Australia |
| Chee Yap | New York University, USA |

## Local Organization

| | |
|---|---|
| Yuqing Lin (Chair) | The University of Newcastle, Australia |
| Mousa Alfalayleh | The University of Newcastle, Australia |
| Ljiljana Brankovic | The University of Newcastle, Australia |
| Beti Georgievski | The University of Newcastle, Australia |
| Alexandre Mendes | The University of Newcastle, Australia |
| Jianmin Tang | University of Ballarat, Australia |

# External Reviewers

Fusheng Bai
Roman Cada
Feng Chen
Shir Feibish
Pinar Heggernes
Danny Hermelin
Takehiro Ito
Taisuke Izumi
Yoshiaki Katayama
Roman Kuzel
Slawomir Lasota
Hiroki Morizumi
Kiki Ariyanti Sugeng
Oren Weimann
Xiao Zhou

# Table of Contents

# $L(2,1)$-labeling of Bipartite Permutation Graphs

Toru Araki*

Department of Computer and Information Sciences, Iwate University
Morioka, Iwate, 020-8551, Japan
arakit@cis.iwate-u.ac.jp

**Abstract.** An $L(2,1)$-labeling of a graph $G$ is an assignment $f$ from vertices of $G$ to the set of non-negative integers $\{0,1,\ldots,\lambda\}$ such that $|f(u)-f(v)| \geq 2$ if $u$ and $v$ are adjacent, and $|f(u)-f(v)| \geq 1$ if $u$ and $v$ are at distance 2 apart. The minimum value of $\lambda$ for which $G$ has $L(2,1)$-labeling is denoted by $\lambda(G)$. The $L(2,1)$-labeling problem is related to the channel assignment problem for wireless networks.
In this paper, we present a polynomial time algorithm for computing $L(2,1)$-labeling of a bipartite permutation graph $G$ such that the largest label is at most $bc(G)+1$, where $bc(G)$ is the biclique number of $G$. Since $\lambda(G) \geq bc(G)$ for any bipartite graph $G$, the upper bound is nearly optimal.

## 1 Introduction

The *channel assignment problem* for wireless networks is to assign a channel to each radio transmitter so that close transmitters are received channels so as to avoid interference. This situation can be modeled by a graph whose vertices are the radio transmitters, and the adjacency indicate possible interference. The aim is to assign integers (corresponding to the channel) to the vertices such that adjacent vertices receive integers at least 2 apart, and nonadjacent vertices with a common neighbor receive distinct integers. This is called $L(2,1)$-labeling problem which is widely accepted model for the channel assignment problem. A formal definition is given as follows.

**Definition 1.** *An $L(2,1)$-*labeling *of $G$ is an assignment $f$ from $V$ to the set of integers $\{0,1,\ldots,\lambda\}$ such that $|f(u)-f(v)| \geq 2$ if $uv \in E$ and $|f(u)-f(v)| \geq 1$ if $\mathrm{dist}(u,v)=2$. The minimum value of $\lambda$ for which $G$ has $L(2,1)$-labeling is denoted by $\lambda(G)$.*

The notion of $L(2,1)$-labeling has attracted a lot of attention for not only its motivation by the channel assignment problem, and also for its interesting graph theoretic properties. Griggs and Yeh [1] first considered this problem. There are many papers that study the problem for several graph classes (for example, see surveys [2, 3]). The complexity for deciding $\lambda(G) \leq k$ for fixed $k$

* This work was supported by Japan Society of the Promotion of Science, Grant-in-Aid for Young Scientists (B) (no. 19700001).

is NP-complete [1], and for bipartite graphs and chordal graph are also NP-complete [4].

In this paper, we focus on the class of *bipartite permutation graphs* which is a permutation graph and bipartite graph. This class was investigated by Spinrad, Brandstädt, and Stewart [5]. Studies for the class are motivated by the fact that many NP-hard problems are efficiently solved in graphs of this class. For example, algorithms for domination problems [6, 7], the path partition problem [8], and the longest path problem [9] were investigated. Books [10, 11] include surveys some algorithmic result for the class. Boldaender et al. [4] proved that $\lambda(G) \leq 5\Delta - 2$ for any permutation graph, where $\Delta$ is the maximum degree of $G$, and such labeling is calculated by a polynomial time algorithm that is greedy manner.

We consider the $L(2,1)$-labeling problem for bipartite permutation graphs. We present a polynomial time algorithm for computing a *nearly* optimal labeling. More precisely, the maximum value assigned for vertices is at most $bc(G) + 1$, where $bc(G)$ is the biclique number.

## 2 Preliminaries

Let $G = (V, E)$ be a graph with vertex set $V$ and edge set $E$. The *neighborhood* of a vertex $u$ is $N_G(u) = \{v \mid uv \in E\}$. The *degree* of a vertex $u$ is $\deg u = |N_G(u)|$. The *distance* between two vertices $u$ and $v$, denoted by $\mathrm{dist}(u, v)$, is the length of shortest path between $u$ and $v$. A graph $G = (V, E)$ is *bipartite* if $V$ can be partitioned into two subsets $X$ and $Y$ such that every edge joins a vertex in $X$ and another vertex in $Y$. A partition $X \cup Y$ of $V$ is called *bipartition*. A bipartite graph with bipartition $X \cup Y$ is denoted by $G = (X, Y, E)$. A bipartite graph $G = (X, Y, E)$ is *complete* if each vertex in $X$ is adjacent to every vertices in $Y$. For a bipartite graph, a subset of vertices is *biclique* if it induces a complete bipartite subgraph. The *biclique number* of a bipartite graph $G$ is the number of vertices in a maximum biclique of $G$ and it is denoted by $bc(G)$.

A graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ is called a *permutation graph* if there is a permutation $\pi$ over $\{1, 2, \ldots, n\}$ such that $v_i v_j \in E$ if and only if $(i - j)(\pi^{-1}(i) - \pi^{-1}(j)) < 0$. When a permutation graph is bipartite, it is said to be a *bipartite permutation graph*.

Intuitively, a permutation graph can be constructed from a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ on $\{1, 2, \ldots, n\}$ in the following visual manner. Line up the numbers 1 to $n$ horizontally on a line $L_1$. On the line below it, line up the corresponding permutation so that $\pi_i$ is below $i$ on a line $L_2$. Then connect each $i$ and $\pi_i^{-1}$ with a line segment which is corresponding to vertex $v_i$. The resulting diagram is referred to as a *permutation diagram*. In the permutation graph corresponding to $\pi$, two vertices $v_i$ and $v_j$ are adjacent if and only if the corresponding lines are crossing. An example of a bipartite permutation graph and the corresponding permutation diagram is shown in Fig. 1.

In the permutation diagram of a bipartite permutation graph $G = (X, Y, E)$, we can order line segments $x_1, x_2, \ldots, x_m$ in $X$ from left to right (these are drawn

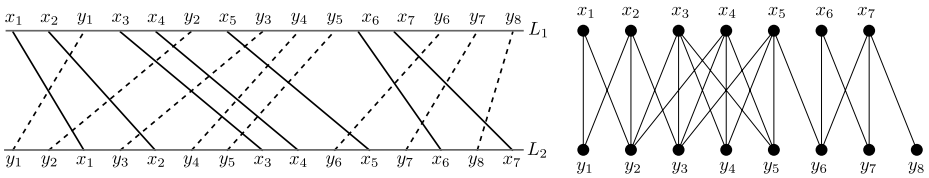



**Fig. 1.** A bipartite permutation graph and the corresponding permutation diagram.

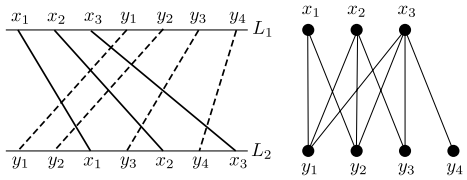



**Fig. 2.** A chain graph and the corresponding permutation diagram.

by solid lines in Fig. 1). We also order vertices $y_1, y_2, \ldots, y_n$ in $Y$ from left to right (these are dotted lines in Fig. 1). From now on, we suppose that vertices in $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ are sorted such that the corresponding lines are arranged from left to right in the permutation diagram. It should be noted that Spinrad et al. [5] developed an $O(|V|+|E|)$ time algorithm for recognizing whether a given graph is a bipartite permutation graph and producing such orderings of the vertices if so.

A bipartite graph $G = (X, Y, E)$ is a *chain graph* if vertices can be ordered by inclusion: that is, there is an ordering of vertices $x_1, x_2, \ldots, x_m$ in $X$ and $y_1, y_2, \ldots, y_n$ in $Y$ such that $N_G(x_1) \subseteq N_G(x_2) \subseteq \cdots \subseteq N_G(x_m)$ and $N_G(y_n) \subseteq \cdots \subseteq N_G(y_2) \subseteq N_G(y_1)$. It is known that any chain graph is a bipartite permutation graph [9].

**Lemma 1 (Uehara, Valiente [9]).** *Let $G = (X, Y, E)$ be a connected chain graph with $N_G(x_1) \subseteq N_G(x_2) \subseteq \cdots \subseteq N_G(x_m)$ and $N_G(y_n) \subseteq \cdots \subseteq N_G(y_2) \subseteq N_G(y_1)$. Then, it has a corresponding permutation diagram such that (1) $x_1 < x_2 < \cdots < x_m < y_1 < y_2 < \cdots < y_n$ on $L_1$, and (2) $y_1 < x_1$ and $y_n < x_m$ on $L_2$. Conversely, if a graph $G$ has a corresponding permutation diagram satisfying conditions (1) and (2), then it is a connected chain graph. See Fig. 2.*

## 3 Labeling of Chain Graphs

In this section, we show that an optimal $L(2,1)$-labeling of chain graph can be solved in linear time. For simplicity, we may assume that the given graph is connected. The following is easily obtained.

**Lemma 2.** *For a complete bipartite graph $G = (X, Y, E)$, $\lambda(G) = |X| + |Y|$.*

It is obvious that $\lambda(G) \geq \lambda(H)$ if $H$ is a subgraph of $G$. Hence we obtain a lower bound of $\lambda(G)$ for any bipartite graph $G$ from Lemma 2.

**Corollary 1.** *$\lambda(G) \geq bc(G)$ for any bipartite graph $G$.*

**Theorem 1.** *Let $G = (X, Y, E)$ be a connected chain graph such that $N_G(x_1) \subseteq N_G(x_2) \subseteq \cdots \subseteq N_G(x_m)$ and $N_G(y_n) \subseteq \cdots \subseteq N_G(y_2) \subseteq N_G(y_1)$. Define a labeling $\mathsf{cl}$ of vertices such that*

$$\mathsf{cl}(x_i) = bc(G) - m + i, \text{ for } 1 \leq i \leq m,$$
$$\mathsf{cl}(y_j) = j - 1, \text{ for } 1 \leq j \leq n.$$

*Then $\mathsf{cl}$ is an optimal $L(2,1)$-labeling of $G$. The labeling $\mathsf{cl}$ satisfies the inequality $2 \leq \mathsf{cl}(x_i) - \mathsf{cl}(y_j) \leq bc(G)$ for $x_i y_j \in E$. Moreover, $\mathsf{cl}(x_i) - \mathsf{cl}(y_j) = bc(G)$ for $x_i y_j \in E$ if and only if $i = m$ and $j = 1$.*

*Proof.* Since every vertex in $X$ (or $Y$) receives distinct labels, every pair of vertices distance two apart have distinct labels.

Then we show that $\mathsf{cl}(x_i) - \mathsf{cl}(y_j) \geq 2$ if $x_i y_j \in E$. Suppose to the contrary that $\mathsf{cl}(x_i) - \mathsf{cl}(y_j) \leq 1$. Then $(k - m + i) - (j - 1) \leq 1$, where $k = bc(G)$. Hence $k \leq m - i + j$. On the other hand, the set of $(m - i + 1) + j$ vertices $\{x_i, x_{i+1}, \ldots, x_m\} \cup \{y_1, y_2, \ldots, y_j\}$ is a biclique. Thus we obtain $k \geq (m - i + 1) + j$. This contradicts the inequality $k \leq m - i + j$.

Since $\lambda(G) \geq bc(G)$ and $\max_{v \in X \cup Y} \mathsf{cl}(v) = \mathsf{cl}(x_m) = bc(G)$, the labeling $f$ is an optimal $L(2,1)$-labeling. □

**Lemma 3.** *$bc(G) = \max_{1 \leq j \leq n} \{j + \deg y_j\}$ for a chain graph $G$.*

*Proof.* This can be derived easily from the fact that $N_G(x_1) \subseteq \cdots \subseteq N_G(x_m)$ and $N_G(y_n) \subseteq \cdots \subseteq N_G(y_1)$. □

We present an algorithm for computing the biclique number and an optimal labeling for a chain graph in Algorithm 1 and 2. Clearly, this algorithm runs in linear time.

**Theorem 2.** *An optimal $L(2,1)$-labeling of a chain graph can be computed in $O(N)$ time, where $N$ is the number of vertices.*

An example of the $L(2,1)$-labeling $\mathsf{cl}$ obtained by LABELING_CHAIN($G$) is illustrated in Fig. 3. The chain graph $G$ with $|X| = 7$ and $|Y| = 6$ has the biclique number $bc(G) = \max_{y_j \in Y} \{j + \deg y_j\} = 3 + \deg y_3 = 9$ (in fact, the set $\{x_2, \ldots, x_7\} \cup \{y_1, y_2, y_3\}$ forms the maximum biclique).

## 4 Labeling of Bipartite Permutation Graphs

In this section, we present a polynomial time algorithm for calculating an $L(2,1)$-labeling $f$ for a bipartite permutation graph $G$ so that $\max f(v) \leq bc(G) + 1$.

**Algorithm 1**: BICLIQUE_CHAIN($G$)

**Input**: a chain graph $G = (X, Y, E)$ with $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$.
**Output**: the biclique number $bc(G)$
$bc \leftarrow 0$;
**for** $j \leftarrow 1$ **to** $n$ **do**
  **if** $bc < j + \deg y_j$ **then** $bc \leftarrow j + \deg y_j$
**end**
**return** $bc$

**Algorithm 2**: LABELING_CHAIN($G$)

**Input**: a chain graph $G = (X, Y, E)$ with $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$.
**Output**: an $L(2,1)$-labeling $\mathsf{cl}$ of $G$
$bc \leftarrow$ BICLIQUE_CHAIN($G$) ; `/* the biclique number of G */`
**foreach** $x_i \in X$ **do** $\mathsf{cl}(x_i) \leftarrow bc - m + i$;
**foreach** $y_j \in Y$ **do** $\mathsf{cl}(y_j) \leftarrow j - 1$;
**return** $\mathsf{cl}$

**Definition 2.** *Let $G = (X, Y, E)$ be a bipartite permutation graph. For $y_j \in Y$, let $G_j$ be the subgraph of $G$ induced by $X_j \cup Y_j$, where*

$$X_j = N_G(y_j) = \{x_i, x_{i+1}, \ldots, x_k\}, \text{ and}$$
$$Y_j = \{y_l \mid x_k y_l \in E \text{ and } l \geq j\}.$$

**Lemma 4.** *$G_j$ is a chain graph such that $N_{G_j}(x_i) \subseteq N_{G_j}(x_{i+1}) \subseteq \cdots \subseteq N_{G_j}(x_k)$ and $N_{G_j}(y_l) \subseteq \cdots \subseteq N_{G_j}(y_{j+1}) \subseteq N_{G_j}(y_j)$, where $y_l$ is the maximum neighbor of $x_k$.*

*Proof.* It is easy to see that the vertices in $G_j$ are arranged such that $x_i < x_{i+1} < \cdots < x_k < y_j < y_{j+1} < \cdots < y_l$ on $L_1$ of the permutation diagram, and $y_j < x_i$ and $y_l < x_k$ on $L_2$. By Lemma 1, the lemma holds. □

### 4.1 Algorithm

The outline of our algorithm for a bipartite permutation graph $G$ is as follows:

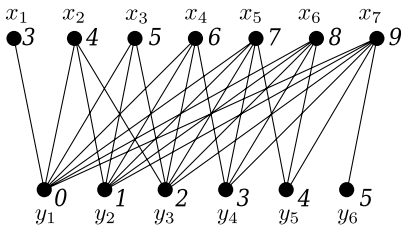



**Fig. 3.** A chain graph $G$ with $bc(G) = 9$ and its optimal $L(2,1)$-labeling.

**Algorithm 3**: LABELING_BIPARTITE_PERMUTATION($G$)

**Input**: a bipartite permutation graph $G = (X, Y, E)$.
**Output**: an $L(2,1)$-labeling $f$ of $G$

```
1  foreach v ∈ X ∪ Y do label(v) ← undef;
2  x_max ← x_0 ;          /* the maximum vertex in X s.t. label(x) ≠ undef */
3  y_max ← y_0 ;          /* the maximum vertex in Y s.t. label(y) ≠ undef */
4  r ← 1;
5  while r ≤ n do
6      if max N_G(y_r) > x_max then
7          Construct the chain graph G_r ;                  /* Definition 2 */
8          cl ← LABELING_CHAIN(G_r);
9          if r = 1 then s ← 0;
10         else s ← max{0, label(x_max) − cl(x_max), label(y_max) − cl(y_max)} ;
           /* X_r ∪ Y_r is the bipartition of G_r                          */
11         foreach x ∈ X_r do
12             if label(x) = undef then label(x) ← cl(x) + s
13         if s = 0 or s = label(y_max) − cl(y_max) then
14             foreach y ∈ Y_r do
15                 if label(y) = undef then label(y) ← cl(y) + s
16         else
               /* label(x_max) − cl(x_max) > label(y_max) − cl(y_max)      */
17             foreach y ∈ Y_r do label(y) ← cl(y) + s
18         x_max ← max N_{G_r}(y_r);
19         y_max ← max N_{G_r}(x_max);
20     r ← r + 1
21 foreach v ∈ X ∪ Y do f(v) ← label(v) mod (bc(G) + 2) ;
22 return f
```

1. Visit $y_r \in Y$ starting from $r = 1$ to $r = n$ consecutively.
   (a) Construct a chain graph $G_r$ and calculate a labeling $\mathsf{cl}$ of $G_r$ by LABELING_CHAIN.
   (b) Determine the $L(2,1)$-labeling $\mathsf{label}(v)$ of vertices $v$ in $G_r$ by *adjusting* $\mathsf{cl}$ to already assigned labels of $G$.
2. After the assignment $\mathsf{label}$ are determined for all vertices, calculate $f = \mathsf{label}(v) \bmod (bc(G) + 2)$, and output the resulting label assignment $f$.

The detail of the algorithm is described in Algorithm 3.

### 4.2 Example of Our Algorithm

We present Figs. 4–8 as an example of our labeling algorithm for the bipartite permutation graph $G$ of Fig. 1.

1. In Fig. 4, the chain graph $G_1$ and its labeling $\mathsf{cl}$ are calculated. Then $\mathsf{label}$ for vertices in $G_1$ is defined.

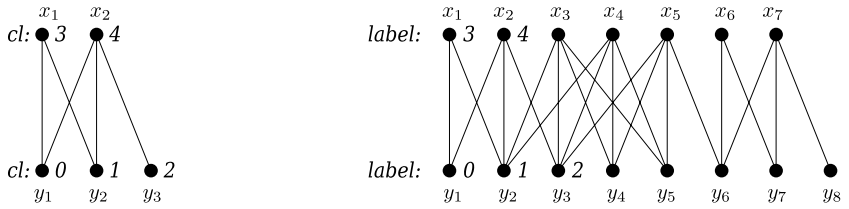



**Fig. 4.** The chain graph $G_1$ and its labeling cl (left), and the labeling label of $G$ (right). In this case, $s = 0$.

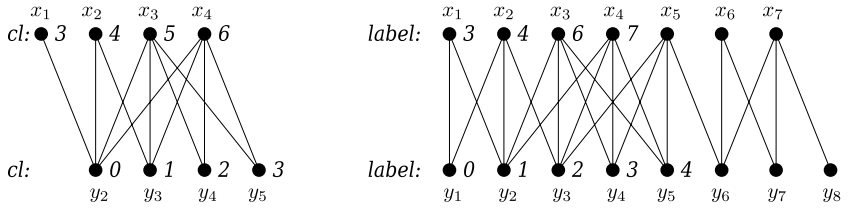



**Fig. 5.** $G_2$ and its labeling cl, and label of $G$ ($s = 1$).

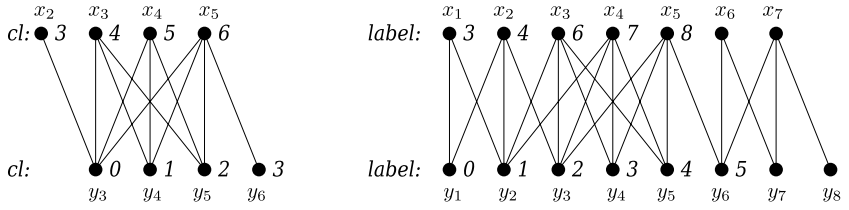



**Fig. 6.** $G_3$ and its labeling cl, and label of $G$ ($s = 2$).

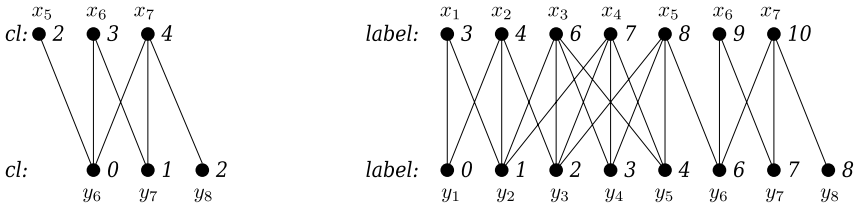



**Fig. 7.** $G_6$ and its labeling cl, and label of $G$ ($s = 6$).

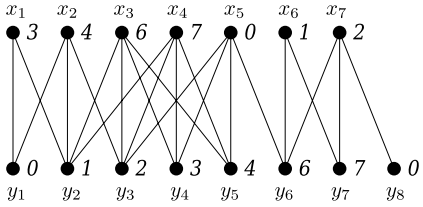



**Fig. 8.** The $L(2,1)$-labeling of $G$ which is obtained from label and $bc(G) + 2 = 8$.

2. The chain graph $G_2$ and its labeling $\mathsf{cl}$ are obtained as in Fig. 5. In this case, $s = \max\{0, \mathsf{label}(x_2) - \mathsf{cl}(x_2), \mathsf{label}(y_3) - \mathsf{cl}(y_3)\} = \max\{0, 4-4, 2-1\} = 1$. Thus $\mathsf{label}(v) = \mathsf{cl}(v) + 1$ for $v \in \{x_3, x_4, y_4, y_5\}$.
3. The chain graph $G_3$ and its labeling $\mathsf{cl}$ are obtained as in Fig. 6. In this case, $s = \max\{0, \mathsf{label}(x_4) - \mathsf{cl}(x_4), \mathsf{label}(y_5) - \mathsf{cl}(y_5)\} = \max\{0, 7-5, 4-2\} = 2$. Thus $\mathsf{label}(v) = \mathsf{cl}(v) + 2$ for $v \in \{x_5, y_6\}$.
4. The chain graph $G_6$ and its labeling $\mathsf{cl}$ are obtained as in Fig. 7. In this case, $s = \max\{0, \mathsf{label}(x_5) - \mathsf{cl}(x_5), \mathsf{label}(y_6) - \mathsf{cl}(y_6)\} = \max\{0, 8-2, 5-0\} = 6$. Since $s = 6 = \mathsf{label}(x_5) - \mathsf{cl}(x_5) > \mathsf{label}(y_6) - \mathsf{cl}(y_6)\}$, $\mathsf{label}(v) = \mathsf{cl}(v) + 6$ for $v \in \{x_6, x_7, y_6, y_7, y_8\}$ (line 17 of Algorithm 3).
5. Finally, $L(2,1)$-labeling $f$ of $G$ is obtained by $f(v) = \mathsf{label}(v) \bmod 8$ as shown in Fig. 8.

Note that, in each step, the biclique number $bc(G_r)$ is equal to the value of $\mathsf{label}(x) - \mathsf{label}(y_r)$, where $x$ is the maximum neighbor of $y_r$. For example, in Fig. 5, $bc(G_2) = 6 = \mathsf{label}(x_4) - \mathsf{label}(y_2)$ holds.

It also should be noted that if the condition $s = \mathsf{label}(x_{\max}) - \mathsf{cl}(x_{\max}) > \mathsf{label}(y_{\max}) - \mathsf{cl}(y_{\max})$ holds (line 17), the labeling of nodes of $Y$ in $G_r$ are increased. For example, $\mathsf{label}(y_6) = 5$ in Fig. 6, then it is increased to 6 in Fig. 7 because the situation $s = \mathsf{label}(x_5) - \mathsf{cl}(x_5) > \mathsf{label}(y_6) - \mathsf{cl}(y_6)$ occurs.

### 4.3 Correctness

The labeling $\mathsf{label}$ calculated in the algorithm is an $L(2,1)$-labeling of $G$, which is guaranteed by the following two lemmas.

**Lemma 5.** $2 \leq \mathsf{label}(x_i) - \mathsf{label}(y_j) \leq bc(G)$ *if* $x_i y_j \in E$.

*Proof (Sketch of proof).* If an edge $x_i y_j$ is in $G_r$, then $\mathsf{cl}(x_i) - \mathsf{cl}(y_j) \leq bc(G_r)$ by Lemma 1, where $\mathsf{cl}$ is the labeling of $G_r$. Since $\mathsf{label}(x_i) - \mathsf{label}(y_j) \leq \mathsf{cl}(x_i) - \mathsf{cl}(y_j)$, the inequality $\mathsf{label}(x_i) - \mathsf{label}(y_j) \leq bc(G)$ holds.

So we should show that $\mathsf{label}(x_i) - \mathsf{label}(y_j) \geq 2$. This condition would be violated only when the following situation occurs:

(i) $\mathsf{label}(y_j)$ is increased in line 17 for some chain graph $G_r$, and
(ii) The labels of vertices of $X_r$ in $G_r$ are not consecutive numbers.

An example of non-consecutive labels is $G_3$ in Fig. 6. The vertices of $X_3$, $x_2, x_3, x_4$ and $x_5$, have labels 4,6,7 and 8, respectively, in $G$, which are not consecutive numbers. Furthermore, vertex $x_2$ is adjacent to $y_3$ and $\mathsf{label}(x_2) - \mathsf{label}(y_3) = 2$. Thus, if $\mathsf{label}(y_3)$ would be increased after processing $G_3$, then $\mathsf{label}(x_2) - \mathsf{label}(y_3) < 2$.

However, we can show that the above situation (i) and (ii) does not occur simultaneously. The detailed proof of this will be presented in the full version of this paper. □

**Lemma 6.** *The labeling* $\mathsf{label}$ *satisfies the following inequalities:*

1. $1 \leq \mathsf{label}(x_k) - \mathsf{label}(x_i) \leq bc(G) - 2$ *if* $\mathrm{dist}(x_i, x_k) = 2$ *and* $1 \leq i < k \leq m$.
2. $1 \leq \mathsf{label}(y_l) - \mathsf{label}(y_j) \leq bc(G) - 2$ *if* $\mathrm{dist}(y_j, y_l) = 2$ *and* $1 \leq j < l \leq n$.

*Proof.* Suppose that $\mathrm{dist}(x_i, x_k) = 2$ and $i < k$. Clearly $\mathsf{label}(x_i) < \mathsf{label}(x_k)$. Let $y$ be a common neighbor of $x_i$ and $x_k$, and $q = \mathsf{label}(y)$. By Lemma 5, we have $\mathsf{label}(x_k) \leq q + bc(G)$ and $\mathsf{label}(x_i) \geq q + 2$. Hence $\mathsf{label}(x_k) - \mathsf{label}(x_i) \leq bc(G) - 2$.

Similarly, we suppose that $\mathrm{dist}(y_j, y_l) = 2$ and $j < l$. Clearly $\mathsf{label}(y_j) < \mathsf{label}(y_l)$. Let $x$ be a common neighbor of $y_j$ and $y_l$, and $p = \mathsf{label}(x)$. By Lemma 5, we have $\mathsf{label}(y_l) \leq p - 2$ and $\mathsf{label}(y_j) \geq p - bc(G)$. Hence $\mathsf{label}(y_l) - \mathsf{label}(y_j) \leq bc(G) - 2$. □

**Theorem 3.** *The labeling $f$ calculated by Algorithm 3 is an $L(2,1)$-labeling of $G$, and $\max_{v \in X \cup Y} f(v) \leq bc(G) + 1$. This algorithm runs in $O(|V| + |E|)$ time.*

*Proof.* Since $f(v) = \mathsf{label}(v) \bmod (bc(G) + 2)$, the inequality $\max_{v \in X \cup Y} f(v) \leq bc(G) + 1$ holds.

Let $xy \in E$, where $x \in X$ and $y \in Y$. Then, by Lemma 5, $2 \leq \mathsf{label}(x) - \mathsf{label}(y) \leq bc(G)$. Since $f(x) = \mathsf{label}(x) \bmod (bc(G)+2)$ and $f(y) = \mathsf{label}(y) \bmod (bc(G) + 2)$, the value of $|f(x) - f(y)|$ cannot be 0 or 1.

If $\mathrm{dist}(x_i, x_k) = 2$, then $\mathsf{label}(x_k) - \mathsf{label}(x_i) \leq bc(G) - 2$ by Lemma 6. Hence $|f(x_k) - f(x_i)| \geq 1$. Similarly, we can show that $|f(y_l) - f(y_j)| \geq 1$ if $\mathrm{dist}(y_j, y_l) = 2$.

If the degree of $y_j$ and $\max N_G(y_j)$ are $d_1$ and $d_2$, respectively, then the chain graph $G_j$ has at most $d_1 + d_2$ vertices. Hence, $\mathsf{cl}$ and $\mathsf{label}$ of vertices in $G_j$ are calculated in $O(d_1 + d_2) = O(\Delta)$ time, where $\Delta$ is the maximum degree of $G$. Since the number of chain graphs constructed in our algorithm is $O(|V|)$, the total running time of the algorithm is $O(|V| + \Delta|V|) = O(|V| + |E|)$. □

**Corollary 2.** *Any bipartite permutation graph $G$ satisfies $\lambda(G) \leq bc(G) + 1$.*

## 5 Conclusion

In this paper, we investigated the $L(2,1)$-labeling problem for bipartite permutation graphs. We showed that an optimal $L(2,1)$-labeling of a chain graph, a special class of bipartite permutation graphs, can be computed in linear time. We also present a linear time algorithm for computing $L(2,1)$-labeling of a bipartite permutation graph such that the maximum label is at most $bc(G) + 1$. Since $\lambda(G) \geq bc(G)$ for any bipartite graph $G$, our algorithm computes a nearly optimal solution.

We conclude this paper by presenting two open problems.

It should be noted that there exists a bipartite permutation graph $G$ such that $\lambda(G) = bc(G) + 1$. For example, the bipartite permutation graph $G$ in Fig. 9 has $bc(G) = 6$ and $\lambda(G) = 7 = bc(G) + 1$. Hence the set of bipartite permutation graphs is classified into two classes, one consists of $G$ with $\lambda(G) = bc(G)$ and another consists of $G$ with $\lambda(G) = bc(G) + 1$.

*Problem 1.* Characterize bipartite permutation graph $G$ with $\lambda(G) = bc(G)$.

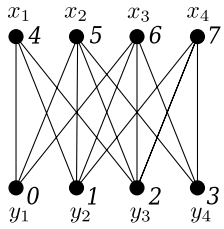



**Fig. 9.** A bipartite permutation graph $G$ with $\lambda(G) = bc(G) + 1$.

Our algorithm does not guarantee that the resulting labeling is optimal. The complexity of the $L(2,1)$-labeling problem for bipartite permutation graphs is still open.

*Problem 2.* Develop a polynomial time algorithm for computing an optimal $L(2,1)$-labeling of a bipartite permutation graph, or prove NP-completeness of the problem for bipartite permutation graphs.

## References

1. Griggs, J.R., Yeh, R.K.: Labelling graphs with a condition at distance 2. SIAM Journal on Discrete Mathematics **5**(4) (nov 1992) 586–595
2. Calamoneri, T.: The $L(h,k)$-labelling problem: A survey and annotated bibliography. The Computer Journal **49**(5) (2006) 585–608
3. Yeh, R.K.: A survey on labeling graphs with a condition at distance two. Discrete Mathematics **306** (2006) 1217–1231
4. Bodlaender, H.L., Kloks, T., Tan, R.B., van Leeuwen, J.: Approximations for $\lambda$-coloring of graphs. The Computer Journal **47**(2) (2004) 193–204
5. Spinrad, J., Brandstädt, A., Stewart, L.: Bipartite permutation graphs. Discrete Applied Mathematics **18**(3) (1987) 279–292
6. Xu, G., Kang, L., Shan, E.: Acyclic domination on bipartite permutation graphs. Information Processing Letters **99** (2006) 139–144
7. Lu, C.L., Tang, C.Y.: Solving the weighted efficient edge domination problem on bipartite permutation graphs. Discrete Applied Mathematics **87** (1998) 203–211
8. Steiner, G.: On the $k$-path partition of graphs. Theoretical Computer Science **290** (2003) 2147–2155
9. Uehara, R., Valiente, G.: Linear structure of bipartite permutation graphs and the longest path problem. Information Processing Letters **103** (2007) 71–77
10. Golumbic, M.C.: Algorithmic graph theory and perfect graphs. 2nd edition. Elsevier B.V. (2004)
11. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: A survey. SIAM (1999)

# Smaller and Faster Lempel-Ziv Indices ⋆

Diego Arroyuelo and Gonzalo Navarro

Dept. of Computer Science, Universidad de Chile, Chile.
{darroyue,gnavarro}@dcc.uchile.cl

**Abstract.** Given a text $T[1..u]$ over an alphabet of size $\sigma = O(\text{polylog}(u))$ and with $k$-th *order empirical entropy* $H_k(T)$, we propose a new *compressed full-text self-index* based on the Lempel-Ziv (LZ) compression algorithm, which replaces $T$ with a representation requiring about three times the size of the compressed text, i.e $(3+\epsilon)uH_k(T) + o(u \log \sigma)$ bits, for any $\epsilon > 0$ and $k = o(\log_\sigma u)$, and in addition gives indexed access to $T$: it is able to locate the *occ* occurrences of a pattern $P[1..m]$ in the text in $O((m + occ) \log u)$ time. Our index is smaller than the existing indices that achieve this locating time complexity, and locates the occurrences faster than the smaller indices. Furthermore, our index is able to count the pattern occurrences in $O(m)$ time, and it can extract any text substring of length $\ell$ in optimal $O(\ell/\log_\sigma u)$ time. Overall, our indices appear as a very attractive alternative for space-efficient indexed text searching.

## 1 Introduction

With the huge amount of text data available nowadays, the *full-text searching* problem plays a fundamental role in modern computer applications. Full-text searching consists of finding the *occ* occurrences of a given pattern $P[1..m]$ in a text $T[1..u]$, where both $P$ and $T$ are sequences over an alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$. Unlike *word-based* text searching, we wish to find any *text substring*, not only whole words or phrases. This has applications in texts where the concept of *word* is not well defined (e.g., Oriental languages), or texts where words do not exist at all (e.g., DNA, protein, and MIDI pitch sequences).

We assume that the text is large and known in advance to queries, and we need to perform several queries on it. Therefore, we can construct an *index* on the text, which is a data structure allowing efficient access to the pattern occurrences, yet increasing the space requirement. Our main goal is *to provide fast access to the text using as little space as possible*. Classical full-text indices, like *suffix trees* and *suffix arrays*, have the problem of a high space requirement: they require $O(u \log u)$ and $u \log u$ bits respectively, which in practice is about 10 and 4 times the text size, not including the text.

*Compressed self-indexing* is a recent trend in full-text searching, which consists in developing full-text indices that store enough information so as to search

⋆ Supported in part by CONICYT PhD Fellowship Program (first author), and Fondecyt Grant 1-050493 (second author).

and retrieve any part of the indexed text without storing the text itself, while requiring space proportional to the compressed text size. Because of their compressed nature and since the text is replaced by the index, typical compressed self-indices are much smaller than classical indices, allowing one to store indices of large texts entirely in main memory, in cases where a classical index would have required to access the much slower secondary storage. There exist two classical kind of queries, namely: (*1*) $\mathsf{count}(T, P)$, which counts the number of occurrences of $P$ in $T$; (*2*) $\mathsf{locate}(T, P)$, which reports the starting positions of the *occ* occurrences of $P$ in $T$. Self-indices also need operation (*3*) $\mathsf{extract}(T, i, j)$, which decompresses substring $T[i..j]$, for any text positions $i \leqslant j$.

Let $H_k(T)$ denote the $k$-th order empirical entropy of a sequence of symbols $T$ [12]. The value $uH_k(T)$ provides a lower bound to the number of bits needed to compress $T$ using any compressor that encodes each symbol considering only the context of $k$ symbols that precede it in $T$. It holds that $0 \leqslant H_k(T) \leqslant H_{k-1}(T) \leqslant \cdots \leqslant H_0(T) \leqslant \log \sigma$ (by log we mean $\log_2$ in this paper).

The main types of compressed self-indices [16] are *Compressed Suffix Arrays* (CSA) [8, 19], indices based on *backward search* [6] (which are alternative ways to compress suffix arrays), and the indices based on the *Lempel-Ziv* compression algorithm (LZ-indices for short) [10]. LZ-indices have shown to be very effective in practice for locating occurrences and extracting text, outperforming other compressed indices. Compressed indices based on suffix arrays store extra non-compressible information to carry out these tasks, whereas the extra data stored by LZ-indices is compressible. Therefore, when the texts are highly compressible, LZ-indices can be smaller and faster than alternative indices; and in other cases they offer very attractive space/time trade-offs.

What characterizes the particular niche of LZ-indices is the $O(uH_k(T))$ space combined with $O(\log u)$ time per located occurrence. The smallest LZ-indices available are those by Arroyuelo et al. [1] (ANS-LZI for short), which require $(2+\epsilon)uH_k(T) + o(u \log \sigma)$ bits of space, for any constant $0 < \epsilon < 1$ and any $k = o(\log_\sigma u)$; however, their time complexity of $O(m^2 \log m + (m + occ) \log u)$ makes them suitable only for short patterns, when the quadratic term is less significant. Other LZ-indices, like a compact version of that by Ferragina and Manzini [6], or the one by Russo and Oliveira [18] (ILZI for short), achieve $O((m + occ) \log u)$ time. They are, however, relatively large, at least $5uH_k(T) + o(u \log \sigma)$ bits of space.

In this paper we propose a new LZ-index scheme requiring $(3 + \epsilon)uH_k(T) + o(u \log \sigma)$ bits of space, for $\sigma = O(\text{polylog}(u))$ and any $k = o(\log_\sigma u)$, with an overall locating time of $O((m + occ) \log u)$, a counting time of $O(m)$, and an extracting time of $O(\ell / \log_\sigma u)$ for a substring of length $\ell$. In this way we achieve the same locating complexity of larger LZ-indices [6, 18]. Note that the original index in [6] achieves better locate time, $O(m + occ)$, yet it requires $O(uH_k(T) \log^\gamma)$ bits of space, for any $\gamma > 0$.

The CSA of Sadakane [19] (SAD-CSA for short) has a locating complexity of $O((m + occ) \log^\epsilon u)$, for any $\epsilon > 0$, however the space requirement is proportional to the zero-th order empirical entropy plus a non-negligible extra space,

$\epsilon^{-1}uH_0(T) + O(u \log\log\sigma)$ bits. The *Alphabet-Friendly FM-index* [7] (AF-FMI for short), on the other hand, requires $uH_k(T) + o(u\log\sigma)$ bits of space, however its locate complexity is $O(m + occ \log^{1+\epsilon} u)$, which is slower than ours. Finally, the CSA of Grossi, Gupta, and Vitter [8] (GGV-CSA for short) requires $\epsilon^{-1}uH_k(T) + o(u\log\sigma)$ bits of space, with a locating time of $O((\log u)^{\frac{\epsilon}{1-\epsilon}}(\log\sigma)^{\frac{1-2\epsilon}{1-\epsilon}})$ per occurrence, after a counting time of $O(\frac{m}{\log_\sigma u} + (\log u)^{\frac{1+\epsilon}{1-\epsilon}}(\log\sigma)^{\frac{1-3\epsilon}{1-\epsilon}})$, where $0 < \epsilon < 1/2$. When $\epsilon$ approaches $1/3$, the space requirement is about $3uH_k(T) + o(u\log\sigma)$ bits, with a locating time of $O(\frac{m}{\log_\sigma u} + \log^2 u + occ((\log u)^{1/2}(\log\sigma)^{1/2}))$. Thus, using the same space their time per occurrence located is slightly lower, in exchange for an $O(\log^2 u)$ extra additive factor.

In Table 1 we summarize the space and time complexities of some existing compressed self-indices. Locate times in the table are per occurrence reported, after counting the pattern occurrences. In the case of our LZ-index, we have to pay extra $O(m \log u)$ time.

**Table 1.** Comparison of our LZ-index with alternative compressed self-indices. The result for the GGV-CSA [8] is shown for $\epsilon = 1/3$, and assuming $\sigma = O(\text{polylog}(u))$ in all cases.

| Index | Space in bits |
|---|---|
| SAD-CSA [19] | $\epsilon^{-1}uH_0(T) + O(u\log\log\sigma)$ |
| GGV-CSA [8] | $3uH_k(T) + o(u\log\sigma)$ |
| AF-FMI [7] | $uH_k(T) + o(u\log\sigma)$ |
| ANS-LZI [1] | $(2+\epsilon)uH_k(T) + o(u\log\sigma)$ |
| RO-LZI [18] | $(5+\epsilon)uH_k(T) + o(u\log\sigma)$ |
| Our LZI | $(3+\epsilon)uH_k(T) + o(u\log\sigma)$ |

| Index | `count` | `locate` (per occurrence) | `extract` |
|---|---|---|---|
| [19] | $O(m\log u)$ | $O(\log^\epsilon u)$ | $O(\ell + \log^\epsilon u)$ |
| [8] | $O(\frac{m}{\log_\sigma u} + \log^2 u)$ | $O(\log^{1/2} u \log^{1/2}\sigma)$ | $O(\log^{1/2} u \log^{1/2}\sigma + \ell/\log_\sigma u)$ |
| [7] | $O(m)$ | $O(\log^{1+\epsilon} u)$ | $O(\ell + \log^{1+\epsilon} u)$ |
| [1] | $O((m+occ)\log u)$ | free after counting | $O(\ell/\log_\sigma u)$ |
| [18] | $O((m+occ)\log u)$ | free after counting | $O(\ell/\log_\sigma u)$ |
| Our LZI | $O(m)$ | $O(\log u)$ | $O(\ell/\log_\sigma u)$ |

## 2 Searching in Lempel-Ziv Compressed Texts

Assume that the text $T[1..u]$ has been compressed using the `LZ78` algorithm [21] into $n+1$ phrases, $T = B_0 \dots B_n$, such that $B_0 = \varepsilon$. The search of a pattern $P[1..m]$ in a `LZ78`-compressed text has the additional problem that, as the text is parsed into phrases, a pattern occurrence can span several (two or more)

consecutive phrases. We call *occurrences of type 1* those occurrences contained in a single phrase (there are $occ_1$ occurrences of type 1), and *occurrences of type 2* those occurrences spanning two or more phrases (there are $occ_2$ occurrences of this type). Next we review the existing Lempel-Ziv self-indices. The first compressed index based on `LZ78` was that of Kärkkäinen and Ukkonen [10], which has a locating time $O(m^2 + (m + occ)\log u)$ and a space requirement of $O(uH_k(T))$ bits [16], plus the text. However, this is not a self-index.

Ferragina and Manzini [6] define the FM-LZI, a compressed self index based on the `LZ78` compression algorithm, requiring $O(uH_k(T)\log^\gamma u)$ bits of space, for any constant $\gamma > 0$. This index is able to report the *occ* pattern occurrences in optimal $O(m + occ)$ time. This is the *fastest* existing compressed self-index, achieving the same time complexity as suffix trees, yet requiring $o(u \log u)$ bits and without needing the text to operate. However, the extra $O(\log^\gamma u)$ factor makes this index large in practice. However, we can replace their data structure for range queries by that of Chazelle [4], such that the resulting version of FM-LZI requires $(5+\epsilon)uH_k(T) + o(u\log\sigma)$ bits of space, for any constant $0 < \epsilon < 1$ and any $k = o(\log_\sigma u)$, and is capable of locating the pattern occurrences in $O((m + occ)\log u)$ time.

The LZ-index of Navarro [15] (Nav-LZI for short) has a greater locate time than that of LZ-indices in general, yet the smallest existing LZ-index is a variant of the Nav-LZI: the index defined by Arroyuelo et al. [1] (ANS-LZI for short) requires $(2+\epsilon)uH_k(T) + o(u\log\sigma)$ bits of space, and its locate time is $O(m^2 \log m + (m + occ)\log u)$. Although the locating time per occurrence is $O(\log u)$ as for other LZ-indices, the $O(m^2 \log m)$ term makes the ANS-LZI attractive only for short patterns.

The key to achieve such a small space requirement is that the Nav-LZI (and therefore the ANS-LZI) do not use the concept of suffix arrays at all. Rather, the search is based only in an implicit representation of the text through the `LZ78` parsing of it: the *LZTrie*, which is the trie representing the `LZ78` phrases of $T$. As the text is scattered throughout the *LZTrie*, we have to distinguish between occurrences spanning two consecutive phrases (occurrences of type 2) and occurrences spanning more than two phrases (*occurrences of type 3*). For occurrences of type 3 we must consider the $O(m^2)$ possible substrings of $P$, search for all these strings in *LZTrie*, then form maximal concatenations of consecutive phrases, to finally check every candidate [15]. All of this takes $O(m^2 \log m)$ time.

The space of the ANS-LZI can be reduced to $(1+\epsilon)uH_k(T) + o(u\log\sigma)$ bits, with a locating time of $O(m^2)$ on average for patterns of length $m \geqslant 2\log_\sigma u$, yet without worst-case guarantees at search time.

Russo and Oliveira [18] discard the `LZ78` parsing of $T$ and use a so-called maximal parsing instead, which is constructed for the reversed text. In this way they avoid the $O(m^2)$ checks for the different pattern substrings. The resulting LZ-index (the RO-LZI) requires $(5+\epsilon)uH_k(T) + o(u\log\sigma)$ bits of space, for any constant $0 < \epsilon < 1$ and any $k = o(\log_\sigma u)$. The locating time of the index is $O((m + occ)\log u)$. As for all the previous LZ-indices, the extract time for any text substring of length $\ell$ is the optimal $O(\ell/\log_\sigma u)$.

## 3 Smaller and Faster Lempel-Ziv Indices

In the review of Section 2 we conclude that LZ-indices can be as small as to require $(2+\epsilon)uH_k(T)+o(u\log\sigma)$ bits of space yet with a locate time $O(m^2\log m+(m+occ)\log u)$, or we can achieve time $O((m+occ)\log u)$ for locate with a greater index requiring $(5+\epsilon)uH_k(T)+o(u\log\sigma)$ bits of space. On the other hand, we can be fast to count only using the FM-LZI: $O(m)$ counting time in the worst case. We show that we can be fast for all these operations with a significantly smaller LZ-index.

### 3.1 Index Definition

As we aim at a small index, we use the ANS-LZI as a base, specifically the version requiring $(1+\epsilon)uH_k(T)+o(u\log\sigma)$ bits of space, which is composed of:

- *LZTrie*: is the trie storing the LZ78 phrases $B_0,\ldots,B_n$ of $T$. As the set of LZ78 phrases is prefix-closed, this trie has exactly $n+1$ nodes. We represent the trie structure using the following data structures: (*1*) $par[0..2n]$: the tree shape of *LZTrie* represented using DFUDS [2], requiring $2n+o(n)$ bits of storage, allowing us to compute operations $parent_{lz}(x)$ (which gets the parent of node $x$), $child_{lz}(x,i)$ (which gets the $i$-th child of $x$), $subtreesize_{lz}(x)$ (which gets the size of the subtree of $x$, including $x$ itself), $depth_{lz}(x)$ (which gets the depth of $x$ in the trie) and $LA_{lz}(x,d)$ (a *level-ancestor query*, which gets the ancestor at depth $d$ of node $x$), both of which can be computed on DFUDS by using the idea of Jansson et al. [9], and finally $ancestor_{lz}(x,y)$ (which tells us whether $x$ is an ancestor of node $y$), all of them in $O(1)$ time. As in [1], we add the data structure of [20] to extract any text substring of length $\ell$ in optimal $O(\ell/\log_\sigma u)$ time, requiring only $o(u\log\sigma)$ extra bits. (*2*) $ids[0..n]$: is the preorder sequence of LZ78 phrase identifiers. Permutation $ids$ is represented using the data structure of Munro et al. [14] such that we can compute $ids$ in $O(1)$ time and its inverse permutation $ids^{-1}$ in $O(1/\epsilon)$ time, requiring $(1+\epsilon)n\log n$ bits for any constant $0<\epsilon<1$. (*3*) $letts[1..n]$: the array storing the edge labels of *LZTrie* according to a DFUDS traversal of the trie. We solve operation $child(x,\alpha)$ (which gets the child of node $x$ with label $\alpha\in\{1,\ldots,\sigma\}$) in constant time as follows (this is slightly different to the original approach [2]). Suppose that node $x$ has position $p$ within $par$. Let $k$ be the number of $\alpha$s up to position $p-1$ in $letts$, and let $p+i$ be the position of the $(k+1)$-th $\alpha$ in $letts$. If $p+i$ lies within positions $p$ and $p+degree(x)$, the child we are looking for is $child(x,i+1)$, which is computed in constant time over $par$; otherwise $x$ has no child labeled $\alpha$. If $\sigma=O(\text{polylog}(u))$, we represent $letts$ using the *wavelet tree* of [7] in order to compute $k$ and $p+i$ in constant time by using $rank_\alpha$ and $select_\alpha$ respectively, and requiring $n\log\sigma+o(n)$ bits of space. We can also retrieve the symbol corresponding to node $x$ (i.e., the symbol by which $x$ descend from its parent) in constant time by $letts[rank_\iota(par,p)-1]$. Sequence $letts$ is also used to get the symbols of the text for extract queries.

Overall, *LZTrie* requires $(1+\epsilon)n\log n + 2n + n\log\sigma + o(u\log\sigma)$ bits, which is $(1+\epsilon)uH_k(T) + o(u\log\sigma)$ bits [11], for any $k = o(\log_\sigma u)$.
- *RevTrie*: is the *PATRICIA* tree [13] of the reversed `LZ78` phrases of $T$. In this trie there could be internal nodes not representing any phrase. We call these nodes *empty*. We compress empty unary nodes, and so we only represent the empty non-unary nodes. As a result, the number of nodes in this trie is $n \leqslant n' \leqslant 2n$. The trie is represented using the DFUDS representation, requiring $2n' + n'\log\sigma + o(n') \leqslant 4n + 2n\log\sigma + o(n)$ bits of space.
- $R[1..n]$: a mapping from *RevTrie* preorder positions (for non-empty nodes) to *LZTrie* preorder positions, requiring $n\log n = uH_k(T) + o(u\log\sigma)$ bits.
- $TPos[1..u]$: a bit vector marking the $n$ phrase beginnings. We represent $TPos$ using the data structure of [17] for *rank* and *select* queries in $O(1)$ time and requiring $uH_0(TPos) + o(u) \leqslant n\log\log n + o(u) = o(u\log\sigma)$ bits of space[1].

We can compress the $R$ mapping [1], so as to require $o(u\log\sigma)$ bits, by adding *suffix links* to *RevTrie*, which are represented by function $\varphi$. $R(i)$ (seen as a function) can be computed in constant time by using $\varphi$ [1]. If we store the values of $\varphi$ in preorder (according to *RevTrie*), the resulting sequence can be divided into at most $\sigma$ strictly increasing subsequences, and hence it can be compressed using the $\delta$-code of Elias [5] such that its space requirement is $n\log\sigma$ bits in the worst case, which is $o(u\log\sigma)$. The overall space requirement of the three above data structures is $(1+\epsilon)uH_k + o(u\log\sigma)$ bits.

To avoid the $O(m^2\log m)$ term in the locating complexity, we must avoid occurrences of type 3 (which make the ANS-LZI slower). Hence we add the *alphabet friendly FM-index* [7] of $T$ (AF-FMI($T$) for short) to our index. By itself this self-index is able to search for pattern occurrences, requiring $uH_k(T) + o(u\log\sigma)$ bits of space. However, its locate time per occurrence is $O(\log^{1+\epsilon} u)$, for any constant $\epsilon > 0$, which is greater than the $O(\log u)$ time per occurrence of LZ-indices. As AF-FMI($T$) is based on the *Burrows-Wheeler transform* (BWT) [3] of $T$, it can be (conceptually) thought as the suffix array of $T$.

To find occurrences spanning several phrases we define *Range*, a data structure for 2-dimensional range searching in the grid $[1..u] \times [1..n]$. For each `LZ78` phrase with identifier $id$, for $0 < id \leqslant n$, assume that the *RevTrie* node for $id$ has preorder $j$, and that phrase $(id+1)$ starts at position $p$ in $T$. Then we store the point $(i, j)$ in *Range*, where $i$ is the lexicographic order of the suffix of $T$ starting at position $p$.

Suppose that we search for a given string $s_2$ in AF-FMI($T$) and get the interval $[i_1, i_2]$ in the BWT (equivalently, in the suffix array of $T$), and that the search for string $s_1^r$ in *RevTrie* yields a node such that the preorder interval for its subtree is $[j_1, j_2]$. Then, a search for $[i_1, i_2] \times [j_1, j_2]$ in *Range* yields all phrases ending with $s_1$ such that in the next phrase starts an occurrence of $s_2$.

We transform the grid $[1..u] \times [1..n]$ indexed by *Range* to the equivalent grid $[1..n] \times [1..n]$ by defining a bit vector $\mathcal{V}[1..u]$, indicating (with a `1`) which positions of AF-FMI($T$) index an `LZ78` phrase beginning. We represent $\mathcal{V}$ with the data

[1] $rank_1(TPos, i)$ is the number of `1`'s in $TPos$ up to position $i$. $select_1(TPos, j)$ yields the position of the $j$-th `1` in $TPos$.

structure of [17] allowing *rank* queries, and requiring $uH_0(\mathcal{V}) + o(u) = o(u \log \sigma)$ bits of storage. Thus, instead of storing the point $(i, j)$ as in the previous definition of *Range*, we store the point $(rank_1(\mathcal{V}, i), j)$. The same search of the previous paragraph now becomes $[rank_1(\mathcal{V}, i_1), rank_1(\mathcal{V}, i_2)] \times [j_1, j_2]$.

As there is only one point per row and column of *Range*, we can use the data structure of Chazelle [4], requiring $n \log n(1+o(1)) = uH_k(T) + o(u \log \sigma)$ bits of space and allowing us to find the $K$ points in a given two-dimensional range in time $O((K+1) \log n)$. As a result, the overall space requirement of our LZ-index is $(3+\epsilon)uH_k(T) + o(u \log \sigma)$, for any $k = o(\log_\sigma u)$ and any constant $0 < \epsilon < 1$.

### 3.2 Search Algorithm

Assume that $P[1..m] = p_1 \dots p_m$, for $p_i \in \Sigma$. We need to consider two types of occurrences of $P$ in $T$.

*Locating Occurrences of Type 1.* Assume that phrase $B_j$ contains $P$. If $B_j$ does not end with $P$ and if $B_j = B_\ell \cdot c$, for $\ell < j$ and $c \in \Sigma$, then by `LZ78` properties $B_\ell$ contains $P$ as well. Therefore we must find the shortest possible phrases containing $P$, which according to `LZ78` are all phrases ending with $P$. This work can be done by searching for $P^r$ in *RevTrie*. Say we arrive at node $v$. Any node $v'$ in the subtree of $v$ (including $v$ itself) corresponds to a phrase terminated with $P$. Thus we traverse and report all the subtrees of the *LZTrie* nodes $R(v')$ corresponding to each $v'$. Total locate time is $O(m + occ_1)$.

*Locating Occurrences of Type 2.* To find the pattern occurrences spanning two or more consecutive phrases we must consider the $m-1$ partitions $P[1..i]$ and $P[i+1..m]$ of $P$, for $1 \leqslant i < m$. For every partition we must find all phrases terminated with $P[1..i]$ such that the next phrase starts at the same position as an occurrence of $P[i+1..m]$ in $T$. Hence, as explained before, we must search for $P^r[1..i]$ in *RevTrie* and for $P[i+1..m]$ in AF-FMI($T$). Thus, every partition produces two one-dimensional intervals, one in each of the above structures.

The $m-1$ intervals in AF-FMI($T$) can be found in $O(m)$ time thanks to the *backward search* concept, since the process to count the number of occurrences of $P[2..m]$ proceeds in $m-1$ steps, each one taking constant time: in the first step we find the BWT interval for $p_m$, then we find the interval for occurrences of $p_{m-1}p_m$, and so on to finally find the interval for $p_2 \dots p_m = P[2..m]$. However, the work in *RevTrie* can take time $O(m^2)$ if we search for strings $P^r[1..i]$ separately, as done for the ANS-LZI. Fortunately, some work done to search for a given $P^r[1..i]$ can be reused to search for other strings.

We have to search for strings $p_{m-1}p_{m-2} \dots p_1$; $p_{m-2} \dots p_1$;…; and $p_1$ in *RevTrie*. Note that every $p_j \dots p_1$ is the longest proper suffix of $p_{j+1}p_j \dots p_1$. Suppose that we successfully search for $P^r[1..m-1] = p_{m-1}p_{m-2} \dots p_1$, reaching the node with preorder $i$ in *RevTrie*, hence finding the corresponding preorder interval in *RevTrie* in $O(m)$ time. Now, to find the node representing suffix $p_{m-2} \dots p_1$ we only need to follow suffix link $\varphi(i)$ (which takes constant time) instead of searching for it from the *RevTrie* root (which would take $O(m)$ time

again). The process of following suffix links can be repeated $m-1$ times up to reaching the node corresponding to string $p_1$, with total time $O(m)$. This is the main idea to get the $m-1$ preorder intervals in *RevTrie* in time less than quadratic. The general case is slightly more complicated and corresponds to the *descend and suffix walk* method used in the RO-LZI [18]. In the sequel we explain the way we implement descend and suffix walk in our data structure.

We first prove a couple of properties. First, we know that every non-empty node in *RevTrie* has a suffix link [1], yet we need to prove that every *RevTrie* node (including empty-non-unary nodes) has also a suffix link.

*Property 1.* Every empty non-unary node in *RevTrie* has a suffix link.

*Proof.* Assume that node with preorder $i$ in *RevTrie* is empty non-unary, and that it represents string $\mathtt{ax}$, for $\mathtt{a} \in \Sigma$ and $\mathtt{x} \in \Sigma^*$. As node $i$ is an empty non-unary node, the node has at least two children. In other words, there exist at least two strings of the form $\mathtt{axy}$ and $\mathtt{axz}$, for $\mathtt{y}, \mathtt{z} \in \Sigma^*$, $\mathtt{y} \neq \mathtt{z}$, both strings corresponding to non-empty nodes, and hence these nodes have a suffix link. These suffix links correspond to strings $\mathtt{xy}$ and $\mathtt{xz}$ in *RevTrie*. Thus, it must exist a non-unary node for string $\mathtt{x}$, so every empty node $i$ has a suffix link. □

We store the $n' \leqslant 2n$ suffix links $\varphi$ in preorder (as explained before), requiring $2n \log \sigma$ bits of space in the worst case, which is $o(u \log \sigma)$.

The second property is that, although *RevTrie* is a PATRICIA tree and hence we store only the first symbol of each edge label, we can get all of it.

*Property 2.* Any edge label in *RevTrie* can be extracted in optimal time.

*Proof.* To extract the label for edge $e_{ij}$ between nodes with preorder $i$ and $j$ in *RevTrie*, note that the length of the edge label can be computed as $depth_{lz}(R[j]) - depth_{lz}(R[i])$. We can access the node from where to start the extraction by $x = LA_{lz}(R[j], depth_{lz}(R[j]) - depth_{lz}(R[i]))$, in constant time. The label of $e_{ij}$ is the label of the root-to-$x$ path (read backwards). □

In this way, every time we arrive to a *RevTrie* node, the string represented by that node will match the corresponding prefix of the pattern.

Previously we show that it is possible to search for all strings $P^r[1..i]$ in time $O(m)$, assuming that $P^r[1..m-1]$ exists in *RevTrie* (therefore all $P^r[1..i]$ exist in *RevTrie*). The general case is as follows. Suppose that, searching for $p_{m-1}p_{m-2}\ldots p_1$, we arrive at a node with preorder $i$ in *RevTrie* (hereafter node $i$), and we try to descend to a child node with preorder $j$ (hereafter node $j$). Assume that node $i$ represents string $\mathtt{ax}$, for $\mathtt{a} \in \Sigma$ and $\mathtt{x} \in \Sigma^*$. According to Property 2, we are sure that $\mathtt{ax} = p_{m-1}\ldots p_t$, for some $1 \leqslant t \leqslant m-1$. Assume also that edge $e_{ij}$ between nodes $i$ and $j$ is labeled $\mathtt{yz}$, for $\mathtt{y}, \mathtt{z} \in \Sigma^*$, $\mathtt{z} = z_1 \ldots z_q$. If we discover that $p_{t-1}\ldots p_k = \mathtt{y}$ and $p_{k-1} \neq z_1$, then this means that symbol $z_1$ in the edge label differs from the corresponding symbol $p_{k-1}$ in $P^r[1..m-1]$, and so we cannot descend to node $j$. This means that there are no phrases ending with $P^r[1..m-1]$, and we go on to consider $P^r[1..m-2]$. To

reuse the work done up to node $i$, we follow the suffix link to get the node $\varphi(i)$, and from this node we descend using $\mathtt{y} = p_{t-1} \dots p_k$. As this substring of $P$ has been already checked in the previous step, the descent is done checking only the first symbols of the labels, up to a node such that the next node in the path represents a string longer than $|\mathtt{xy}|$. At this point the descent is done as usual, extracting the edge labels and checking with the pattern symbols. In this way the total amortized time is $O(m)$.

If the search in *RevTrie* for $P^r[1..i]$ yields the preorder interval $[x, y]$, and the search for $P[i+1..m]$ in AF-FMI($T$) yields interval $[x', y']$, the two-dimensional range $[x', y'] \times [x, y]$ in *Range* yields all pattern occurrences for the given partition of $P$. For every pattern occurrence we get a point $(i', j')$. The corresponding phrase identifier can be found as $id = ids(R(j'))$, to finally compute the text position by $select_1(TPos, id+1) - i$. Overall, occurrences of type 2 are found in $O((m + occ_2) \log n)$ time.

For count queries we can achieve $O(m)$ time by just using the AF-FMI($T$). For extract queries we use the data structure of Sadakane and Grossi [20] in the *LZTrie* to extract any text substring $T[p..p + \ell]$ in optimal $O(\ell/\log_\sigma u)$ time: the identifier for the phrase containing position $p$ can be computed as $id = rank_1(TPos, p)$. Then, by using $ids^{-1}$ we compute the corresponding *LZTrie* node from where to extract the text.

We have proved:

**Theorem 1.** *There exists a compressed full-text self-index requiring* $(3+\epsilon)uH_k(T) + o(u \log \sigma)$ *bits of space, for* $\sigma = O(\text{polylog}(u))$*, any* $k = o(\log_\sigma u)$*, and any constant* $0 < \epsilon < 1$*, which is able to: report the occ occurrences of pattern* $P[1..m]$ *in text* $T[1..u]$ *in* $O((m + occ/\epsilon) \log u)$ *worst-case time; count pattern occurrences in* $O(m)$ *time; and extract any text substring of length* $\ell$ *in time* $O(\ell/(\epsilon \log_\sigma u))$.

## References


1. D. Arroyuelo, G. Navarro, and K. Sadakane. Reducing the space requirement of LZ-index. In *Proc. CPM*, pages 319–330, 2006.
2. D. Benoit, E. Demaine, I. Munro, R. Raman, V. Raman, and S.S. Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
3. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
4. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.
5. P. Elias. Universal codeword sets and representation of integers. *IEEE Trans. Inform. Theory*, 21(2):194–203, 1975.
6. P. Ferragina and G. Manzini. Indexing compressed texts. *Journal of the ACM*, 54(4):552–581, 2005.
7. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Transactions on Algorithms (TALG)*, 3(2):article 20, 2007.
8. R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. SODA*, pages 841–850, 2003.

9. J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees. In *Proc. SODA'07*, pages 575–584, 2007.
10. J. Kärkkäinen and E. Ukkonen. Lempel-Ziv parsing and sublinear-size index structures for string matching. In *Proc. WSP*, pages 141–155, 1996.
11. R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 1999.
12. G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.
13. D. R. Morrison. Patricia – practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968.
14. I. Munro, R. Raman, V. Raman, and S.S. Rao. Succinct representations of permutations. In *Proc. ICALP*, LNCS 2719, pages 345–356, 2003.
15. G. Navarro. Indexing text using the Ziv-Lempel trie. *J. Discrete Algorithms*, 2(1):87–114, 2004.
16. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.
17. R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding $k$-ary trees and multisets. In *Proc. SODA*, pages 233–242, 2002.
18. L. Russo and A. Oliveira. A compressed self-index using a Ziv-Lempel dictionary. In *Proc. SPIRE*, LNCS 4209, pages 163–180, 2006.
19. K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48(2):294–313, 2003.
20. K. Sadakane and R. Grossi. Squeezing Succinct Data Structures into Entropy Bounds. In *Proc. SODA*, pages 1230–1239, 2006.
21. J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory*, 24(5):530–536, 1978.

# Superconnectivity of regular graphs with small diameter

Camino Balbuena[1⋆], Jianmin Tang[2], Kim Marshall[2], Yuqing Lin[3]

[1] *Departament de Matemàtica Aplicada III*
Universitat Politècnica de Catalunya, Barcelona, Spain

[2] *School of Information Technology and Mathematical Sciences*
University of Ballarat, Ballarat, Victoria 3353, Australia

[3] *School of Electrical Engineering and Computer Science*
The University of Newcastle, NSW 2308, Australia

**Abstract.** The superconnectivity $\kappa_1$ of a connected graph $G$ is defined as the minimum cardinality of a vertex-cut over all vertex-cuts $X$ such that no vertex $x \notin X$ has all its neighbors in $X$. In this paper we prove for any $\delta$-regular graph of diameter $D$ and odd girth $g$ that if $D \leq g-2$ then $\kappa_1 > \delta$ when $g \geq 5$ and a complete graph otherwise.
**Key words.** connectivity, superconnectivity, cutset, diameter, girth.

## 1 Introduction

Let $G = (V, E)$ be a graph with vertex set $V = V(G)$ and edge set $E = E(G)$. Throughout this paper, only undirected simple graphs without loops or multiple edges are considered. Unless otherwise stated, we follow [7] for terminology and definitions.

The set of vertices adjacent to a vertex $v$ is called the *neighborhood of* $v$ and denoted by $N(v)$. A vertex in the neighborhood of $v$ is a *neighbor* of $v$. The *degree* of a vertex $v$ is $\deg(v) = |N(v)|$, and the minimum degree $\delta = \delta(G)$ of $G$ is the minimum degree over all vertices of $G$. A graph is called *$\delta$-regular* if all its vertices have the same degree $\delta$. If $S \subset V$ then $G[S]$ stand for the *subgraph induced* by $S$. The degree of a vertex $v$ in an induced subgraph $H$ of $G$ is $deg_H(v) = |N(v) \cap V(H)|$. The *minimum edge-degree* of $G$, denoted by $\xi = \xi(G)$, is defined as $\xi(G) = \min\{(d(u) + d(v) - 2 : uv \in E(G)\}$. The *distance* $d(u, v)$ of two vertices $u$ and $v$ in $G$ is the length of a shortest path between $u$ and $v$. The *diameter* of a graph $G$, written $D = D(G)$, is the maximum distance of any two vertices among all the vertices of $G$. The *girth* $g = g(G)$ is the length of a shortest cycle in $G$. For $S \subset V$, $d(w, S) = d_G(w, S) = \min\{d(w, s) : \ s \in S\}$ denotes the *distance* between a vertex $w$ and a set $S$. For every $v \in V$ and every positive integer $r \geq 0$, $N_r(v) = \{w \in V \ : \ d(w, v) = r\}$ denotes the *neighborhood*

⋆ m.camino.balbuena@upc.edu j.tang@ballarat.edu.au klmarshall@students.ballarat.edu.au yuqing.lin@newcastle.edu.au

*of $v$ at distance $r$*. Similarly, for $S \subset V$, the neighborhood of $S$ at distance $r$ is denoted $N_r(S) = \{w \in V : d(w, S) = r\}$. Observe that $N_0(S) = S$. When $r = 1$ we write $N(v)$ and $N(S)$, instead of $N_1(v)$ and $N_1(S)$.

A graph $G$ is *connected* if there is a path between any two vertices of $G$. If $X \subset V$ and $G - X$ is not connected, then $X$ is said to be a *cutset* or a *disconnecting set*. Analogously, If $F \subset E$ and $G - F$ is not connected, then $F$ is said to be an *edge-cut* or an *edge disconnecting set*. We say that $G$ is *$r$-connected* if the deletion of at least $r$ vertices of $G$ is required to disconnect the graph and in this case we say that the *vertex connectivity* $\kappa = \kappa(G) \geq r$. A complete graph with $r + 1$ vertices is $r$-connected. A graph with minimum degree $\delta$ is *maximally connected* if it is $\delta$-connected, or equivalently $\kappa = \delta$. The notion of superconnectedness was proposed in [4–6]. A graph is *superconnected*, for short *super-$\kappa$*, if all minimum cutsets consist of the vertices adjacent with one vertex, see Boesch [5], Boesch and Tindell [6] and Fiol, Fàbrega and Escudero [9]. Observe that a superconnected graph is necessarily maximally connected, $\kappa = \delta$, but the converse is not true. For example, a cycle $\mathcal{C}_g$ of length $g$ with $g \geq 6$ is a maximally connected graph that is not superconnected. A cutset $X$ of $G$ is called a *non-trivial cutset* if $X$ does not contain the neighborhood $N(u)$ of any vertex $u \notin X$. Provided that some non-trivial cutset exists, the *superconnectivity* of $G$ denoted by $\kappa_1$ was defined in [1, 9] as:

$$\kappa_1 = \kappa_1(G) = \min\{|X| : X \text{ is a non-trivial cutset}\}.$$

A non-trivial cutset $X$ is called a *$\kappa_1$-cut* if $|X| = \kappa_1$. Notice that if $\kappa_1 \leq \delta$, then $\kappa_1 = \kappa$ and that $\kappa_1 > \delta$ is a sufficient and necessary condition for $G$ to be super-$\kappa$, since all the minimum disconnecting sets with cardinality equal to $\delta$ must be trivial. A *non-trivial edge-cut*, the *edge-superconnectivity* $\lambda_1 = \lambda_1(G)$ and a *$\lambda_1$-cut* are defined analogously.

Some known sufficient conditions on the diameter of a graph in terms of its girth to guarantee lower bounds on $\kappa$, $\lambda$, $\kappa_1$ and $\lambda_1$ are listed in the following theorem.

**Theorem 1.** *Let $G$ be a graph with minimum degree $\delta \geq 2$, diameter $D$, girth $g$, edge minimum degree $\xi$, connectivities $\lambda$ and $\kappa$ and superconnectivities $\kappa_1$ and $\lambda_1$. Then,*

*(i) [10] $\lambda = \delta$ if $D \leq 2\lfloor (g-1)/2 \rfloor$.*
*(ii) [10] $\kappa = \delta$ if $D \leq 2\lfloor (g-1)/2 \rfloor - 1$.*
*(iii) [3] $\lambda_1 = \xi$ if $D \leq g - 2$.*
*(iv) [2] $\kappa_1 \geq \xi$ if $D \leq g - 3$.*

In this paper we improve Theorem 1 (ii) by proving that a $\delta$-regular graph $G$ with $\delta \geq 3$ and diameter at most $g - 2$ is super-$\kappa$ when $g$ odd. To do this we require the following known result.

**Proposition 1.** *[2] Let $G = (V, E)$ be a connected graph with girth $g$ and minimum degree $\delta \geq 2$. Let $X \subset V$ be a $\kappa_1$-cut of $|X| < \xi(G)$. Then for each connected component $C$ of $G - X$ there exists some vertex $u_0 \in V(C)$ such that $d(u_0, X) \geq \lceil (g-3)/2 \rceil$ and $|N_{\lceil (g-3)/2 \rceil}(u_0) \cap X| \leq 1$.*

In Section 2 we present our results.

## 2 Main results

We use Proposition 1 to prove the existence of some structural properties in a component $C$ when $g$ is odd and $\max\{d(u,X) : u \in V(C)\} = (g-3)/2$.

**Lemma 1.** *Let $G$ be a $\kappa_1$-connected graph with odd girth and minimum degree $\delta \geq 3$. Let $X$ be a $\kappa_1$-cut with $|X| = \delta$ and assume that there exists a connected component $C$ of $G - X$ such that $\max\{d(u,X) : u \in V(C)\} = (g-3)/2$. Then the following assertions hold:*

*(i) If $u \in V(C)$ is such that $d(u,X) = (g-3)/2$ and $|N_{(g-3)/2}(u) \cap X| = 1$, then $u$ has degree $d(u) = \delta$ and $\delta - 1$ neighbors $z$ such that $d(z,X) = (g-3)/2$ and $|N_{(g-3)/2}(z) \cap X| = 1$.*
*(ii) If $u \in V(C)$ is such that $d(u,X) = (g-3)/2$ and $|N_{(g-3)/2}(u) \cap X| = 1$, then $|N_{(g-1)/2}(u) \cap X| = \delta - 1$.*
*(iii) There exists a $(\delta-1)$-regular subgraph $\Gamma$ such that for every vertex $w \in V(\Gamma)$, $d_G(w) = \delta$ and $d(w,X) = (g-3)/2$.*
*(iv) If $g = 5$ then $|N(X) \cap V(C)| \geq \delta(\delta-1)$. And if $g \geq 7$ then $|N(X) \cap V(C)| \geq (\delta-1)^2 + 2$.*

As a consequence of Lemma 1 we obtain Theorem 2 which is an improvement of Theorem 1 $(i)$ for regular graphs of odd girth.

**Theorem 2.** *Let $G$ be a $\delta$-regular graph with $\delta \geq 3$ and odd girth $g$. If the diameter $D \leq g - 2$, then $G$ is super-$\kappa$ when $g \geq 5$ and a complete graph otherwise.*

The graph depicted in Figure 1 shows a non $\delta$-regular graph with $g = 5$, $D = 3$ which is non super-$\kappa$. Consequently, the hypothesis of regularity is essential to establish Theorem 2.

### Acknowledgments

Research supported by the Ministry of Science and Technology, Spain, the European Regional Development Fund (ERDF) under project MTM2005-08990-C02-02.

## References

1. C. Balbuena and A. Carmona, On the connectivity and superconnectivity of bipartite digraphs and graphs, *Ars. Combin.* 61 (2001), 3 – 21.
2. C. Balbuena, M. Cera, A. Diánez, P. García-Vázquez, and X. Marcote, On the restrited connectivity and superconnectivity in graphs with given girth, *Discrete Math.* (2006), doi: 10.1016/j.disc.2006.07.016

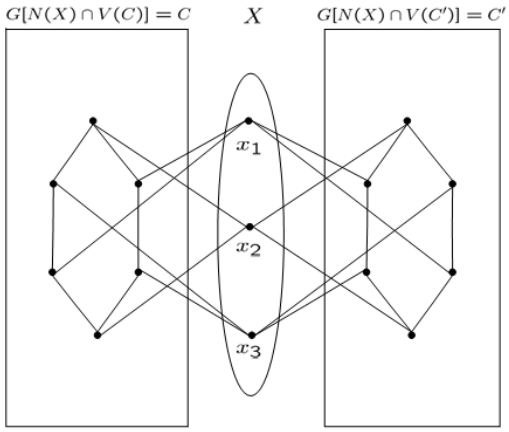



**Fig. 1.** A graph with $g = 5$ and $\kappa_1 = \delta = 3$.

3. C. Balbuena, P. García-Vázquez, and X. Marcote, Sufficient conditions for $\lambda'$-optimality in graphs with girth $g$, *J. of Graph Theory* 52(1) (2006), 73-86.
4. D. Bauer, F. Boesch, C. Suffel, and R. Tindell, Connectivity extremal problems and the design of reliable probabilistic networks, The theory and application of graphs, Y. Alavi and G. Chartrand (Editors), Wiley, New York (1981), 89–98.
5. F.T. Boesch, Synthesis of reliable networks-A survey, *IEEE Trans. Reliability* 35 (1986), 240–246.
6. F.T. Boesch and R. Tindell, Circulants and their connectivities, *J. Graph Theory* 8(4) (1984), 487–499.
7. G. Chartrand and L. Lesniak, *Graphs and digraphs*, Third edition, Chapman and Hall, London, 1996.
8. J. Fàbrega and M.A. Fiol, Maximally connected digraphs. *J. Graph Theory* 13 (1989), 657–668.
9. M.A. Fiol, J. Fàbrega, and M. Escudero, Short paths and connectivity in graphs and digraphs. *Ars Combin.* 29B (1990), 17–31.
10. T. Soneoka, H. Nakada, M. Imase, and C. Peyrat, Sufficient conditions for maximally connected dense graphs, *Discrete Math.* 63 (1987), 53–66.

# On diregularity of digraphs of defect two

Dafik [1,2], Mirka Miller [1,3], Costas Iliopoulos [4] and Zdenek Ryjacek [3]

[1]School of Information Technology and Mathematical Sciences
University of Ballarat, Australia

[2]Department of Mathematics Education
Universitas Jember, Indonesia

[3]Department of Mathematics
University of West Bohemia, Plzeň, Czech Republic

[4]Department of Computer Science
Kings College, London, UK

d.dafik@students.ballarat.edu.au
m.miller@ballarat.edu.au
csi@dcs.kcl.ac.uk
ryjacek@kma.zcu.cz

**Abstract**: Since Moore digraphs do not exist for $k \neq 1$ and $d \neq 1$, the problem of finding the existence of digraph of out-degree $d \geq 2$ and diameter $k \geq 2$ and order close to the Moore bound becomes an interesting problem. To prove the non-existence of such digraphs, we first may wish to establish their diregularity. It is easy to show that any digraph with out-degree at most $d \geq 2$, diameter $k \geq 2$ and order $n = d + d^2 + \ldots + d^k - 1$, that is, two less than Moore bound must have all vertices of out-degree $d$. However, establishing the regularity or otherwise of the in-degree of such a digraph is not easy. In this paper we prove that all digraphs of defect two are out-regular and almost in-regular.

**Key Words***: Diregularity, digraph of defect two, degree-diameter problem.*

## 1 Introduction

By a *directed graph* or a *digraph* we mean a structure $G = (V(G), A(G))$, where $V(G)$ is a finite nonempty set of distinct elements called *vertices*, and $A(G)$ is a set of ordered pair $(u, v)$ of distinct vertices $u, v \in V(G)$ called *arcs*.

The *order* of the digraph $G$ is the number of vertices in $G$. An *in-neighbour* (respectively, *out-neighbour*) of a vertex $v$ in $G$ is a vertex $u$ (respectively, $w$) such

[1] This research was supported by the Australian Research Council (ARC) Discovery Project grant DP04502994.

that $(u, v) \in A(G)$ (respectively, $(v, w) \in A(G)$). The set of all in-neighbours (respectively, out-neighbours) of a vertex $v$ is called the *in-neighbourhood* (respectively, the *out-neighbourhood*) of $v$ and denoted by $N^-(v)$ (respectively, $N^+(v)$). The *in-degree* (respectively, *out-degree*) of a vertex $v$ is the number of all its in-neighbours (respectively, out-neighbours). If every vertex of a digraph $G$ has the same in-degree (respectively, out-degree) then $G$ is said to be *in-regular* (respectively, *out-regular*). A digraph $G$ is called a *diregular* digraph of degree $d$ if $G$ is in-regular of in-degree $d$ and out-regular of out-degree $d$.

An alternating sequence $v_0a_1v_1a_2...a_lv_l$ of vertices and arcs in $G$ such that $a_i = (v_{i-1}, v_i)$ for each $i$ is called a *walk* of length $l$ in $G$. A walk is *closed* if $v_0 = v_l$. If all the vertices of a $v_0 - v_l$ walk are distinct, then such a walk is called a *path*. A *cycle* is a closed path. A *digon* is a cycle of length 2.

The *distance* from vertex $u$ to vertex $v$, denoted by $\delta(u, v)$, is the length of a shortest path from $u$ to $v$, if any; otherwise, $\delta(u, v) = \infty$. Note that, in general, $\delta(u, v)$ is not necessarily equal to $\delta(v, u)$. The *in-eccentricity* of $v$, denoted by $e^-(v)$, is defined as $e^-(v) = max\{\delta(u, v) : u \in V\}$ and *out-eccentricity* of $v$, denoted by $e^+(v)$, is defined as $e^+(v) = max\{\delta(v, u) : u \in V\}$. The radius of $G$, denoted by rad$(G)$, is defined as rad$(G)$= $min\{e^-(v) : v \in V\}$. The *diameter* of $G$, denoted by diam$(G)$, is defined as diam$(G)$= $max\{e^-(v) : v \in V\}$. Note that if $G$ is a strongly connected digraph then, equivalently, we could have defined the radius and the diameter of $G$ in terms of out-eccentricity instead of in-eccentricity. The *girth* of a digraph $G$ is the length of a shortest cycle in $G$.

The well known *degree/diameter* problem for digraphs is to determine the largest possible order $n_{d,k}$ of a digraph, given out-degree at most $d \geq 1$ and diameter $k \geq 1$. There is a natural upper bound on the order of digraphs given out-degree at most $d$ and diameter $k$. For any given vertex $v$ of a digraph $G$, we can count the number of vertices at a particular distance from that vertex. Let $n_i$ , for $0 \leq i \leq k$, be the number of vertices at distance $i$ from $v$. Then $n_i \leq d^i$, for $0 \leq i \leq k$, and consequently,

$$n_{d,k} = \sum_{i=0}^{k} n_i \leq 1 + d + d^2 + \ldots + d^k. \quad (1)$$

The right-hand side of (1), denoted by $M_{d,k}$, is called the *Moore bound*. If the equality sign holds in (1) then the digraph is called a *Moore digraph*. It is well known that Moore digraphs exist only in the cases when $d = 1$ (directed cycles of length $k + 1$, $C_{k+1}$ , for any $k \geq 1$) or $k = 1$ (complete digraphs of order $d + 1, K_{d+1}$, for any $d \geq 1$) [2, 11].

Note that every Moore digraph is diregular (of degree one in the case of $C_{k+1}$ and of degree $d$ in the case of $K_{d+1}$). Since for $d > 1$ and $k > 1$ there are no Moore digraphs, we are next interested in digraphs of order $n$ 'close' to Moore bound.

It is easy to show that a digraph of order $n$, $M_{d,k} - M_{d,k-1} + 1 \leq n \leq M_{d,k} - 1$, with out-degree at most $d \geq 2$ and diameter $k \geq 2$ must have all vertices of out-degree $d$. In other words, the out-degree of such a digraph is constant ($= d$). This can be easily seen because if there were a vertex in the digraph with out-degree $d_1 < d$ (i.e., $d_1 \leq d - 1$), then the order of the digraph,

$$\begin{aligned} n &\leq 1 + d_1 + d_1 d + \ldots + d_1 d^{k-1} \\ &= 1 + d_1(1 + d + \ldots + d^{k-1}) \\ &\leq 1 + (d-1)(1 + d + \ldots + d^{k-1}) \\ &= (1 + d + \ldots + d^{k}) - (1 + d + \ldots + d^{k-1}) \\ &= M_{d,k} - M_{d,k-1} \\ &< M_{d,k} - M_{d,k-1} + 1, \end{aligned}$$

However, establishing the regularity or otherwise of in-degree for an *almost* Moore digraph is not easy. It is well known that there exist digraphs of out-degree $d$ and diameter $k$ whose order is just two or three less than the Moore bound and in which *not all* vertices have the same in-degree. In Fig. 1 we give two examples of digraphs of diameter 2, out-degree $d = 2, 3$, respectively, and order $M_{d,2} - d$, with vertices not all of the same in-degree.

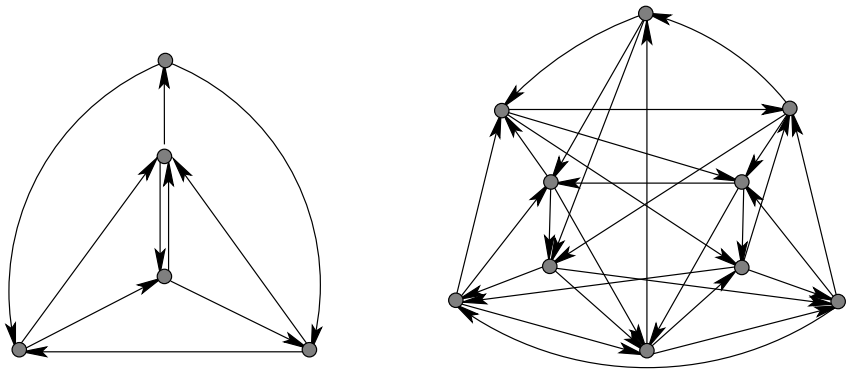


**Fig. 1.** Two examples of non-diregular digraphs.

Miller, Gimbert, Širáň and Slamin [7] considered the diregularity of digraphs of defect one, that is, $n = M_{d,k} - 1$, and proved that such digraphs are diregular. For defect two, diameter $k = 2$ and any out-degree $d \geq 2$, non-diregular digraphs always exist. One such family of digraphs can be generated from Kautz digraphs which contain vertices with identical out-neighbourhoods and so we can apply vertex deletion scheme, see [8], to obtain non-diregular digraphs of defect two, diameter $k = 2$, and any out-degree $d \geq 2$. Fig. 2(a) shows an example of Kautz digraph $G$ of order $n = M_{3,2} - 1$ which we will use to illustrate the vertex deletion scheme. Note the existence of identical out-neighbourhoods, for example, $N^+(v_{11}) = N^+(v_{12})$. Deleting vertex $v_{12}$, together with its outgoing arcs, and then reconnecting its incoming arcs to vertex 11, we obtain a new digraph $G_1$ of order $n = M_{3,2} - 2$, as shown in Fig. 2(b).

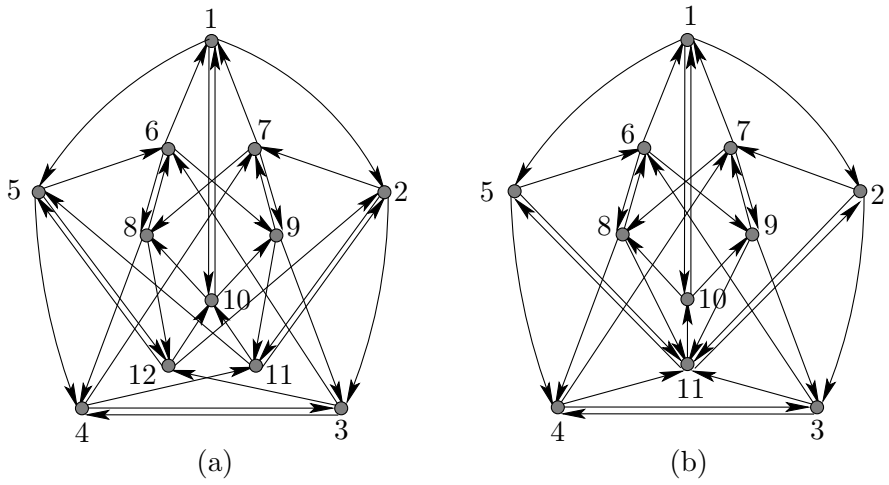



**Fig. 2.** Digraphs $G$ of order 12 and $G_1$ of order 11.

We now introduce the notion of 'almost diregularity'. Throughout this paper, let $S$ be the set of all vertices of $G$ whose in-degree is less than $d$. Let $S'$ be the set of all vertices of $G$ whose in-degree is greater than $d$; and let $\sigma^-$ be the *in-excess*, $\sigma^- = \sigma^-(G) = \sum_{w \in S'}(d^-(w) - d) = \sum_{v \in S}(d - d^-(v))$. Similarly, let $R$ be the set of all vertices of $G$ whose out-degree is less than $d$. Let $R'$ be the set of all vertices of $G$ whose out-degree is greater than $d$. We define the *out-excess*, $\sigma^+ = \sigma^+(G) = \sum_{w \in R'}(d^+(w) - d) = \sum_{v \in R}(d - d^+(v))$. A digraph

of average in-degree $d$ is called *almost in-regular* if the in-excess is at most equal to $d$. Similarly, a digraph of average out-degree $d$ is called *almost out-regular* if the out-excess is at most equal to $d$. A digraph is *almost diregular* if it is almost in-regular and almost out-regular. Note that if $\sigma^- = 0$ (respectively, $\sigma^+ = 0$) then $G$ is in-regular (respectively, out-regular). In this paper we prove that all digraphs of defect two, diameter $k \geq 3$ and out-degree $d \geq 2$ are out-regular and almost in-regular.

## 2 Results

Let $G$ be a digraph of out-degree $d \geq 3$, diameter $k \geq 3$ and order $M_{d,k} - 2$. Since the order of $G$ is $M_{d,k} - 2$, using a counting argument, it is easy to show that for each vertex $u$ of $G$ there exist exactly two vertices $r_1(u)$ and $r_2(u)$ (not necessarily distinct) in $G$ with the property that there are two $u \to r_i(u)$ walks, for $i = 1, 2$, in $G$ of length not exceeding $k$. The vertex $r_i(u)$, for each $i = 1, 2$, is called the *repeat* of $u$; this concept was introduced in [5].

We will use the following notation throughout. For each vertex $u$ of a digraph $G$ described above, and for $1 \leq s \leq k$, let $T_s^+(u)$ be the multiset of all endvertices of directed paths in $G$ of length at most $s$ which start at $u$. Similarly, by $T_s^-(u)$ we denote the multiset of all starting vertices of directed paths of length at most $s$ in $G$ which terminate at $u$. Observe that the vertex $u$ is in both $T_s^+(u)$ and $T_s^-(u)$, as it corresponds to a path of zero length. Let $N_s^+(u)$ be the set of all endvertices of directed paths in $G$ of length exactly $s$ which start at $u$. Similarly, by $N_s^-(u)$ we denote the set of all starting vertices of directed paths of length exactly $s$ in $G$ which terminate at $u$. If $s = 1$, the sets $T_1^+(u) \setminus \{u\}$ and $T_1^-(u) \setminus \{u\}$ represent the out- and in-neighbourhoods of the vertex $u$ in the digraph $G$; we denote these neighbourhoods simply by $N^+(u)$ and $N^-(u)$, respectively. We illustrate the notations $T_s^+(u)$ and $N_s^+(u)$ in Fig. 3.

We will use the following notation throughout.

**Notation 1** *We say that $G$ is a $(d, k, \delta)$-digraph, that is, $G \in \mathcal{G}(d, k, \delta)$, if $G$ is a digraph of defect $\delta$, maximum out-degree $d$ and diameter $k$.*

We will present our new results concerning the diregularity of digraphs of order close to Moore bound in the following sections.

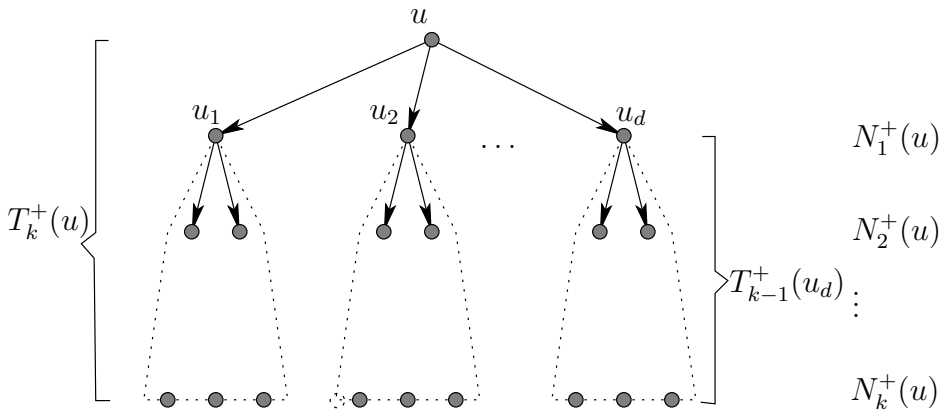



**Fig. 3.** Multiset $T_k^+(u)$

### 2.1 Diregularity of $(d, k, 2)$-digraphs

In this section we present a new result concerning the in-regularity of digraphs of defect two for any out-degree $d \geq 2$ and diameter $k \geq 3$. Let $S$ be the set of all vertices of $G$ whose in-degree is less than $d$. Let $S'$ be the set of all vertices of $G$ whose in-degree is greater than $d$; and let $\sigma$ be the *in-excess*, $\sigma^- = \sum_{w \in S'}(d^-(w) - d) = \sum_{v \in S}(d - d^-(v))$.

**Lemma 1** *Let $G \in \mathcal{G}(d, k, 2)$. Let $S$ be the set of all vertices of $G$ whose in-degree is less than $d$. Then $S \subseteq N^+(r_1(u)) \cup N^+(r_2(u))$, for any vertex $u$.*

**Proof.** Let $v \in S$. Consider an arbitrary vertex $u \in V(G)$, $u \neq v$, and let $N^+(u) = \{u_1, u_2, ..., u_d\}$. Since the diameter of $G$ is equal to $k$, the vertex $v$ must occur in each of the sets $T_k^+(u_i)$, $i = 1, 2, ..., d$. It follows that for each $i$ there exists a vertex $x_i \in \{u\} \cup T_{k-1}^+(u_i)$ such that $x_i v$ is an arc of $G$. Since the in-degree of $v$ is less than $d$ then the in-neighbours $x_i$ of $v$ are not all distinct. This implies that there exists some vertex which occurs at least twice in $T_k^+(u)$. Such a vertex must be a repeat of $u$. As $G$ has defect 2, there are at most two vertices of $G$ which are repeats of $u$, namely, $r_1(u)$ and $r_2(u)$. Therefore, $S \subseteq N^+(r_1(u)) \cup N^+(r_2(u))$. □

Combining Lemma 1 with the fact that every vertex in $G$ has out-degree $d$ gives

**Corollary 1** $|S| \leq 2d$.

In principle, we might expect that the in-degree of $v \in S$ could attain any value between 1 and $d-1$. However, the next lemma asserts that the in-degree cannot be less than $d-1$.

**Lemma 2** *Let $G \in \mathcal{G}(d,k,2)$. If $v_1 \in S$ then $d^-(v_1) = d-1$.*

**Proof.** Let $v_1 \in S$. Consider an arbitrary vertex $u \in V(G)$, $u \neq v_1$, and let $N^+(u) = \{u_1, u_2, ..., u_d\}$. Since the diameter of $G$ is equal to $k$, the vertex $v_1$ must occur in each of the sets $T_k^+(u_i)$, $i = 1, 2, ..., d$. It follows that for each $i$ there exists a vertex $x_i \in \{u\} \cup T_{k-1}^+(u_i)$ such that $x_i v_1$ is an arc of $G$. If $d^-(v_1) \leq d-3$ then there are at least three repeats of $u$, which is impossible. Suppose that $d^-(v_1) \leq d-2$. By Lemma 1, the in-excess must satisfy

$$\sigma^- = \sum_{x \in S'} (d^-(x) - d) = \sum_{v_1 \in S} (d - d^-(v_1)) = |S| \leq 2d.$$

We now consider the number of vertices in the multiset $T_k^-(v_1)$. To reach $v_1$ from all the other vertices in $G$, the number of distinct vertices in $T_k^-(v_1)$ must be

$$|T_k^-(v_1)| \leq \sum_{u \in N_{t-1}^-(v_1)} d^-(u) = d|N_{t-1}^-(v_1)| + \varepsilon_t, \tag{2}$$

where $2 \leq t \leq k$ and $\varepsilon_2 + \varepsilon_3 + \ldots + \varepsilon_k \leq \sigma$. If $d^-(v_1) = d-2$ then $|N^-(v_1)| = |N_1^-(v_1)| = d-2$. It is not difficult to see that a safe upper bound on the sum of $|T_k^-(v_1)|$ is obtained from the inequality (2) by setting $\varepsilon_2 = 2d$, and $\varepsilon_t = 0$ for $3 \leq t \leq k$. This gives

$$\begin{aligned}
|T_k^-(v_1)| &\leq 1 + |N_1^-(v_1)| + |N_2^-(v_1)| + |N_3^-(v_1)| + \ldots + |N_k^-(v_1)| \\
&= 1 + (d-2) + (d(d-2) + \varepsilon_2) + (d(d(d-2) + \varepsilon_2) + \varepsilon_3) \\
&\quad (1 + d + \cdots + d^{k-3}) \\
&= 1 + (d-2) + (d(d-2) + 2d) + (d(d(d-2) + 2d) + 0) \\
&\quad (1 + d + \cdots + d^{k-3}) \\
&= 1 + d - 2 + d^2 + d^3(1 + d + \cdots + d^{k-3}) \\
&= M_{d,k} - 2.
\end{aligned}$$

Since $\varepsilon_2 = 2d$, $\varepsilon_t = 0$ for $3 \leq t \leq k$, and $G$ contains a vertex of in-degree $d-2$ then $|S| = d$. Let $S = \{v_1, v_2, \ldots, v_d\}$. Every $v_i$, for $i = 2, 3, \ldots, d$, has to reach $v_1$ at distance at most $k$. Since $v_1$ and every $v_i$ have exactly the same

in-neighbourhood then $v_1$ is forced to be selfrepeat. This implies that $v_1$ occurs twice in the multiset $T_k^-(v_1)$. Hence $|T^-(v_1)| < M_{d,k}-2$, which is a contradiction. Therefore $d^-(v_1) = d-1$, for any $v_1 \in S$. □

**Lemma 3** *If $S$ is the set of all vertices of $G$ whose in-degree is $d-1$ then $|S| \leq d$.*

**Proof.** Suppose $|S| \geq d+1$. Then there exist $v_i \in S$ such that $d^-(v_i) = 1$, for $i = 1, 2, \ldots, d+1$. The in-excess $\sigma^- = \sum_{v \in S}(d - d^-(v)) \geq d+1$. This implies that $|S'| \geq 1$. However, we cannot have $|S'| = 1$. Suppose, for a contradiction, $S' = \{x\}$. To reach $v_1$ (and $v_i$, $i = 2, 3, \ldots, d+1$) from all the other vertices in $G$, we must have $x \in \bigcap_{i=1}^{d+1} N^-(v_i)$, which is impossible as the out-degree of $x$ is $d$. Hence $|S'| \geq 2$.

Let $u \in V(G)$ and $u \neq v_i$. To reach $v_i$ from $u$, we must have $\bigcup_{i=1}^{d+1} N^-(v_i) \subseteq \{r_1(u), r_2(u)\}$. Since the out-degree is $d$ then $|\bigcup_{i=1}^{d+1} N^-(v_i)| = d$. Without loss of generality, we suppose $x_1 \in \bigcup_{i=1}^{d} N^-(v_i)$ and $x_2 \in N^-(v_{d+1})$, where $x_1, x_2 \in S'$. Now consider the multiset $T_k^+(x_1)$. Since every $v_i$, for $i = 1, 2, \ldots, d$, respectively, must reach $\{v_{j \neq i}\}$, for $j = 1, 2, \ldots, d+1$, within distance at most $k$, then $x_1$ occurs three times in $T_k^+(x_1)$, otherwise $x_1$ will have at least three repeats, which is impossible. This implies that $x_1$ is a double selfrepeat. Since two of $v_i$, say $v_k$ and $v_l$, for $k, l \in \{1, 2, \ldots, d+1\}$, occur in the walk joining two selfrepeats then $v_k$ and $v_l$ are selfrepeats. Then it is not possible for the $d$ out-neighbours of $x_1$ to reach $v_{d+1}$. □

**Theorem 1** *For $k \geq 3$ and $d \geq 2$, every $(d, k, 2)$-digraph is out-regular and almost in-regular.*

**Proof.** Out-regularity of $(d, k, 2)$-digraphs was explained in the Introduction. Hence we only need to proof that every $(d, k, 2)$-digraph is almost in-regular. If $S = \emptyset$ then $(d, k, 2)$-digraph is diregular. By Lemma 2, if $S \neq \emptyset$ then all vertices in $S$ have in-degree $d-1$. This gives

$$\sigma = \sum_{x \in S'} (d^-(x) - d) = \sum_{v \in S} (d - d^-(v)) = |S| \leq 2d.$$

Take an arbitrary vertex $v \in S$; then $|N^-(v)| = |N_1^-(v)| = d-1$. By the diameter assumption, the union of all the sets $N_t^-(v)$ for $0 \leq t \leq k$ is the entire vertex set $V(G)$ of $G$, which implies that

$$|V(G)| \leq \sum_{t=0}^{k} |N_t^-(v)|. \tag{3}$$

To estimate the above sum we can observe the following inequality

$$|N_t^-(v)| \leq \sum_{u \in N_{t-1}^-(v)} d^-(u) = d|N_{t-1}^-(v)| + \varepsilon_t, \tag{4}$$

where $2 \leq t \leq k$ and $\varepsilon_2 + \varepsilon_3 + \ldots + \varepsilon_k \leq \sigma$.

It is not difficult to see that a safe upper bound on the sum of $|V(G)|$ is obtained from the inequality (4) by setting $\varepsilon_2 = \sigma = |S|$, and $\varepsilon_t = 0$, for $3 \leq t \leq k$; note that the latter is equivalent to assuming that *all* vertices from $S \setminus \{v\}$ are contained in $N_k^-(v)$ and that all vertices of $S'$ belong to $N_1^-(v)$. This way we successively obtain:

$$\begin{aligned}
|V(G)| &\leq 1 + |N_1^-(v)| + |N_2^-(v)| + |N_3^-(v)| + \ldots + |N_k^-(v)| \\
&\leq 1 + (d-1) + (d(d-1) + |S|)(1 + d + \cdots + d^{k-2}) \\
&= d + d^2 + \cdots + d^k + (|S| - d)(1 + d + \cdots + d^{k-2}) \\
&= M_{d,k} - 2 + (|S| - d)(1 + d + \cdots + d^{k-2}) + 1.
\end{aligned}$$

But $G$ is a digraph of order $M_{d,k} - 2$; this implies that

$$\begin{aligned}
(|S| - d)(1 + d + \cdots + d^{k-2}) + 1 &\geq 0 \\
(|S| - d)\frac{d^{k-1} - 1}{d - 1} + 1 &\geq 0 \\
|S| &\geq d - \frac{d-1}{d^{k-1} - 1}
\end{aligned}$$

As $0 < \frac{d-1}{d^{k-1}-1} < 1$, whenever $k \geq 3$ and $d \geq 4$, it follows that $|S| \geq d$. Since $1 \leq |S| \leq d$. This implies $|S| = d$. □

We conclude with a conjecture.

**Conjecture 1** *All digraphs of defect 2 are diregular for diameter $k \geq 3$ and maximum out-degree $d \geq 2$.*

## References


1. E.T. Baskoro, M. Miller, J. Plesník, On the structure of digraphs with order close to the Moore bound, *Graphs and Combinatorics*, **14(2)** (1998) 109–119.
2. W.G. Bridges, S. Toueg, On the impossibility of directed Moore graphs, *J. Combin. Theory* Series B, **29** (1980) 339–341.
3. Dafik, M. Miller and Slamin, Diregularity of digraphs of defect two of out-degree three and diameter $k \geq 3$, preprint.

4. B.D. McKay, M. Miller, J. Širáň, A note on large graphs of diameter two and given maximum degree, *J. Combin. Theory* (B), **74** (1998) 110–118.
5. M. Miller, I. Fris, Maximum order digraphs for diameter 2 or degree 2, *Pullman volume of Graphs and Matrices*, *Lecture Notes in Pure and Applied Mathematics*, **139** (1992) 269–278.
6. M. Miller, J. Širáň, Digraphs of degree two which miss the Moore bound by two, *Discrete Math.*, **226** (2001) 269–280.
7. M. Miller, J. Gimbert, J. Širáň, Slamin, Almost Moore digraphs are diregular, *Discrete Math.*, **216** (2000) 265-270.
8. M. Miller, Slamin, On the monotonocity of minimum diameter with respect to order and maximum out-degree, *Proceeding of COCOON 2000, Lecture Notes in Computer Science* 1558 (D.-Z Du, P. Eades, V.Estivill-Castro, X.Lin (eds.)) (2000) 193-201.
9. M. Miller, I. Fris, Minimum diameter of diregular digraphs of degree 2, *Computer Journal*, **31** (1988) 71–75.
10. M. Miller, J. Širáň, Moore graphs and beyond: A survey of the degree/diameter problem, *Electronic J. Combin.*, **11** (2004).
11. J. Plesník, Š. Znám, Strongly geodetic directed graphs, *Acta F. R. N. Univ. Comen. - Mathematica XXIX*, (1974)
12. Slamin, M. Miller, Diregularity of digraphs close to Moore bound, Prosiding Konferensi Nasional X Matematika, ITB Bandung, MIHMI, **6, No.5** (2000) 185-192.

# Change Detection through Clustering and Spectral Analysis

Diane Donovan[1], Birgit Loch[2],
H.B. Thompson[1] and Jayne Thompson[3]

[1] Department of Mathematics, University of Queensland, St Lucia 4072, Australia
[2] Department of Mathematics and Computing, University of Southern Queensland Toowoomba 4350, Australia
[3] School of Physics, Melbourne University, Parkville 3052, Australia

**Abstract.** This paper defines a metric on a sequence of graphs which can be used to measure the distance between consecutive graphs. The method is based on vertex clustering through spectral analysis of the Laplace matrix.

Keywords: Networks, Clustering, Spectral Graph Theory

## 1 Introduction

Our aim is to investigate intragraph clustering in large enterprise networks and to define a metric, for quantifying network change, based on the evolution of vertex clusters. The use of clustering as a technique for change detection is a relatively recent idea and one which requires further study. In this paper, a metric will be defined on a sequence of graphs $G^1, \ldots, G^s$. The basis for the metric will be a clustering procedure which uses edge weights to determine a partition $\mathcal{C}^h$ of the vertex set of graph $G^h$. Then a distance measure $d(\mathcal{C}^h, \mathcal{C}^{h+1})$, $1 \leq h \leq s-1$, will be computed and used to quantify the distance $d(G^h, G^{h+1})$. The clustering procedure will be based on spectral analysis. This combination of spectral analysis, intragraph clusters and change detection is a new field of study and one which will be explored in the current paper.

In spectral graph theory (see [11] Section 8.6, page 452, for a general discussion) the eigenvalues and eigenvectors of associated matrices are calculated and used to characterize a graph's global structure. The literature includes some papers studying spectral theory and quantifying change in networks. For instance, for each graph $G^h$ selected from a sequence of graphs, $G^1, \ldots, G^s$, Bunke, Dickinson, Kraetzl and Wallis (see [1], page 72) determine the $k$ largest positive eigenvalues $\lambda_1^h, \ldots, \lambda_k^h$ and quantify a graph distance measure by setting

$$d(G^h, G^{h+1}) = \frac{\sum_{j=1}^{k} (\lambda_j^h - \lambda_j^{h+1})^2}{\min\left(\sum_{j=1}^{k} (\lambda_j^h)^2, \sum_{j=1}^{k} (\lambda_j^{h+1})^2\right)}. \tag{1}$$

This measure is termed the *spectral distance* and it will be one of the techniques used to calibrate the results obtained in this paper (see Section 6). Using similar ideas, Robles-Kelly and Hancock [8] calculate the largest eigenvalue for the adjacency matrix of a graph. The associated eigenvector is then used to identify an ordered path traversing every vertex. This ordered path forms the basis for computing the distance between graphs within the sequence. Robles-Kelly and Hancock also go on to discuss clustering, however individual clusters are determined using the largest eigenvalue. In this paper, we take a different approach and adapt techniques which have been used extensively for image analysis and graphs embedded in Euclidean space, see [6]. Initially, we will focus on a rigorous theoretical discussion of the eigenvectors of the Laplace matrix and associated techniques for partitioning the vertex set. By carefully analysing the theory we are able to demonstrate that the second smallest eigenvalue provides a good measure for determining vertex cluster sets. As noted by Hagan and Kahng [4], the advantage of this technique is that the partitioning is based on global information extracted from the overall network. Once we have determined the vertex clustering for each graph in a sequence, the Rand Index (see Dickinson, Bunke, Dadej and Kraetzl [3] and also [1] page 118) is used to define a graph distance measure.

Section 3 provides the necessary background on the Rand index. The underlying theory is reviewed in Section 4, providing a rigorous justification for the techniques under investigation. Section 5 discusses the necessary heuristics and provides some justification for the methods used. The theory is then tested in Section 6 and a comparison is made with distance measures studied by Bunke, Dickinson, Kraetzl and Wallis [1].

## 2 Definitions

A *graph* $G = (V, E(V))$ is a *vertex set* $V$ and a collection $E(V)$ of 2-element subsets, chosen from $V$. If $\{u, v\} \in E(V)$, then $\{u, v\}$ is called an *edge* and $u$ and $v$ the *endpoints* of the edge. In this paper all graphs will be simple, in that there are no repeated edges and all edges have two distinct endpoints. Each edge in the graph will be assigned a label or *weight*. Thus it will be assumed that there exists a function $\beta$, where

$$\beta : E(V) \longrightarrow \mathbb{Z}^+ \cup \{0\}.$$

When we wish to emphasize the fact that the edges of the graph are labeled we will use the notation $G = (V, E(V), \beta)$.

It will be productive to define a number of matrices to summarize specific information about a graph. We begin by setting $m = |V|$ and define an ordering on the vertex set; that is, the vertices are given an arbitrary order $v_1, \ldots, v_m$ and this ordering is used to define a vector $\mathcal{V} = (v_1, \ldots, v_m)^T$. An *adjacency matrix* $A = (a_{ij})$, for a graph $G = (V, E(V), \beta)$, is defined to be a $|V| \times |V|$ matrix with

entries

$$a_{ij} = \begin{cases} \beta(\{v_i, v_j\}), \text{ where } \{v_i, v_j\} \in E(V), \\ 0, \text{ otherwise.} \end{cases}$$

Given the adjacency matrix $A$ we let $a_i = \sum_j a_{ij}$ be the sum of the $i$th row (or column since $A$ is symmetric) and define the *diagonal matrix* to be $D = (d_{ij})$, where $d_{ij} = a_i \delta_{ij}$. Finally $B = D - A$ is the *disconnection* or *Laplace matrix*. Note that since $A$ is symmetric, $B$ is self adjoint. Hence, by the Spectral Theorem, $B$ has a full complement of orthogonal eigenvectors.

*Example 1.* These concepts are illustrated for the graph given in Figure 1.

- vertex set is $V = \{A, B, C, D, E, F\}$;
- vertex ordering is given by $\mathcal{V} = (A, B, C, D, E, F)^T$;
- edge set is $E(V) = \{\{A, D\}, \{B, D\}, \{B, C\}, \{C, E\}, \{D, E\}, \{D, F\}\}$;
- weights are $\beta(\{A, D\}) = 5,\ \beta(\{B, D\}) = 1,\ \beta(\{B, C\}) = 2,$ $\beta(\{C, E\}) = 5,\ \beta(\{D, E\}) = 2,\ \beta(\{D, F\}) = 3.$
- matrices are

$$D - A = \begin{bmatrix} 5&0&0&0&0&0 \\ 0&3&0&0&0&0 \\ 0&0&7&0&0&0 \\ 0&0&0&11&0&0 \\ 0&0&0&0&7&0 \\ 0&0&0&0&0&3 \end{bmatrix} - \begin{bmatrix} 0&0&0&5&0&0 \\ 0&0&2&1&0&0 \\ 0&2&0&0&5&0 \\ 5&1&0&0&2&3 \\ 0&0&5&2&0&0 \\ 0&0&0&3&0&0 \end{bmatrix} = \begin{bmatrix} 5&0&0&-5&0&0 \\ 0&3&-2&-1&0&0 \\ 0&-2&7&0&-5&0 \\ -5&-1&0&11&-2&-3 \\ 0&0&-5&-2&7&0 \\ 0&0&0&-3&0&3 \end{bmatrix} = B.$$

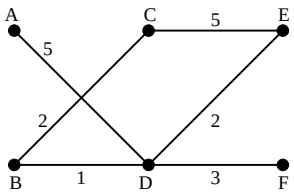



Figure 1: A weighted graph $G = (V, E(V), \beta)$

## 3 Distance Measure

The focus of this paper will be the partitioning of the vertex set $V$ of a graph $G$ into *cluster sets* $C_1, \ldots, C_t$, such that $V = C_1 \cup \ldots \cup C_t$ and $C_i \cap C_j = \emptyset$, for $1 \leq i < j \leq t$. The collection $\mathcal{C} = \{C_1, \ldots, C_t\}$ is termed a *clustering* of $V$. This partition defines an equivalence relation $\rho(\mathcal{C})$ on the vertex set $V$; that is, for any two vertices $x, y \in V$, $(x, y) \in \rho(\mathcal{C})$ if and only if there exists $C_i \in \mathcal{C}$ such that $x, y \in C_i$.

Given two graphs $G^1$ and $G^2$, with associated clusterings $\mathcal{C}^1$ and $\mathcal{C}^2$, we seek to quantify the distance $d(G^1, G^2)$ by specifying the distance between $\mathcal{C}^1$ and $\mathcal{C}^2$. We say a pair of vertices $x, y \in V$ are *consistent* if either $(x, y) \in \rho(\mathcal{C}^1)$

and $(x,y) \in \rho(\mathcal{C}^2)$, or $(x,y) \notin \rho(\mathcal{C}^1)$ and $(x,y) \notin \rho(\mathcal{C}^2)$. Otherwise the vertices $x$ and $y$ are said to be *inconsistent*. That is, two vertices $x$ and $y$ are consistent if they belong to the same cluster set in $\mathcal{C}^1$ and the same cluster set in $\mathcal{C}^2$, or they are in different cluster sets in both $\mathcal{C}^1$ and $\mathcal{C}^2$. Then $R^+ = |\{\{x,y\} \mid x,y \text{ are consistent vertices in } V\}|$ and $R^- = |\{\{x,y\} \mid x,y \text{ are inconsistent vertices in } V\}|$. Note that if $|V| = m$, then $R^+ + R^- = m(m-1)/2$. Finally the *Rand Index* for a pair of clusterings $\mathcal{C}^1$ and $\mathcal{C}^2$ is defined to be

$$R(\mathcal{C}^1, \mathcal{C}^2) = 1 - \frac{R^+}{R^+ + R^-}.$$

We note that $R(\mathcal{C}^1, \mathcal{C}^2) \in [0,1]$, with $R(\mathcal{C}^1, \mathcal{C}^2) = 0$ if and only if $|\mathcal{C}^1| = |\mathcal{C}^2|$ and all pairs of vertices are consistent, and $R(\mathcal{C}^1, \mathcal{C}^2) = 1$ if all pairs of vertices are inconsistent.

For a sequence of graphs the Rand Index will be used to measure the distance between consecutive graphs in the sequence. That is, for a given sequence of graphs $G^1, \ldots, G^s$, we will determine a sequence of clusterings $\mathcal{C}^1, \ldots, \mathcal{C}^s$, where $\mathcal{C}^i$ is a clustering on the vertex set of graph $G^i$. Then

$$d(G^i, G^{i+1}) = R(\mathcal{C}^i, \mathcal{C}^{i+1}).$$

## 4 The clustering procedure

In this section we will follow the work of Hall [6], including ideas of Hagen and Kahng [4], and develop techniques needed to partition the vertex set of a general graph; see also [5]. Initially the partition will give two disjoint subsets, but repeated application will provide a hierarchical clustering tree with branches of degree 2, the root of the tree corresponding to $V$ and the leaves corresponding to the cluster sets.

Thus initially we seek to take a graph $G = (V, E(V), \beta)$ and partition the vertex set $V$ into two subsets $U$ and $W$ such that the sum of weights of edges connecting $U$ and $W$, defined to be $w(U,W) = \sum_{x \in U, y \in W} \beta(\{x,y\})$, is minimized. More specifically, we seek to minimize the *cut ratio*

$$r = \frac{w(U,W)}{|U|.|W|}.$$

We begin with the work of Hall [6], where $G = (V, E(V), \beta)$ is a graph embedded in $\mathbb{R}^2$, and hence each vertex $v_i$ corresponds to a point $(x_i, y_i) \in \mathbb{R}^2$. The vector $X^T = (x_1, \ldots, x_m)$, where $x_i \leq x_j$, defines an ordering on the vertex set $V = (v_1, \ldots, v_m)$. Hall seeks to reposition the vertices to minimise the sum of the edge weights times the squared distances between the corresponding $x$-coordinates of $X^T$, minimizing

$$z = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} (x_i - x_j)^2 a_{ij}.$$

Hall's method clusters together those vertices which are "strongly" connected.

We note that since $A$ is symmetric

$$\begin{aligned} z &= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(x_i - x_j)^2 a_{ij} \\ &= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(x_i^2 - 2x_i x_j + x_j^2)a_{ij} \\ &= \frac{1}{2}\left(\sum_{i=1}^{m} x_i^2 a_i - 2\sum_{i=1}^{m}\sum_{j=1}^{m} x_i x_j a_{ij} + \sum_{j=1}^{m} x_j^2 a_j\right) \\ &= \sum_{i=1}^{m} x_i^2 a_i - \sum_{i=1}^{m}\sum_{j=1}^{m} x_i x_j a_{ij} \\ &= X^T DX - X^T AX \\ &= X^T BX. \end{aligned}$$

Hence to minimize the weighted sum of the squares of the distance between the $x$-coordinates of the vertices we seek to minimize the expression $X^T BX$, where $B$ is symmetric, positive semidefinite.

Hagen and Kahng [4] extend these ideas to study general graphs, not just those which are embedded in the plane.

So let $G = (V, E(V), \beta)$ be any graph and $\{U, W\}$ be any partition of the vertex set $V = \{v_1, \ldots, v_m\}$ into two subsets; that is, $V = U \cup W$ and $U \cap W = \emptyset$. Set $p = |U|/m$ and $q = |W|/m$ and let $X^T = (x_1, x_2, \ldots, x_m)$ be a vector of length $m$ with coordinates defined by

$$x_i = \begin{cases} q, \text{ if } v_i \in U, \text{ and} \\ -p, \text{ if } v_i \in W. \end{cases} \tag{2}$$

Note that $p+q = |U|/m + |W|/m = (|U|+|W|)/m = 1$, $X^T \mathbf{1} = qpm + (-p)qm = 0$, and

$$|x_i - x_j| = \begin{cases} 0, \text{ if } v_i, v_j \in U \\ 1, \text{ if } v_i \in U, v_j \in W \\ 1, \text{ if } v_i \in W, v_j \in U \\ 0, \text{ if } v_i, v_j \in W. \end{cases}$$

Also note

$$w(U, W) = \frac{1}{2}\sum_{\{v_i, v_j\} \in E(V)} (x_i - x_j)^2 a_{ij}.$$

Moreover

$$\begin{aligned} |X|^2 = \sum_{i=1}^{m} x_i^2 &= (q)^2 pm + (-p)^2 qm = pqm(q+p) \\ &= \frac{pqm^2}{m} = \frac{|U||W|}{m}. \end{aligned}$$

So returning to Hall's analysis, where $\frac{1}{2}\sum_{\{v_i,v_j\}\in E(V)}(x_i - x_j)^2 a_{ij} = X^T BX$,

$$r = \frac{w(U,W)}{|U||W|} = \frac{\sum_{\{v_i,v_j\}\in E(V)}(x_i - x_j)^2 a_{ij}}{2m|X|^2} = \frac{X^T BX}{m|X|^2}. \tag{3}$$

Hence to minimize $r$ we need to minimize $X^T BX/|X|^2$, where $X$ satisfies the conditions given in (2) and $X^T \mathbf{1} = 0$.

We obtain an approximation to this by using Lagrange multipliers to minimize $X^T BX$ amongst $X$ with $X^T X = 1$ and $X^T \mathbf{1} = 0$. Thus given that $B$ is symmetric and $X^T BX = \sum_{ij} x_i b_{ij} x_j$, we have

$$\frac{\partial}{\partial x_s}\left(X^T BX\right) = \sum_j B_{sj}x_j + \sum_i x_i B_{is}, \text{ and}$$
$$\nabla X^T BX = BX + B^T X = 2BX.$$

Consequently, for the Lagrangian $L = X^T BX - \lambda(X^T X - 1) - \mu X^T \mathbf{1}$, $\nabla L = 2BX - 2\lambda X - \mu\mathbf{1}$. After noting that $B\mathbf{1} = \mathbf{0}$ and so $\mathbf{1}$ is an eigenvector with eigenvalue 0, the optimal solution occurs when $0 = (B - \lambda I)X$ and non-trivial solutions for $X$ can be obtained by calculating the eigenvalues $\lambda_i$ of $B$ and associated eigenvectors. Premultiplying by $X^T$ gives $0 = X^T BX - \lambda X^T X$ and since it is assumed that $X^T X = 1$,

$$\lambda = X^T BX = z.$$

Thus (see the Courant-Fischer Minimax principle, [2], page 106) the second eigenvalue

$$\lambda = \min_{X\perp 1, X\neq 0} \frac{X^T BX}{|X|^2}$$

and so Equation (3) implies

$$r = \frac{w(U,W)}{|U||W|} \geq \frac{\lambda}{m}.$$

This suggests $r$ can be minimized by using the *Fiedler eigenvector* corresponding to the second smallest eigenvalue $\lambda$. Further, this eigenvalue can be used to determine the coordinates of the "position" vector $X$ which satisfies the conditions given in (2). That is, $z = \frac{1}{2}\sum_{\{v_i,v_j\}\in E(V)}(x_i - x_j)^2 a_{ij} = X^T BX$ is minimized by the Fiedler eigenvector, with norm 1, and so intuitively the best approximation to this minimum will be obtained by assigning values to $X$ which reflect the differences in the coordinates of the eigenvector and satisfy the conditions given in (2). More precisely, if the difference for two coordinates of the eigenvector is small then the difference between the corresponding coordinates given in (2) should be small. Conversely, if the difference for two coordinates of the eigenvector is large then the difference between the corresponding coordinates given in (2) should be large.

So to determine a partition of the vertices of $V$ into two subsets, the components of the Fiedler vector are ordered in ascending numerical value giving $\mathcal{F} = (f_1, \ldots, f_m)$. The same ordering is applied to the vector $\mathcal{V}$ to obtain a new ordering of the vertex set, or equivalently a new vector $\mathcal{V}^+$. This vector can now be split into subvectors reflecting the partitioning of $V$ into two subsets $U$ and $W$ such that the sum of the weights of the edges joining vertices from different subsets is minimized. The value $|U|$ is termed the *splitting index*. A heuristic for determining the splitting index is discussed in the next section.

Once $U$ and $W$ are determined using the splitting index, the process is repeated for the induced graphs $G(U) = (U, E(U), \beta)$ and $G(W) = (W, E(W), \beta)$.

## 5 Splitting Index

Hagen and Kahng [4] propose four heuristics for determining the splitting index:

(i) partition $\mathcal{V}^+ = (v_1^+, \ldots, v_m^+)$ based on the sign of the corresponding component in the Fiedler vector; that is, $U = \{v_i^+ \mid f_i \in \mathcal{F}, f_i < 0\}$ and $W = \{v_i^+ \mid f_i \in \mathcal{F}, f_i \geq 0\}$;
(ii) partition $\mathcal{V}^+$ around the median value of $\mathcal{F}$;
(iii) partition $\mathcal{V}^+ = (v_1^+, \ldots, v_m^+)$ by determining the maximum difference between consecutive components of $\mathcal{F}$; that is, $a_i = |f_i - f_{i+1}|$, $1 \leq 1 \leq m-1$, $U = \{v_1^+, \ldots, v_t^+ \mid a_t = \max\{a_i\}\}$ and $W = \{v_{t+1}^+, \ldots, v_m^+\}$;
(iv) partition $\mathcal{V}^+$ to obtain the least cut ratio; that is, for $1 \leq i \leq m-1$, calculate $r_i = w(U_i, W_i)/|U_i||W_i|$, where $U_i = \{v_1^+, \ldots, v_i^+\}$ and $W_i = \{v_{i+1}^+, \ldots, v_m^+\}$, then set $U = \{v_1^+, \ldots, v_t^+ \mid r_t = \min\{r_i\}\}$ and $W = \{v_{t+1}^+, \ldots, v_m^+\}$.

It is suggested in [4] that method (iv) be used. However, our research indicates that the above methods are susceptible to singularities and thus we will follow the work of Hopcroft, Khan, Kulis and Selman, [7], by investigating instabilities in the data and thus determining the most appropriate heuristic.

Full details of the data set are given in Section 6, however for a random selection of graphs the following observations were made. The graphs contained a large number of pendant vertices (vertices incident with at most one edge). For these graphs heuristics (i), (iii) and (iv) were equivalent. Thus heuristics (ii) and (iv) were tested on a number of randomly selected graphs, then the pendant vertices were removed. When heuristic (iv) was applied the pendant vertices dominated the majority of cluster sets and their removal reduced the clustering to a small number of cluster sets containing a large number of vertices. When heuristic (ii) was applied and the pendant vertices were removed the number of cluster sets did not change. At times some of the pendant vertices were collected together into the same cluster but at times they were fairly evenly distributed across the cluster sets. Hence given the data set, it was decided that heuristic (ii) should be applied to determine the splitting index.

## 6 Experimentation

In this section we apply the above procedure (using heuristic (ii)) to data obtained from an enterprise communication network[4]. This data set was obtained by placing probes on physical links and recording the volume of information in daily communications. Three types of statistics were gathered: sender and receiver identifications and a count of the TCP/IP traffic. The sender and receiver identifications were clustered into 328 business domains or vertices in the resultant network. Data was collected over a period of 102 days and this information gives rise to a series of graphs $G^1, \ldots, G^{102}$.

In [1], Bunke, Dickinson, Kraetzl and Wallis conducted a number of tests on the distance between consecutive graphs in the above sequence. We have selected two techniques proposed in [1], the Spectral Distance (see (1), Section 1) and the Edit Distance, and compared these techniques with the procedure presented in this paper. The results of these tests are given below, but first we provide a brief discussion of edit distance.

Let $G^h = (V, E^h(V), \beta^h)$ and $G^{h+1} = (V, E^{h+1}(V), \beta^{h+1})$, where $|V| = m$ and let $B^h = [b^h_{ij}]$ and $B^{h+1} = [b^{h+1}_{ij}]$ represent the corresponding $m \times m$ disconnection matrices. Then the *edit distance* is given by

$$d(G^h, G^{h+1}) = \sum_{1 \le i < j \le m} |b^h_{ij} - b^{h+1}_{ij}| + \sum_{b^h_{ii}=0, b^{h+1}_{ii} \neq 0} 1 + \sum_{b^h_{ii} \neq 0, b^{h+1}_{ii}=0} 1.$$

That is, the sum of the differences in the weights of edges in $G^h$ and $G^{h+1}$ plus the number of vertices of non-zero degree which only occur in one of $G^h$ or $G^{h+1}$. This is a robust measure as it accurately records the overall change in communication traffic across the network.

The values of $d(G^h, G^{h+1})$, using each of the three techniques (spectral distance, edit distance and clustering techniques), has been calculated for $1 \le h \le 101$ and the results are displayed in the following graphs.

In the first of these graphs, we note that both the edit distance and the clustering procedure detect major changes in the network between days 20 and 25, 60 and 65, and 85 and 90. The spectral distance detects the first of these changes, but not the other two major perturbations. A more interesting aspect highlighted in the first graph is the differences between days 20 and 90. Other than the major changes mentioned above, the edit distance shows relatively little change from one day to the next. However the clustering technique tends to imply that during the same period there are changes in intergroup communications, indicating that the dynamics of the network may be changing through this period. These results suggest that the clustering technique presented here may be useful in detecting major changes in the overall communication traffic, but may also be useful in detecting changes in group dynamics.

[4] We wish to acknowledge the generous support of Miro Kraetzl, who among other things has given us access to the data set.

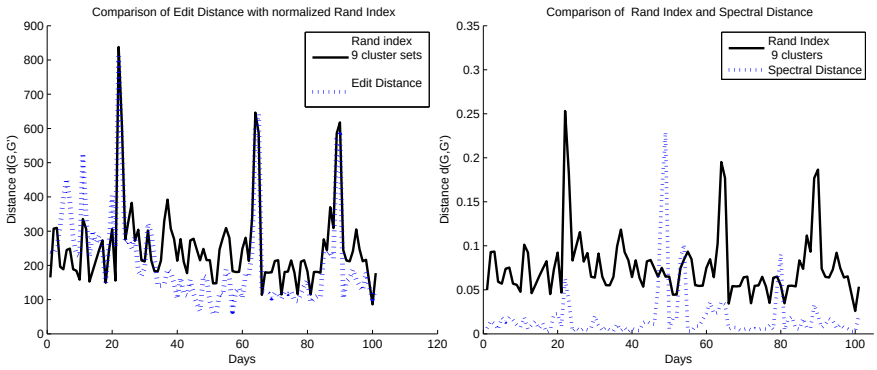



Figure 2: Time Series: Network Evolution

## 7 Conclusions and further research

The above results show that the Fiedler vector can be used to determine an intragraph clustering. Further, it may tentatively be suggested that this clustering technique may provide a method comparable to the edit distance for detecting change in networks. For the data tested here, there are some instances where the clustering technique detects changes which were not so obvious in the edit distance. These results are positive and indicate that further investigation of the technique is warranted, and may lead to more refined methods providing a robust tool for the detection of change within large enterprise networks. Possible areas for further investigation are an extended analysis of the heuristics used to determine the splitting index; an investigation of the use of the M-cut ratio as proposed by Shi and Malik [9]; an investigation of related algorithms which overcome the problem of a hierarchical clustering process, for instance some of the techniques proposed by Tolliver and Miller [10].

## References

1. Horst Bunke, Peter J Dickinson, Miro Kraetzl and Walter D Wallis: A Graph-Theoretic Approach to Enterprise Network Dynamics, Birkhäuser. Boston (2007)
2. John W Dettman: Mathematical methods in Physics and Engineering. 2nd edition McGraw-Hill New York (1969)
3. Peter J Dickinson, Horst Bunke, Arek Dadej and Miro Kraetzl: A novel graph distance measure and its application to monitoring change in computer networks. In the Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics. volume 3 Orlando, FL (2003) 33–338
4. Lars Hagen and Andrew B Kahng: New spectral methods for ratio cut partitioning and clustering. IEEE Transactions Computer-Aided Design Santa Clara CA (1992) 422–427
5. Lars W Hagen and Andrew B Kahng: Combining problem reduction and adaptive multistart: A new technique for superior iterative partitioning. IEEE Transactions on Computer-aided Design of Intergrated Circuits and Systems **16** No. 7 (1997) 709–717

6. Kenneth M Hall: An $r$-dimensional quadratic placement algorithm. Management Science **17** No. 3 (1970) 219–229
7. John Hopcroft, Omar Khan, Brian Kulis and Bart Selman: Tracking evolving communities in large linked networks. Proceedings of the National Academy of Sciences of the United States of America (2004) 1–5 doi:10.1073/pnas.0307750100
8. Antonio Robles-Kelly and Edwin R Hancock: Graph Edit Distance from Spectral Seriation. IEEE Transactions on Pattern Analysis and Machine Intelligence **27** No. 3 (2005) 365–378
9. Jianbo Shi and Jitendra Malik: Normalized Cut and Image Segementation. IEEE Transactions on Pattern Analysis and Machine Intelligence **22** No. 8 (2000) 888–905
10. David A Tolliver and Gary L Miller: Graph partitioning by spectral rounding: application in image segmentation and clustering. Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06) (2006) 1–8.
11. Douglas B West: Introduction to Graph Theory. Prentice-Hall Inc. Upper Saddle River (1996).

# TCAM representations of intervals of integers encoded by binary trees*

**Wojciech Fraczak**[1,3], **Wojciech Rytter**[2,4], and **Mohammadreza Yazdani**[3]

[1] Dépt d'informatique, Université du Québec en Outaouais, Gatineau PQ, Canada
[2] Inst. of Informatics, Warsaw University, Warsaw, Poland
[3] Systems and Computer Engineering, Carleton University, Ottawa ON, Canada
[4] Department of Mathematics and Informatics, Copernicus University, Torun, Poland

**Abstract.** We consider the problem of minimal representations of intervals of positive integers by Ternary Content Addressable Memory (TCAM). The integers are encoded (represented) as binary strings of the same length $n$ and a TCAM is a string-oriented representation in terms of unions of simple sets, called rules. Each rule is a concatenation (of length $n$) of singleton sets (i.e., single digits 0 and 1) or the set $\{0,1\}$ denoted by $*$. Two important encodings are lexicographic encoding (standard binary representation of integers) and binary reflected Gray encoding. We consider a family of encoding schemes, called dense-tree encodings, which includes both of them: each integer $i \in \{0,1,\ldots,2^n-1\}$ is represented as the label of the path from the root to the $i$-th leaf of a perfect binary tree $T$ of height $n$, whose edges are labeled by zeros and ones. A set $X \subseteq \{0,1,\ldots,2^n-1\}$ corresponds to the set $T[X] \subseteq \{0,1\}^n$ of binary strings representing integers in $X$ as branches of the tree $T$. We provide exact bounds (with respect to $n$) on the minimal sizes of TCAMs representing sets of strings $T[X]$, for an interval $X$. Three important cases are analyzed: $T$ can correspond to lexicographic, Gray or general dense-tree encoding. Some other issues related to the minimal sizes and number of *essential* rules of TCAMs are also investigated.

## 1 Introduction

The *Ternary Content Addressable Memory* (TCAM), [8, 7], is a type of associative memory with a highly parallel architecture which is used for performing very fast (constant time) table look up operations. The problem of interval representation by TCAMs appears in network processing engines where the header fields of each IP packet (e.g., source address, destination address, port number, etc.) should be matched under strict time constraints against the entries of an Access Control List (ACL) [2, 6, 11]. In these engines, an ACL is often represented by a TCAM. Each entry of the ACL defines either a single value or an interval of

* The research of the first, second, and the third author were supported by grants of NSERC, Polish Ministery of Science and Higher Education N 206 004 32/0806, and Ontario Graduate Scholarship, respectively.

values for the fields of the packet header. If an ACL entry defines only single values for all header fields, then it can be directly and very efficiently represented using a single TCAM rule. However, if the ACL entry defines some non-trivial intervals for some header fields of the packet, then it may need more than one TCAM rule [9, 10].

A TCAM can be defined as a two dimensional array of cells, where each cell carries one of the three values 0, 1, or $*$. Each row of this array is called a TCAM rule. An example of a TCAM is shown in Figure 1.

| | |
|---|---|
| 1 | 0 1 * * * |
| 2 | * 0 1 * * |
| 3 | * * * 0 1 |
| 4 | * * 0 1 * |
| 5 | 1 * * * 0 |

**Fig. 1.** A TCAM $\mathcal{T}$ of width 5 with 5 rules. $L(\mathcal{T}) = \{0,1\}^5 \setminus \{00000,\ 11111\}$.

A rule of a TCAM of width $n$ is a sequence $r = e_1 e_2 \dots e_n$, where $e_i \in \{0, 1, *\}$ for $i \in \{1, \dots, n\}$. It defines the following non-empty language $L(r)$:

$$L(r) \stackrel{\texttt{def}}{=} L(e_1)L(e_2)\dots L(e_n),$$

where $L(0) \stackrel{\texttt{def}}{=} \{0\}$, $L(1) \stackrel{\texttt{def}}{=} \{1\}$, and $L(*) \stackrel{\texttt{def}}{=} \{0, 1\}$. For example, $L(\texttt{0*1}) = \{0\} \cdot \{0, 1\} \cdot \{1\} = \{001, 011\}$.

We say that $r$ covers a set of strings $S$ if $\forall w \in S, w \in L(r)$. A TCAM $\mathcal{T}$ of width $d$ with $k$ rules defines the following partial mapping $\mathcal{T} : \{0,1\}^d \to \{1, \dots, k\}$. $\mathcal{T}(w) = i$ if and only if $w$ is a word of the language defined by the rule number $i$, and $w$ is not a word of the language defined by any rule with a number smaller than $i$.

The language $L(\mathcal{T})$ of a TCAM $\mathcal{T}$ is the union of the languages defined by its rules, i.e., the domain of the partial mapping $\mathcal{T}$. The number of rules in a TCAM is called the *size* of the TCAM.

For $a \in \{0, 1\}$, by $\overline{a}$ we will denote the *complement* of $a$, i.e., $\overline{0} \stackrel{\texttt{def}}{=} 1$ and $\overline{1} \stackrel{\texttt{def}}{=} 0$.

The problem of interval representation in TCAM is sometimes referred to as the problem of "range representation" in TCAM, [10, 9, 2]. In this paper we present a systematic approach for tackling this problem. For a family of encoding schemes, which includes the lexicographic encoding and the binary reflected Gray encoding, we provide exact bounds on the minimal sizes of TCAMs for intervals representation. We also provide bounds on the number of *essential* TCAM rules (prime implicants) for intervals in the lexicographic encoding and the binary reflected Gray encoding.

## 2 Dense-tree encodings of integers and interval sets

We define a *full* tree of height $n$ as a perfect binary tree of height $n$ such that each pair of sibling edges are labeled 0 and 1. The assignment of labels to the edges (two alternatives per each internal node) can be chosen arbitrarily.

Let $T_n$ be a full tree of height $n$. The label $w \in \{0,1\}^n$ of the path from the root to $i$-th leaf defines the $n$-bit encoding of number $i$, with 0 corresponding to the furthest left leaf of the tree. In that way, $T_n$ defines a bijection $T_n : \{0,1\}^n \hookrightarrow \{0,1,\ldots,2^n-1\}$, called an $n$-bit *dense-tree encoding*. The lexicographic encoding (i.e., standard unsigned binary encoding) and the binary reflected Gray encoding [3, 5] are two important examples of dense-tree encodings. They are presented in Figure 2 in forms of full trees for 4-bit encodings.

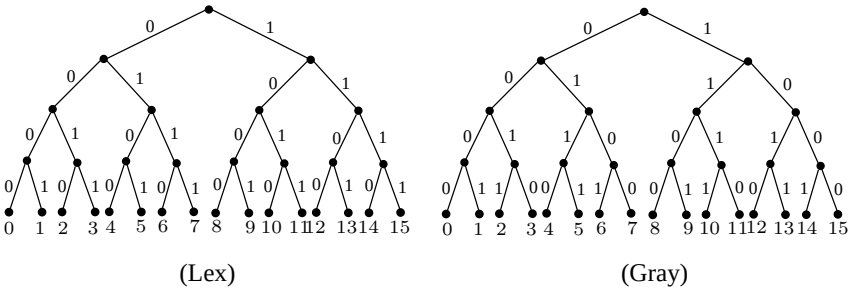



**Fig. 2.** Two sample 4-bit dense-tree encodings: Lexicographic encoding (`Lex`) and reflected Gray encoding (`Gray`).

In the context of a dense-tree encoding $T_n$, a set $X \subseteq \{0,1\}^n$ defines both the set $T_n(X)$ of integers and a subset of leaves of $T_n$. $X$ can be represented by a *skeleton tree* (see Figure 3). The skeleton tree of $X$ is obtained from $T_n$ by:

**(1)** removing all edges which are not leading to the leaves of $X$;

**(2)** secondly, turning into leaves all full subtrees.

We say that a skeleton tree $S$ is a *chain*, if every vertex of $S$ has at most one non-leaf child. A *double-chain* is a skeleton tree with at most one vertex $v$ having two non-leaf children and such that all ancestors of $v$ have only one child. Examples of a chain and a double-chain are illustrated in Figure 4.

For two integers $x, y$, we denote by $[x, y]$ the set $\{x, x+1, \ldots, y\}$ and call it an *interval*. A set of binary strings $X$ of length $n$ is an *interval-set* of a dense-tree encoding $T_n$ if $T_n(X)$ is an interval.

**Lemma 1.** *Let $X \subseteq \{0,1\}^n$. The following statements are equivalent:*

1. *There exists a dense-tree encoding $T_n$ for which $T_n(X) = [0, |X| - 1]$;*
2. *There exists a dense-tree encoding $T_n$ for which $T_n(X) = [2^n - |X|, 2^n - 1]$;*
3. *For any dense-tree encoding the skeleton tree of $X$ is a chain.*

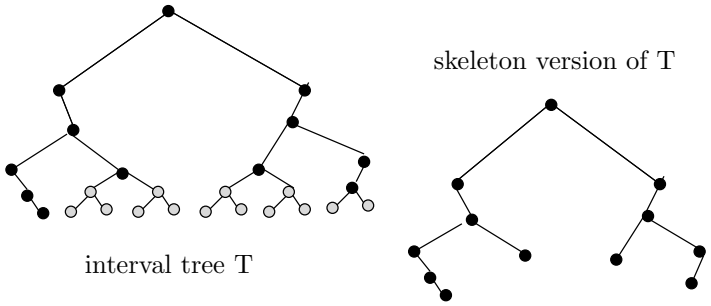



**Fig. 3.** An interval tree $T$ and its skeleton version.

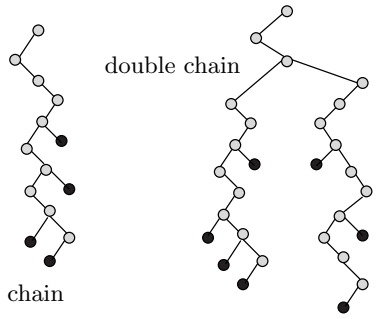



**Fig. 4.** A chain and a double-chain.

**Lemma 2.** *Let $X \subseteq \{0,1\}^n$. The following statements are equivalent:*

1. *There exists a dense-tree encoding $T_n$ for which $T_n(X)$ is an interval;*
2. *For any dense-tree encoding the skeleton tree of $X$ is a double-chain.*

**Corollary 1.** *There is a linear time algorithm to test whether there exists a dense-tree encoding $T_n$ for which $T_n(X)$ is an interval.*

In contrast the problem of determining whether a given TCAM defines an interval is intractable, as stated by the following lemma.

**Lemma 3.** *The problem of testing for a given TCAM $\mathcal{T}$ if $L(\mathcal{T})$ is an interval set is co-NP complete.*

*Proof.* Testing if a TCAM of width $n$ corresponds to a full tree is co-NP complete, since determining whether a given Boolean formula in a disjunctive normal form is a tautology is known to be co-NP complete. For a given TCAM $\mathcal{T}$, we construct the TCAM $\mathcal{T}' = \{\texttt{10}\} \cdot \mathcal{T} \cup \{*1^{n+1}\}$. TCAM $\mathcal{T}'$ defines an interval set if and only if $\mathcal{T}$ corresponds to a full tree. □

## 3 Prefix rules and essential rules of TCAMs

Let $E : \{0,1\}^n \hookrightarrow \{0,1,\ldots,2^n-1\}$ be an encoding of integer values by $n$-bit strings. Any subset $S \subseteq \{0,1,\ldots,2^n-1\}$ can be represented by a TCAM $\mathcal{T}$ of width $n$, i.e., $\mathcal{T}$ *represents* $S$ iff $E^{-1}(S) = L(\mathcal{T})$.

The problem of finding a minimal size TCAM $\mathcal{T}$ (i.e., a TCAM with the minimal number of rules) for a given set $S$ is known to be NP-hard (as it corresponds to the problem of finding a minimal disjunctive normal form for a Boolean expression). However, in this paper we are interested only in subsets which are intervals.

A TCAM rule is called a prefix rule if all non-star symbols occur as a prefix of it, e.g., `0101****` is a prefix rule.

**Theorem 1.** *Let $X \subseteq \{0,1\}^n$ be an interval-set of a dense-tree encoding $T_n$. The minimum number of prefix rules covering $X$ equals the number of leaves in the skeleton tree of $X$ in $T_n$.*

**Theorem 2.** *Let the skeleton tree of $X \subseteq \{0,1\}^n$ in a dense-tree encoding $T_n$ be a chain with $k$ leaves. The minimal size of a TCAM for $X$ is $k$.*

*Proof.* We first show that $k$ rules are sufficient for representing $X$. Let $k$ denote the number of leaves. Obviously $k$ prefix rules are enough, each rule corresponding to a leaf.

Now we show that we need at least $k$ rules (prefix or non-prefix). Assume, by contradiction, we have $k'$ rules $e_1, e_2, \ldots, e_{k'}$, with $k' < k$. Let $d = d[1]d[2]\ldots d[n]$ be the label of the path from the root to the sibling of the bottom leaf of the chain, which means that $d \notin X$. Each rule $e_i$ has to have at least one position $p_i$ such that $e_i[p_i]$ is $\overline{d[p_i]}$; we choose always the first such position. Since $k' < k$, there is a position $r$ which corresponds to an incoming edge of a leaf of the chain and which was not selected by any of positions $p_i$. We change the symbol on this position of $d$ to $\overline{d[r]}$. Then the resulting string belongs to the interval but it is not covered by any of the rules. □

**Corollary 2.** *In an $n$-bit dense-tree encoding, representing each of the intervals $[1, 2^n-1]$ and $[0, 2^n-2]$ needs exactly $n$ TCAM rules.*

Consider a set $X \subseteq \{0,1\}^n$. A TCAM rule $r$ is called an *$X$-limited* rule if it does not cover any string out of $X$, i.e., $L(r) \subseteq X$. An $X$-limited rule $r$ is said to be *$X$-essential* if there is no other $X$-limited TCAM rule that covers $r$. In the context of two-level logic generation, an $X$-essential TCAM rule is called a "prime implicant" of $X$ (see [1]). Prime implicants play an important role in the process of two-level logic generation. Any coverage of $X$ by a TCAM of size $k$ can be turned into a coverage by $k$ $X$-essential TCAM rules. Therefore, one can consider only $X$-essential TCAM rules in the process of finding a minimal coverage of $X$.

In the general case of two-level logic minimization, the number of prime implicants for a given set may be exponential with respect to the number $n$ of

input bits [1]. In the following, we prove that in the case of the $n$-bit lexicographic encoding and $n$-bit reflected Gray encoding, an interval-set $X \subseteq \{0,1\}^n$ admits no more than $n(n-1)+1$ and $2n$ $X$-essential TCAM rules, respectively.

**Lemma 4.** *Let $X \subseteq \{0,1\}^n$ such that the skeleton tree of $X$ in a dense-tree encoding $T_n$ is a chain with $k \leq n$ leaves. There are exactly $k$ different $X$-essential TCAM rules.*

*Proof.* It can be verified that every $X$-essential rule is of form $s_1 s_2 \dots s_i \dots s_n$, where $i$ is the level of a leaf in the chain and, for $1 \leq j \leq n$,

$$s_j = \begin{cases} \overline{d[i]} & \text{if } i = j \\ d[j] & \text{if } j < i \text{ and there is no leaf at level } j \\ * & \text{otherwise} \end{cases}$$

where $d = d[1]d[2]\dots d[n]$ is the path from the root to the sibling of the bottom leaf of the chain. □

**Theorem 3.** *Let $X \subseteq \{0,1\}^n$ be an interval-set in the $n$-bit lexicographic encoding $\mathtt{Lex}_n$. There is at most $n(n-1)+1$ different $X$-essential TCAM rules.*

*Proof.* Let $\mathtt{Lex}_n(X) = [x, y]$. The proof is by induction on $n$. Suppose that $\mathtt{Lex}_n(0u) = x$ and $\mathtt{Lex}_n(1v) = y$ for some $u, v \in\subseteq \{0,1\}^{n-1}$. The easy case when $x$ and $y$ encodings are starting by the same digit is skipped.

The set $P([0u, 1v])$ of all $X$-essential TCAM rules can be split into three disjoint sets $P_0$, $P_1$, and $P_*$, where $P_a$, $a \in \{0, 1, *\}$, denotes the set of all $X$-essential TCAM rules that start with $a$. By Lemma 4, we have: $|P_0| \leq n-1$ and $|P_1| \leq n-1$. Also $|P_*| = |P([u, v])|$, where the interval $[u, v]$ is encoded by $n-1$ bits. Thus, $p(n) \leq 2(n-1) + p(n-1)$, with $p(1) = 1$, where $p(i)$ denotes the maximum number of different $X$-essential TCAM rules for any interval $I$ with $\mathtt{Lex}_i(X) = I$. □

**Theorem 4.** *Let $X \subseteq \{0,1\}^n$ be an interval-set in the reflected Gray encoding $\mathtt{Gray}_n$. There is no more than $2n$ different $X$-essential TCAM rules.*

*Proof.* Let $\mathtt{Gray}_n(X) = [x, y]$ be such that encodings of $x$ and $y$ differ in the first digit. We assume that the dense-tree encoding is "reflective", i.e., the labels of the right-hand side and the left-hand side subtrees rooted at every node are the mirror copy of each other. Suppose that $x < y$, $x$ is encoded as $av$, $y$ as $\overline{a}u$, $a \in \{0,1\}$, and $u, v \in \{0,1\}^{n-1}$. Then by reflective property, $au$ encodes $2^n - 1 - y$.

In case when $x \leq 2^n - 1 - y$, there is no $X$-essential TCAM rule which starts by $\overline{a}$, and symmetrically, if $x \geq 2^n - y$ then there is no $X$-essential TCAM rule which starts by $a$. Without loss of generality we may assume that $x \leq 2^n - 1 - y$. Therefore, $X$-essential TCAM rules can be split into two sets $P_a$ (i.e., the set of rules starting by $a$), and $P_*$ (i.e., those rules that start by $*$). Every rule from $P_a$ with initial $a$ removed must be an essential TCAM rule for interval $[x, 2^{n-1} - 1]$ on $n-1$ bits, and every rule from $P_*$ with initial $*$ removed must be an essential

TCAM rule for interval $[2^n - 1 - y, 2^{n-1} - 1]$ on $n-1$ bits. Since the skeleton trees of both these intervals are chains (or empty), by Lemma 4 each of them has at most $n-1$ (or none if empty) different essential TCAM rules. □

## 4 Bounds on the size of a TCAM representing an interval

The minimum number of prefix rules required for representing an interval in TCAM is equal to the number of leaves in its corresponding skeleton tree. For some intervals, this number can be significantly reduced by using non-prefix TCAM rules.

**Theorem 5.** *There is a dense-tree encoding $T_n$ and an interval $I$ such that the minimum number of prefix rules representing $I$ is $2n-2$ and the minimum TCAM size is only $n$.*

*Proof.* Consider the interval $I = [1, 2^n - 2]$ in the lexicographic encoding on $n$ bits. The skeleton tree for $X \subset \{0,1\}^n$ such that $\mathtt{Lex}_n(X) = I$ has $2n-2$ leaves and thus by Theorem 1 it cannot be covered by less than $2n-2$ prefix rules. However $X$ can be covered by the following $n$ rules: $X = \{rot^k(01*\ldots*) \mid 0 \geq k < n\}$, where $rot$ denotes the left-rotation of a word, i.e., $rot(aw) \stackrel{\mathtt{def}}{=} wa$, for $a \in \{0,1,*\}$ and $w \in \{0,1,*\}^{n-1}$. (An example of $X$ for $n = 5$ is presented in Figure 1.) □

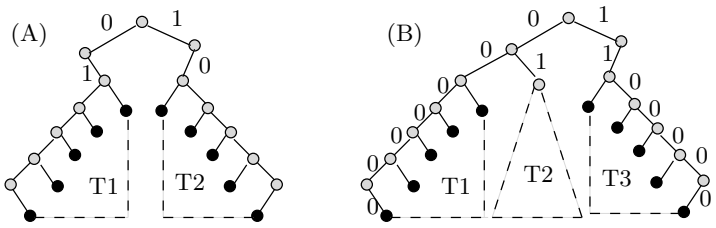



**Fig. 5.** The lexicographic tree in (A) needs at least $2n-4$ rules for lexicographic coding, the (non-lexicographic) tree in (B) needs at least $2n-3$ TCAM rules.

**Theorem 6.**
**(a)** *Let $T_n$ be a lexicographic or reflected Gray encoding of length $n$. There is an interval $I = [x, y]$ which cannot be represented with less than* $\max(n, 2n-4)$ *TCAM rules.*
**(b)** *For each $n \geq 1$ there exists a dense-tree encoding $T_n$ and an interval $I = [x, y]$ which needs* $\max(n, 2n-3)$ *TCAM rules.*

*Proof.* **Point (a)**. For $1 \leq n \leq 3$, we have $\max(n, 2n-4) = n$ and the theorem's claim was proved in Corollary 2. For $n \geq 4$, we provide a proof for

the lexicographic encoding; the proof for the reflected Gray encoding is similar to this proof. Consider the interval $I = [r_l, r_h] = [\mathtt{Lex}_n(s_l), \mathtt{Lex}_n(s_h)] = [\mathtt{Lex}_n(0100\ldots001), \mathtt{Lex}_n(1011\ldots110)]$ whose corresponding interval-set has a skeleton tree as illustrated in Fig. 5(A) for the lexicographic encoding. $s_l$ starts with 01, followed by $n-3$ zeros and ends by 1. $s_h$ starts with 10, followed by $n-3$ ones and ends by 0. We divide this interval to two sub-intervals $I_1 = [r_l, 2^{n-1}-1] = [\mathtt{Lex}_n(0100\ldots001), \mathtt{Lex}_n(0111\ldots11)]$ and $I_2 = [2^{n-1}, r_h] = [\mathtt{Lex}_n(100\ldots00), \mathtt{Lex}_n(1011\ldots110)]$. The encodings of all values in the interval $I_1$ start with 01 and thus they are within the first half of the domain $[0, 2^n-1]$. This means that all the leaves corresponding to the values in $I_1$ fall on the left side of the vertical line that passes through the root of the tree. The encodings for the values of interval $I_2$, however, start with 10 and so their corresponding leaves fall on the right side of the vertical line. This means that any TCAM rule that represents a value from $I_1$ cannot represent any value from $I_2$ and vice verse. As such, the minimum number of TCAM rules required for representing $I$ is equal to the addition of the minimum number of rules that are required to represent each of the intervals $I_1$ and $I_2$, separately.

Since the encodings of all the values from $I_1$ start with 10, we can represent this interval by prepending 10 to the $n-2$ rules that are required to represent the interval $[1, 2^{n-2}-1] = [\mathtt{Lex}_n(00\ldots001), \mathtt{Lex}_n(11\ldots11)]$ over the domain $[0, 2^{n-2}-1]$. Hence, using Corollary 2, we need at least $n-2$ TCAM rules of width $n-2$ to represent this interval. Similarly, it can be seen that representing $I_2$ needs at least $n-2$ TCAM rules. This means that the interval $I$ cannot be represented by less than $2n-4$ TCAM rules.

**Point (b)**. An example of such a dense-tree encoding $T_n$ and the interval $[1, 2^{n-1}+2^{n-2}-2]$ is shown in Figure 5(B). □

**Theorem 7.** *Let $T_n$ be a dense-tree encoding of length $n$. Every interval $I = [x, y]$ can be represented by* $\max(n, 2n-3)$ *TCAM rules.*

*Proof.* If $1 \leq n \leq 3$, we have $\max(n, 2n-3) = n$ and it can be checked manually that every interval can be represented by $n$ or less TCAM rules. For $n > 3$, if the skeleton tree of $I$ has $2n-3$ or less leaves, then by Theorem 1 $I$ can be represented by $2n-3$ or less prefix TCAM rules. The skeleton tree has maximal number of $2n-2$ leaves iff it has a shape similar to tree $T$ of Figure 6. We decompose this tree into three subtrees $T_1$, $T_2$, and $T_3$. The subtrees $T_1$ and $T_2$ are chains with $n-3$ leaves each, and thus altogether they need $2n-6$ TCAM rules. It is enough to show that $T_3$ needs only 3 TCAM rules. Let $abc$ ($\overline{a}b'c'$) be the labels of the left (resp., right) branch of $T_3$, as shown in Figure 6. We consider all possible cases as follows. If $b'c' = \overline{b}\overline{c}$, then rules $*\mathtt{b}\overline{\mathtt{c}}$, $\overline{\mathtt{a}}*\mathtt{c}$, and $\mathtt{a}\overline{\mathtt{b}}*$ represent $T_3$; If $b'c' = \overline{b}c$, then rules $**\overline{\mathtt{c}}$, $\overline{\mathtt{a}}\mathtt{bc}$, and $\mathtt{a}\overline{\mathtt{b}}\mathtt{c}$ represent $T_3$; If $b'c' = bc$, then rules $*\overline{\mathtt{b}}\overline{\mathtt{c}}$, $\mathtt{a}\overline{\mathtt{b}}\mathtt{c}$, and $\mathtt{ab}\overline{\mathtt{c}}$ represent $T_3$. □

**Theorem 8.** *Let $T_n \in \{\mathtt{Lex}_n, \mathtt{Gray}_n\}$ be the lexicographic or the reflected Gray encoding of length $n$. Every interval $I = [x, y]$ can be represented by* $\max(n, 2n-4)$ *TCAM rules.*

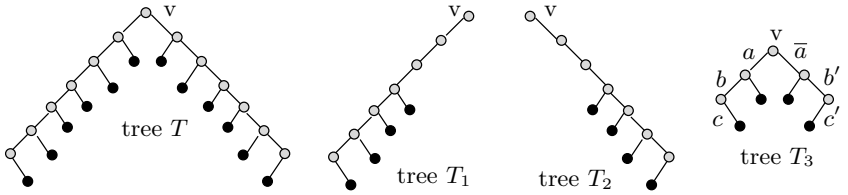



**Fig. 6.** The case when skeleton tree has maximal number of prefix rules $2n-2$. $T$ is decomposed into $T_1$, $T_2$, and $T_3$. For $T_1$ and $T_2$ we use $2n-6$ prefix rules. For $T_3$ three general rules are enough (instead of 4 prefix rules).

*Proof.* We only show the proof for the lexicographic encoding; the proof for the Gray encoding is similar. For $n \leq 4$, it can be manually verified that every interval $I$ can be represented by $n$ or less TCAM rules. For $n \geq 5$, if the interval tree has $2n-2$ leaves, then by proof of Theorem 5 it can be represented by $n$ TCAM rules. If the skeleton tree has $2n-3$ leaves, then it has one of the formats shown in Figure 7, where $C_l$ and $C_r$ are chains. In all these three cases it can be proved that the leaf labeled by letter $B$ can be represented by a combination of the TCAM rules that represent the other labeled leaves and thus (by Theorem 1) the whole interval can be represented by $2n-4$ TCAM rules. For instance, in the skeleton tree of Figure 7-(a), changing the second bit and the first bit of the prefix rules that represent the leaves $A$ and $D$, respectively, to $*$ results in two rules that represent $A'$ and $D'$, either. As such, representing leaf $B$ does not

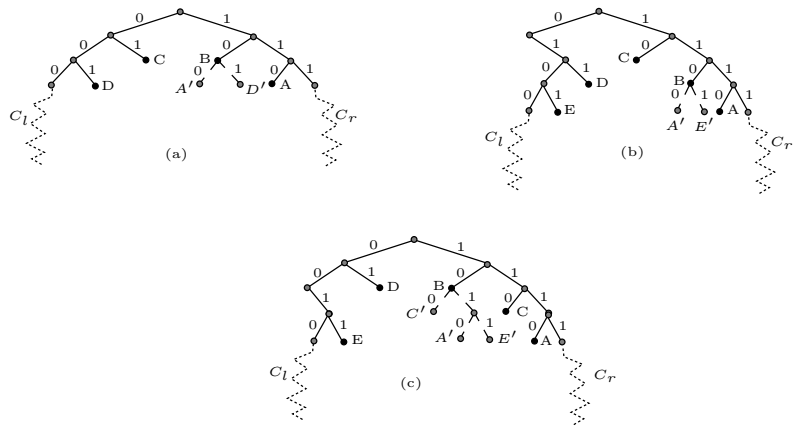



**Fig. 7.** Three possible forms of a skeleton tree with $2n-3$ leaves, for $n \geq 5$.

need a separate TCAM rule. □

## 5 Conclusion and future work

In this paper we studied the problem of interval representation by TCAM. This problem appears in the Internet routers that classify and filter the packets by implementing ACL policy rules. The available methods of interval representation by TCAM use the lexicographic encoding to encode the integers of a given interval. We broadened this choice and explored the problem assuming that the underlying encoding scheme belongs to the family of dense-tree encoding schemes which includes the lexicographic encoding and the binary reflected Gray encoding.

We introduced the notion of a skeleton tree of a set $X \subseteq \{0,1\}^n$ and used it to devise a linear time algorithm which determines whether $X$ is an interval in some dense tree encoding scheme. Skeleton trees appear to be useful also in estimating the sizes of TCAMs. We also showed that the number of prime implicants (called here the *essential rules*) of a given interval set $X \subseteq \{0,1\}^n$ is at most $n(n-1)+1$ and $2n$ for the lexicographic encoding and the reflective Gray encoding, respectively.

Finally, we proved that for the class of dense-tree encoding schemes, every interval over the domain $[0, 2^n - 1]$ can be represented by $2n - 3$ TCAM rules and in addition, there are intervals over the domain $[0, 2^n - 1]$ which need at least $\max(n, 2n - 4)$ rules for their TCAM representation.

Suppose that we use a fixed $n-$bit encoding scheme for representing the integer values of a domain $D = [0, 2^n - 1]$. In the continuation of our work, we would devise an algorithm for finding a TCAM representation of a given interval over $D$ with minimum number of TCAM rules. This problem is a special case of finding a TCAM representation of an arbitrary subset of the integers of domain $D$ with minimal number of TCAM rules. The latter problem is an NP-hard problem as it can be polynomially reduced to the NP-complete problems of the Minimal Disjunctive Normal Form of a Boolean formula [4] or the Two-Level Logic Minimization [1].

## References

1. Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.
2. G. Davis, C. Jeffries, and J. van Lunteren. Method and system for performing range rule testing in a ternary content addressable memory, April 2005. US Patent 6,886,073.
3. E. N. Gilbert. Gray codes and paths on the $n$-cube. *Bell Systems Technical Journal*, 37:815–826, 1958.
4. J. F. Gimpel. A method of producing a boolean function having an arbitrarily prescribed prime implicant table. *IEEE Trans. Computers*, 14:485–488, 1965.
5. F. Gray. Pulse code communications, March 1953. US Patent 2,632,058.
6. Hae-Jin Jeong, Il-Seop Song, Taeck-Geun Kwon, and Yoo-Kyoung Lee. A multi-dimension rule update in a tcam-based high-performance network security system. In *AINA '06: Proceedings of the 20th International Conference on Advanced*

*Information Networking and Applications - Volume 2 (AINA'06)*, pages 62–66, Washington, DC, USA, 2006. IEEE Computer Society.
7. R. Kempke and A. McAuley. Ternary cam memory architecture and methodology, August 1996. US Patent 5,841,874.
8. T. Kohonen. *Content-Addressable Memories*. Springer-Verlag, New York, 1980.
9. Karthik Lakshminarayanan, Anand Rangarajan, and Srinivasan Venkatachary. Algorithms for advanced packet classification with ternary cams. In *SIGCOMM*, pages 193–204, 2005.
10. Huan Liu. Efficient mapping of range classifier into ternary-cam. In *HOTI '02: Proceedings of the 10th Symposium on High Performance Interconnects HOT Interconnects (HotI'02)*, page 95, Washington, DC, USA, 2002. IEEE Computer Society.
11. V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel. Fast and scalable layer four switching. In *Proceedings of ACM Sigcomm*, volume 28, pages 191–202, October 1998.

# Binary Trees, Towers and Colourings (An Extended Abstract)

Alan Gibbons[1] and Paul Sant[2]

[1] Department of Computer Science, King's College London, UK,
[2] Computing and Information Systems Department, University of Bedfordshire, UK

## 1 Introduction

This paper explores new avenues into an old problem. Although the famous *Four-Colour Problem* ($FCP$) of planar graphs has a published solution [6–8] with later improvements [9] there has always been a body of opinion that extant proofs are unsatisfactory, lacking conciseness and lucidity and requiring hours of electronic computation.

Historically, (see, for example, [10]) work on $FCP$ has contributed significantly to the body of graph theory and combinatorics and continues to do so, partly through papers like this. Tait [3] showed that 3-edge colouring of cubic bridgeless graphs was equivalent to $FCP$. Building indirectly on this work others (for example, [4, 5]) showed that $FCP$ was equivalent, by a linear-time reduction, to the so-called *Colouring Pairs of Binary Trees problem* ($CPBT$). This is our starting point. Specifically, the paper explores the notion of so-called *towers*in this context. Other papers cited in the references, by the same authors, investigate ($CPBT$ through other avenues.

Figure 1 shows an instance of $CPBT$. Such an instance always consists of two binary trees of the same size (that is, having the same number of leaves). For analytical reasons, we usually consider that the trees have so-called *root-edges* shown at the top of these structures. Any 3-edge colouring of a binary tree defines a sentence over the colours $\{a, b, c\}$ which is given by reading (from left to right) the colours of those edges which have leaves as endpoints. In this figure, for both trees, the sentence in *bbabb*. The sentence is said to *colour* the tree. In the example, *bbabb* is said specifically to $a$-colour both trees because $a$ is the colour of the root-edge in both cases. More formally, an $a$-colouring of a binary tree is one in which the colours of edges are assigned in such a way that the root-edge is forced (by the colours assigned to all other edges) to be $a$. For *any* pair of binary trees of the same size, $CPBT$ is to prove that there always exists at least one sentence that colours both trees.

Other problems in a variety of combinatorics (for example, in language, automata and integer linear equation theories, [4, 5]) have, in turn, been shown to be equivalent to $CPBT$ which adds to its interest. In [1, 2] the authors study $CPBT$ through the idea of rotations in trees and by looking at broad classes of $CPBT$ problem instances which are solvable independently from $FCP$. This paper takes a fresh look at $CPBT$ through the notion of *towers*. The following sections define towers and decomposition of binary trees into towers. They

also develop combinatorics and algorithmics appropriate to this initial study of towers in the context of $CPBT$.

Before embarking on that, we present a theorem which, for the first time, completely characterises the set of all the strings each of which $a$-colours at least one binary tree with $n$ leaves. In the theorem, $S_n$ is the cardinality of this set.

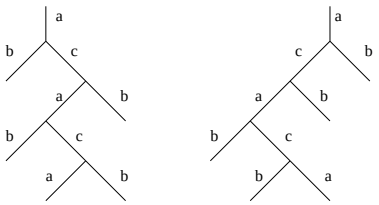



**Fig. 1.** A solution to $CPBT$ for two trees with 5 leaves

**Theorem 1.** *Let $S$ be a string of length $n$ ($> 1$) over the alphabet $\{a, b, c\}$ and let $A, B, C$ respectively be the number of $a$'s, $b$'s and $c$'s in $s$. Then:*

- *$S$ $a$-colours some binary tree with $n$ leaves if and only if*

1. *$S$ contains at least two of the characters $a$, $b$ and $c$.*
2. *for $n$ even, $A$ is even and $(B, C)$ are odd.*
   *for $n$ odd, $A$ is odd and $(B, C)$ are even.*

- *$S_n = (3^n - 1)/4$, $n$ even, and $S_n = (3^n - 3)/4$, $n$ odd*

*Proof.* (sketch) First part: In a 3-edge colouring of any 3-regular rootless tree $T$, the number of edges attached to leaves that are coloured $a$, $b$ or $c$ are either all even or all odd. This forces the parities of $A$, $B$ and $C$ of $S$ defining the colouring of any binary tree obtained by rooting $T$. On the other hand, given any $S$ over $\{a, b, c\}$, we can simulate a colouring of a binary tree by replacing recursively any pair of adjacent and differently coloured letters by the third letter, only if $S$ satisfies the conditions of the theorem for $a$-colouring.

Second part: It is easy to prove that of the $3^n$ distinct strings of length $n$ over $\{a, b, c\}$, $(3^n + 1)/2$ have $A$ even and $(3^n - 1)/2$ have $A$ odd. For $n$ even, the $(3^n + 1)/2$ strings with even $A$ includes the string consisting only of $a$'s. This is disallowed by (1) above, leaving $(3^n + 1)/2 - 1 = (3^n - 1)/2$ strings. Each of these contains at least one character that is not $a$, and each has either $(B, C)$ odd or $(B, C)$ even. We can form pairs of strings such that one string of a pair differs from the other in just one respect: the first character that is not an $a$ in one is a $b$ and in the other is a $c$. Clearly *every* one of the $(3^n - 1)/2$ strings can be accounted for in this way. It follows that for $n$ even, there are $(3^n - 1)/4$ strings with $A$ even, $(B, C)$ odd and containing at least two characters from $\{a, b, c\}$. For $n$ odd, the proof is similar but we start with $(3^n - 1)/2$ strings with $A$ odd.

**Corollary 1.** *In any instance of CPBT, each tree with n leaves can be a-coloured by $2^{n-1}$ distinct strings. It follows that the full space of all strings a-colouring trees of size n is therefore much too large to guarantee that the two sets of strings defined by two trees of such an instance will have a non-empty intersection.*

## 2 Towers

If *both* children of an internal node of a binary tree are leaves, then that internal node is said to be a *terminal*. A *tower* is a binary tree with exactly one terminal. Before going on to look at the decomposition of trees into towers and its relevance for $CPBT$, we present the following theorem which formulates the number of binary trees, $T_{k,n}$ that have $n$ leaves and $k$ terminals.

**Theorem 2.**

$$T_{k,n} = \frac{1}{k}\binom{2k-2}{k-1}\binom{n-2}{2k-2}2^{n-2k}$$

*Proof.* (brief sketch) A tree with $n$ vertices may be constructed from a tree with $(n-1)$ vertices by attaching a pair of edges (creating a terminal vertex) to any leaf. This process of adding an edge pair may (if the new terminal is *not* a child of an old terminal) or may not (if the new terminal is a child of an old terminal) create one more terminal in the new tree than in the old. Thus, a particular tree with $n$ leaves and $k$ terminals may be constructed from $k$ different smaller trees with $(n-1)$ leaves, some of which have $k$ terminals and some of which have $(k-1)$ terminals.

In a tree with $(n-1)$ leaves and $(k-1)$ terminals, there are $(n-2k+1)$ leaves that are not the children of terminals. In a tree with $(n-1)$ leaves and $k$ terminals, there are $2k$ leaves that are the children of terminals. The above considerations establish the following recurrence relation

$$T_{k,n} = ((n-2k+1)T_{k-1,n-1} + 2kT_{k,n-1})/k \tag{1}$$

In addition, and by inspection of the trees, we have that:

$$T_{1,2} = 1, T_{1,3} = 2, T_{1,4} = 4, T_{2,4} = 1, T_{1,5} = 8, T_{2,5} = 6$$

The closed form for $T_{k,n}$ in the theorem statement can now be found using standard means too long to include here. The reader may (tediously) verify that the closed form does indeed satisfy the recurrence relation and initial values.

### 2.1 Decomposition of Trees into Towers

There are several ways to decompose trees into towers. At this time it is not clear what strategy might best help, for example, to provide an algorithmic solution to $CPBT$ (this being a major motivator). Let a *join* define an internal vertex for which *both* children are *not* leaves. Natural methods for decomposition then

involve cutting edges that connect a join to a child (a copy of that edge being retained by the *join* and its child). Such methods might: **(a)** cut both edges from each join to its children, or **(b)** cut just one edge from each join to a child. We look briefly at both of these possibilities.

Cutting both edges produces more towers from a given tree, but the method seems to provide, in some contexts, greater analytic utility. For example, we can reproduce the result of Theorem 2 by considering the reverse process of such a decomposition. Suppose that a tree has $k$ terminals, then we can think of the towers produced by process (a) as super-edges in a "tree" of super-edges which has $k$ "leaves" and, in all, $(2k-1)$ super-edges arranged in one of the standard topologies of a normal $k$-leaved binary tree. This is illustrated in Figure 2. For the tree, $T$, of (a) the edge-cuts decomposing (that is, separating) the tree into towers are indicated. Its super-edge tree is shown in (b).

Now, suppose that we wish to count the number of trees with $n$ leaves and $k$ terminals. We can look at all the distinct ways of generating $(2k-1)$ towers and combining them through topologies such as that of Figure 2 (b). The following method does this, carefully avoiding the duplication of individual binary trees. We start with a large tower with $n$ leaves. Such a tower that will produce $T$ of Figure 2 (a) is shown in (c). Internal edges of the large tower are cut to produce the required $(2k-1)$ super-edges. Note that the stacking of super-edges is topologically ordered according to the digital labels of (b) and (c). How many

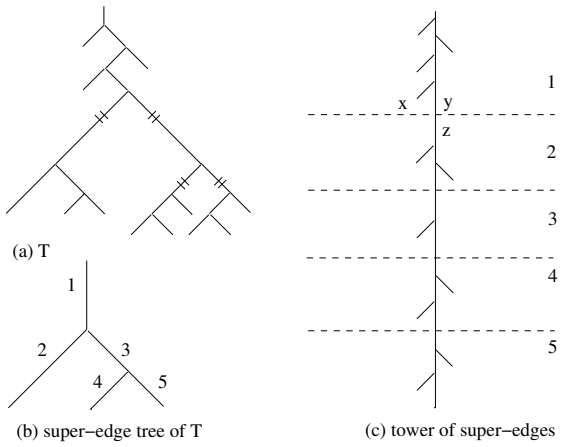



**Fig. 2.** Decomposition of a tree, $T$, into towers.

different trees with $n$ vertices and $k$ terminals can be constructed in this way for a fixed topology such as that of (b)? We start with a tower (such as (c)) with $n$ leaves, there are $2^{n-1}$ such towers. We cut (2k-2) internal edges of the tower to make $(2k-1)$ super-edges; there are (n-2) internal edges, so that these cuts can be made in $\binom{n-2}{2k-2}$ ways. However, the same set of super-edges can be produced by $2^{2k-2}$ different large towers which only differ in the connections made at the

cuts. For example, in Figure 2 (c), the topmost cut severs $y$ from $z$, but after the cut $z$ might just as well have been connected to $x$. Thus, removing duplications in the count, there are overall $\binom{n-2}{2k-2}\ 2^{n-2k}$ different ways of producing trees with $n$ leaves, $k$ terminals with a fixed topology (such as that indicated in (b)). Notice that we employ, for every such tree, the same mapping of super-edges in the tower to positions in the tree (exemplified by the digital labels in (b) and (c)) and this avoids duplications in those trees generated by the process. Now there are $\frac{1}{k}\ \binom{2k-2}{k-1}$ different topologies for the super-edge tree (this is the Catalan number giving the number of binary trees with $k$ leaves). Multiplying the Catalan number with the number of super-edge trees with a specific topology then reproduces the result of Theorem 2.

We very briefly look at the second option, (b), of decomposing a tree into towers. It is easily appreciated that this option (however the choice is made as to which edge of a pair is cut) produces a minimum number of towers (equal to the number of terminals in the tree). A lemma similar to the following also holds for the previous method of decomposition.

**Lemma 1.** *A decomposition of a binary tree into a minimum number of towers can be achieved in linear-time.*

*Proof.* (sketch) We employ a depth-first traversal of the tree which (on exiting a tower) can detect (on the basis of local information and in constant time) whether an edge should be cut. It is well-known that such a traversal takes linear time.

### 2.2 Towers and the *CPBT* problem

In deriving solutions to specific *CPBT* problems we shall be concerned to show that the *3-regular* graph, obtained by conjoining corresponding leaves of the trees, has an *even-cycle cover*. A 3-edge colouring then follows immediately by assigning the colours $a$ and $b$ to edges alternately around each cycle and assigning the colour $c$ to every other edge. The solution to $CPBT$ is then given by the series of colours assigned to the conjoined edges.

We first show that if one of the trees in a *CPBT* problem is a *left* (or *right*) spine, then a solution is easily found. A right (left) spine is, of course, just a tower with its terminal as parent of the rightmost (leftmost) pair of leaves.

**Theorem 3.** *If ($T_1$, $T_2$) is an instance of* CPBT *where $T_1$ is* any *binary tree and $T_2$ is a right (or left) spine, then this instance is solvable.*

*Proof.* We first show that the so-called *Halin* graph based on $T_1$ contains a Hamiltonian cycle. A Halin graph is constructed from $T_1$ by adding a circuit of edges, each connecting adjacent leaves of $T_1$ as indicated in Figure 3(a). Here, $T_1$ is enclosed in a so-called *skirt* of edges and $e$ is the root-edge. A Hamiltonian Circuit for any Halin graph can be inductively constructed by replacing vertices on the skirt, in turn, by triangles. This is indicated in Figure 3(b) where the starting point for any $T_1$ is the graph on the left of that figure. In that starting

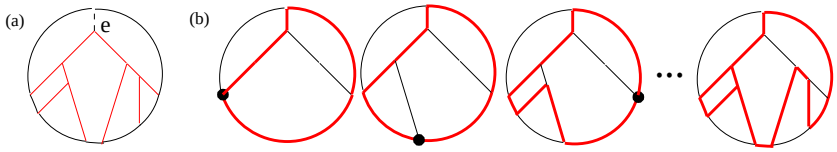



**Fig. 3.** A Halin Graph (a) and construction of its Hamiltonian Circuit (b).

position, the bold (Hamiltonian) circuit is the seed for our final Hamiltonian circuit. At each stage, a vertex (indicated by a heavy dot) is replaced by a triangle and the bold circuit is locally expanded to incorporate two sides of the triangle so that it remains Hamiltonian.

A 3-edge colouring of the Halin graph is now obtained by assigning the colours $a$ and $b$ to the Hamiltonian Circuit and $c$ to the other edges. By removing part of the skirt (for our example, this is shown in Figure 4(a)) we begin to construct the graph which is the conjugation of $T_1$ and a right- or left-spine. This produces four free ends: $W, X, Y$ and $Z$ in the figure. Notice that whatever $T_1$, the free ends $W$ and $X$ will be connected by a 2-coloured chain of edges (coloured $a$ and $b$ in our example) which does *not* contain $Y$ or $Z$. To complete the construction, we interchange the colours of this chain and conjoin (as will always be possible) the free ends $X$ and $Y$. We can now observe (see Figure 4(b)) the graph $T_1$ (above the dotted line) and a spine (below it). By obvious variation in construction, the choice of left- or right- spine is always possible. For our example, the solution to $CPBT$ is *babbbc*.

Unlike the case for Lemma 2, instances of $CPBT$ will generally consist of tree pairs neither of which is a tower. Our vision then first consists of a decomposition of these trees into towers. Then the towers are to be systematically reconnected and coloured. We might start with a pair of towers with some conjoined edges and add towers one or more at a time until the job is done. This process is too ambitious to be tackled in this initial study. However, a starting point has to be the solution of $CPBT$ for pairs of towers. This occupies the remainder of the paper. Incidentally (as the bigger picture requires), solving $CPBT$ for problem instances consisting of two towers also lets us colour pairs of towers where there is only partial overlap in terms of conjoined edges.

### 2.3 The lozenge and restricted lozenge graph

In general, two towers with corresponding leaves conjoined form the archetypical graph shown in Figure 5(a). The edges of one tree are indicated in bold and its longest path can be traced through the vertices $Y$ (its root), $e$, $f$ and $a$. Other edges form the second tree whose longest path is traced through $X$ (its root), $b$, $c$ and $d$. Note that for some pairs of trees the longest paths might be traced respectively by $(X, b, a, f)$ and $(Y, e, d, c)$, but the overall graph is always archetypically as shown. At the heart of the figure is the *lozenge* traced through the vertices $a, b, c, d, e$ and $f$. For this reason, we call such a graph a *lozenge*

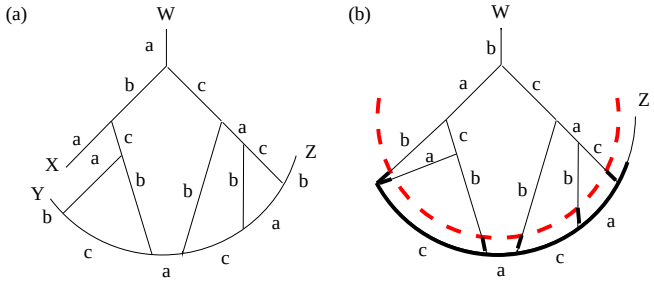



**Fig. 4.**

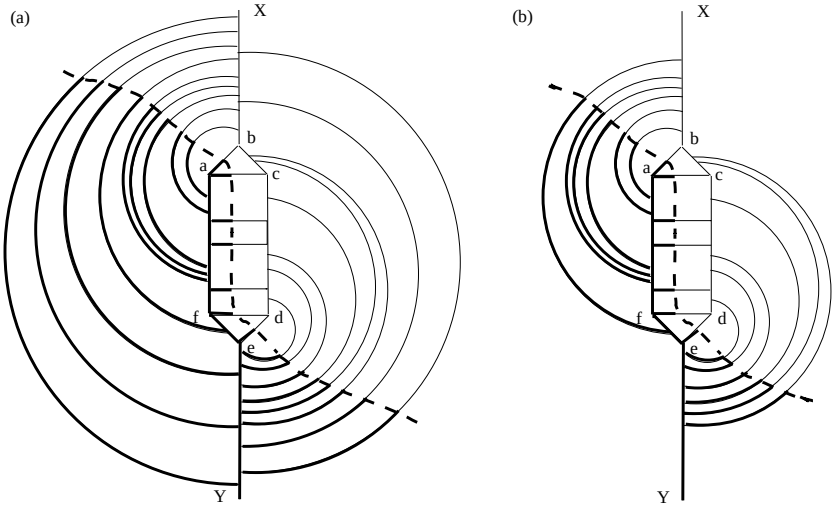



**Fig. 5.** Archetypical lozenge (a) and restricted lozenge (b) graphs.

*graph*. Within the lozenge are a number of horizontal edges which we call *slats*. If there are zero or one slats, then a Hamiltonian Path from $X$, through the lozenge and onto $Y$ is easy to trace. For these cases the $CPBT$ problem is trivially solved so that we will normally assume there are several or more slats.

Outside the lozenge, the edges are shown in a curved manner. In the figure, these edges on the left [right] form (as it were) concentric arcs away from the edge (a,b) [(e,d)]. The edges $(a, b)$ and $(d, e)$ are called the *free* edges, while $(b, c)$ and $(e, f)$ are called the *leaf-spanning edges* (they are actually segments on the longest path of one tree which span between the two leaves at maximum depth in the other tree) of the lozenge. Similarly, those edges defining the perimeter of the lozenge , which are not *free* or *leaf-spanning*, we will call *inter-slat* edges (although in the lozenge graph as a whole they will be segments of the longest paths of one of the trees). The rectangular shapes within the lozenge between successive slats will be called *boxes*. By the *degree* of a perimeter edge of the lozenge, we will mean the number of curved edges that have an endpoint on that edge; notice that the degree of the *free edges* is always zero.

If there are no curved edges on (say) the right-hand-side of Figure 5 (a), then one of the towers is a right- or left-spine and (by Lemma 3) the corresponding $CPBT$ problem is solvable. One approach to solve the $CPBT$ problem for any pair of towers might then be to start with such a coloured lozenge graph and attempt to add and colour the missing edges. However, we will take a different starting point which, we believe, will present fewer problems in extending to the general case. Given an instance of $CPBT$ on the lozenge graph as exemplified in Figure 5 (a), the corresponding problem on the *restricted* lozenge graph is obtained by *deleting all those curved edges that have no endpoint on the lozenge.* Of course, it is possible for our original instance of $CPBT$ to be already in this restricted form. In this case, what follows describes how a solution may be found, otherwise the restricted form is a useful starting point for solving the original problem. To be clear, Figure 5 (b) shows the archetypical form of an instance of $CPBT$ on the restricted lozenge graph.

**Lemma 2.** *Without loss of generality, we may assume that each leaf-spanning edge of a restricted lozenge graph has non-zero degree.*

*Proof.* (sketch) Suppose that a leaf-spanning edge has zero degree, then, locally, the restricted lozenge graph is depicted on the left of Figure 6(a). We assume that the number of slats is greater than 1, otherwise the graph is 3-edge colourable as described earlier. In Figure 6(a), the degrees of the interslat edges $(X, i)$ and $(Y, j)$ are 2 and 1 respectively, although the construction to be described works *whatever* their degrees. As Figure 6(a) indicates, we construct a smaller restricted lozenge by removing that part of the graph enclosed by the dotted curve and join the three loose edges left (crossing the curve) at a single new vertex $Z$ as shown on the right of Figure 6(a). If the smaller restricted lozenge is 3-edge colourable, then we show that the reconstructed graph is also 3-edge colourable. Notice that the edges meeting at $Z$ must be differently coloured. We therefore only need to show that the removed portion of the graph has a 3-edge colouring in which the corresponding edges are all differently coloured. That this is possible is indicated in Figure 6(b). Here (from left to right) the degree of

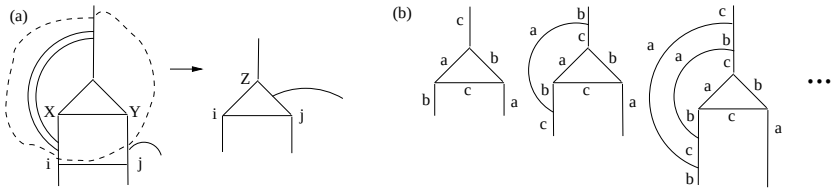



**Fig. 6.**

the original interslat edge $(X, i)$ is increased (in an obvious inductive manner) from 0 upwards until its original value is attained.

It is possible that (if the degree of the interslat edge $(Y, j)$ of the original graph is zero) our smaller restricted lozenge in Figure 6(a) will also have a leaf-spanning edge of zero degree. In this case we simply repeat the operation described above until this is not the case or until the number of slats is one.

**Lemma 3.** *Without loss of generality, we may assume that at least one of the leaf-spanning edges of a restricted lozenge graph has odd degree (and that the other has non-zero degree).*

*Proof.* (sketch) Suppose that one of the leaf-spanning edges has even degree. The situation is depicted on the left of Figure 7(a). Here the degree of $(X, Y)$

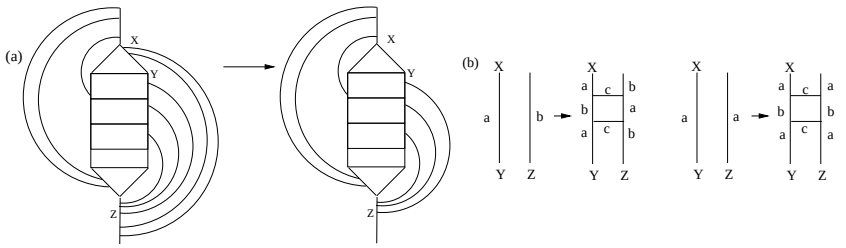



**Fig. 7.**

is 2, but the construction we describe works just as long as it is non-zero and even. As Figure 7(a) indicates, we construct a smaller restricted lozenge graph simply by deleting the edges contributing to the degree of the leaf-spanning edge $(X, Y)$. If the smaller graph is 3-edge colourable, then it is easy to obtain a 3-edge colouring of the reconstructed graph. Figure 7(b) shows how this is possible (for each pair of edges contributing to the even degree of $(X, Y)$) given that (on the left) the replaced edges connect differently coloured edges or (on the right) they connect similarly coloured edges. Notice that the deconstruction of Figure 7(a) yields a restricted lozenge graph with a leaf-spanning edge of zero-degree. We then apply the process of Lemma 2 to obtain an even smaller example with a non-zero degree leaf-spanning edge. If this degree is even, then we repeat the construction of this lemma and so on until we obtain either a restricted lozenge graph with a single slat or a restricted lozenge graph with a leaf-spanning edge of odd degree.

**Theorem 4.** *Every restricted lozenge instance of CPBT is solvable.*

*Proof.* In view of Lemma 3, we may assume that in our instance of *CPBT* at least one of its leaf-spanning edges has *odd* degree and that the other has non-zero degree. Figure 8 shows that a Hamiltonian cycle of the graph exists in this case. In this example, the leaf-spanning edge $(e, f)$ has degree 3, but provided that it is odd, then a similar construction is possible in which all edges that contribute to the degree also lie on the Hamiltonian Cycle. The construction works whatever the degree of the other leaf-spanning edge $(c, d)$.

## 3 Conclusion

We have proposed a new approach to an old problem ($FCP$) whose importance is emphasised by its links to a variety of areas in combinatorics and graph theory.

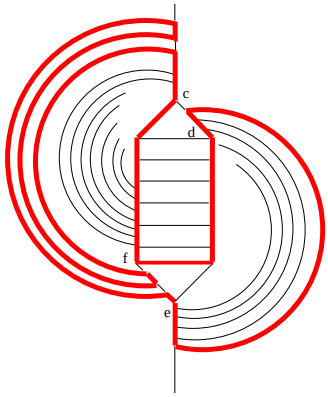



**Fig. 8.**

The lemmas and theorems of this paper are a substantial start to this work. We might next extend Theorem 3 to the full lozenge graph (perhaps relatively easily) and then fully consider issues of inductively combining and colouring towers for general instances of $CPBT$ (challenging). There are also many other challenging combinatorial problems (like finding the size of the set of strings each $a$-colouring some tower) in this general area.

## References

1. Alan Gibbons and Paul Sant, Rotation sequences and edge-colouring of binary tree pairs, Theoretical Computer Science, 326(1-3), pages 409-418, 2004.
2. Alan Gibbons and Paul Sant, Stringology and The Four Colour Problem of Planar Maps, String Algorithmics, Volume 2 (Texts in Algorithms), C. Illiopoulos and T. Lecroq (Eds.), King's College Publications,2004.
3. P.G. Tait, On the colouring of Maps, Proceedings of the Royal Society of Edindurgh (Section A), Volume 10, 501-503, 1880.
4. A. Czumaj and A.M. Gibbons, Guthrie's problem: new equivalences and rapid reductions, *Theoretical Computer Science*, Volume 154, Issue 1, 3-22, January 1996.
5. A.M. Gibbons. Problems on pairs of trees equivalent to the four colour problem of planar maps. Second British Colloquium for Theoretical Computer Science, Warwick, March 1986.
6. K. Appel and W. Haken. Every planar map is four colourable. Part I. Discharging. Illinois Journal of Mathematics 21 (1977), 429-490.
7. K. Appel, W. Haken and J. Koch. Every planar map is four colourable. Part II. Reducibility. Illinois Journal of Mathematics 21 (1977), 491-567.
8. K. Appel and W. Haken. Every planar map is four colourable. Comtemporary Mathematics 98 (1989) entire issue.
9. N. Robertson, D. P. Sanders, P.D.Seymour and R. Thomas. A new proof of the four-colour theorem. Electron. Res. Announc.Amer. Math. Soc. 2 (1996), no.1, 17-25 also: The four colour theorem, J. Combin. Theory Ser. B. 70 (1997), 2-44.
10. T. L. Saaty and P. C. Kainen. The Four-Colour Problem: Assaults and Conquest. McGraw-Hill (1977).

# A Compressed Text Index on Secondary Memory

Rodrigo González * and Gonzalo Navarro **

Department of Computer Science, University of Chile.
{rgonzale,gnavarro}@dcc.uchile.cl

**Abstract.** We introduce a practical disk-based compressed text index that, when the text is compressible, takes much less space than the suffix array. It provides very good I/O times for searching, which in particular improve when the text is compressible. In this aspect our index is unique, as compressed indexes have been slower than their classical counterparts on secondary memory. We analyze our index and show experimentally that it is extremely competitive on compressible texts.

## 1 Introduction and Related Work

Compressed full-text self-indexing [22] is a recent trend that builds on the discovery that traditional text indexes like suffix trees and suffix arrays can be compacted to take space proportional to the compressed text size, and moreover be able to reproduce any text context. Therefore self-indexes replace the text, take space close to that of the compressed text, and in addition provide indexed search into it. Although a compressed index is slower than its uncompressed version, it can run in main memory in cases where a traditional index would have to resort to the (orders of magnitude slower) secondary memory. In those situations a compressed index is extremely attractive.

There are, however, cases where even the compressed text is too large to fit in main memory. One would still expect some benefit from compression in this case (apart from the obvious space savings). For example, sequentially searching a compressed text is much faster than a plain text, because much fewer disk blocks must be scanned [25]. However, this has not been usually the case on indexed searching. The existing compressed text indexes for secondary memory are usually slower than their uncompressed counterparts.

A self-index built on a text $T_{1,n} = t_1 t_2 \dots t_n$ over an alphabet $\Sigma$ of size $\sigma$, supports at least the following queries:

- $count(P_{1,m})$: counts the number of occurrences of pattern $P$ in $T$.
- $locate(P_{1,m})$: locates the positions of all those $occ$ occurrences of $P_{1,m}$.
- $extract(l, r)$: extracts the subsequence $T_{l,r}$ of $T$, with $1 \leq l, r \leq n$.

* Funded by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.
** Partially funded by Fondecyt Grant 1-050493, Chile.

The most relevant text indexes for secondary memory follow:

- The String B-tree [7] is based on a combination between B-trees and Patricia tries. $locate(P_{1,m})$ takes $O(\frac{m+occ}{\tilde{b}} + \log_{\tilde{b}} n)$ worst-case I/O operations, where $\tilde{b}$ is the disk block size measured in integers. This time complexity is optimal, yet the string B-tree is not a compressed index. Its static version takes about 5–6 times the text size plus text.
- The Compact Pat Tree (CPT) [4] represents a suffix tree in secondary memory in compact form. It does not provide theoretical space or time guarantees, but the index works well in practice, requiring 2–3 I/Os per query. Still, its size is 4–5 times the text size, plus text.
- The disk-based Suffix Array [2] is a suffix array on disk plus some memory-resident structures that improve the cost of the search. We divide the suffix array into blocks of $h$ elements, and for each block store the first $m$ symbols of its first suffix. It takes at best $4 + m/h$ times the text size, plus text, and needs $2(1 + \log h)$ I/Os for counting and $\lceil occ/\tilde{b} \rceil$ I/Os for locating (in this paper $\log x$ stands for $\lceil \log_2(x + 1) \rceil$). This is not yet a compressed index.
- The disk-based Compressed Suffix Array (CSA)[17] adapts the main memory compressed self-index [24] to secondary memory. It requires $n(O(\log\log\sigma)+ H_0)$ bits of space ($H_k$ is the $k$th order empirical entropy of $T$ [18]). It takes $O(m \log_{\tilde{b}} n)$ I/O time for $count(P_{1,m})$. Locating requires $O(\log n)$ access per occurrence, which is too expensive.
- The disk-based LZ-Index [1] adapts the main-memory self-index [21]. It uses $8nH_k(T) + o(n \log\sigma)$ bits. It does not provide theoretical bounds on time complexity, but it is very competitive in practice.

In this paper we present a practical self-index for secondary memory, which is built from three components: for *count*, we develop a novel secondary-memory version of backward searching [8]; for *locate* we adapt a recent technique to locally compress suffix arrays [12]; and for *extract* we adapt a technique to compress sequences to $k$-th order entropy while retaining random access [11]. Depending on the available main memory, our data structure requires $2(m - 1)$ to $4(m - 1)$ accesses to disk for $count(P_{1,m})$ in the worst case. It locates the occurrences in $\lceil occ/\tilde{b} \rceil$ I/Os in the worst case, and on average in $cr \cdot occ/\tilde{b}$ I/Os, $0 < cr \leq 1$ is the *compression ratio* achieved: the compressed divided by original text size. Similarly, the time to extract $P_{l,r}$ is $\lceil (r - l + 1)/b \rceil$ I/Os in the worst case (where $b$ is the number of symbols on a disk block), multiplying that time by $cr$ on average. With sufficient main memory our index takes $O(H_k \log(1/H_k) n \log n)$ bits of space, which in practice can be up to 4 times smaller than suffix arrays. Thus, our index is the first in being compressed and at the same time taking advantage of compression in secondary memory, as its locate and extract times are faster when the text is compressible. Counting time does not improve with compression but it is usually better than, for example, disk-based suffix arrays and CSAs. We show experimentally that our index is very competitive against the alternatives, offering a relevant space/time tradeoff when the text is compressible.

**Algorithm** *count*$(P[1,m])$
$i \leftarrow m, c \leftarrow P[m], \text{First} \leftarrow C[c]+1, \text{Last} \leftarrow C[c+1]$;
**while** (First $\leq$ Last) **and** $(i \geq 2)$ **do**
    $i \leftarrow i-1$; $c \leftarrow P[i]$;
    First $\leftarrow C[c] + Occ(c, \text{First}-1) + 1$;
    Last $\leftarrow C[c] + Occ(c, \text{Last})$;
**if** (Last $<$ First) **then return** 0 **else return** Last $-$ First $+ 1$;

**Fig. 1.** Backward search algorithm to find and count the suffixes in $SA$ prefixed by $P$ (or the occurrences of $P$ in $T$).

## 2 Background and Notation

We assume that the symbols of $T$ are drawn from an alphabet $A = \{a_1, \ldots, a_\sigma\}$ of size $\sigma$. We will have different ways to express the size of a disk block: $b$ will be the number of symbols, $\bar{b}$ the number of bits, and $\tilde{b}$ the number of integers in a block.

***The suffix array*** $SA[1,n]$ of a text $T$ contains all the starting positions of the suffixes of $T$, such that $T_{SA[1]\ldots n} < T_{SA[2]\ldots n} < \ldots < T_{SA[n]\ldots n}$, that is, $SA$ gives the lexicographic order of all suffixes of $T$. All the occurrences of a pattern $P$ in $T$ are pointed from an interval of $SA$.

***The Burrows-Wheeler transform (BWT)*** is a reversible permutation $T^{bwt}$ of $T$ [3] which puts together characters sharing a similar context, so that $k$-th order compression can be easily achieved. There is a close relation between $T^{bwt}$ and $SA$: $T^{bwt}_i = T_{SA[i]-1}$. This is the key reason why one can search using $T^{bwt}$ instead of $SA$.

The inverse transformation is carried out via the so-called "$LF$ mapping", defined as follows:

- For $c \in A$, $C[c]$ is the total number of occurrences of symbols in $T$ (or $T^{bwt}$) which are alphabetically smaller than $c$.
- For $c \in A$, $Occ(c,q)$ is the number of occurrences of character $c$ in the prefix $T^{bwt}[1,q]$.
- $LF(i) = C[T^{bwt}[i]] + Occ(T^{bwt}[i], i)$, the "$LF$ mapping".

***Backward searching*** is a technique to find the area of $SA$ containing the occurrences of a pattern $P_{1,m}$ by traversing $P$ backwards and making use of the BWT. It was first proposed for the FM-index [8, 9], a self-index composed of a compressed representation of $T^{bwt}$ and auxiliary structures to compute $Occ(c,q)$. Fig. 1 gives the pseudocode to get the area $SA[\text{First}, \text{Last}]$ with the occurrences of $P$. It requires at most $2(m-1)$ calls to $Occ$. Depending on the variant, each call to $Occ$ can take constant time for small alphabets [8] or $O(\log \sigma)$ time in general [9], using wavelet trees (see below).

***A rank/select dictionary*** over a binary sequence $B_{1,n}$ is a data structure that supports functions $rank_c(B,i)$ and $select_c(B,i)$, where $rank_c(B,i)$ returns the number of times $c$ appears in prefix $B_{1,i}$ and $select_c(B,i)$ returns the position of the $i$-th appearance of $c$ within sequence $B$.

Both *rank* and *select* can be computed in constant time using $o(n)$ bits of space in addition to $B$ [20, 10], or $nH_0(B)+o(n)$ bits [23]. In both cases the $o(n)$ term is $\Theta(n \log\log n/\log n)$.

Let $s$ be the number of one bits in $B_{1,n}$. Then $nH_0(B) \approx s \log \frac{n}{s}$, and thus the $o(n)$ terms above are too large if $s$ is not close to $n$. Existing lower bounds [19] show that constant-time *rank* can only be achieved with $\Omega(n \log\log n/\log n)$ extra bits. As in this paper we will have $s << n$, we are interested in techniques with less overhead over the entropy, even if not of constant-time (this will not be an issue for us). One such rank dictionary [14] encodes the gaps between successive 1's in $B$ using $\delta$-encoding and adds some data to support a binary-search-based *rank*. It requires $s(\log \frac{n}{s} + \frac{\log n}{\log s} + 2\log\log \frac{n}{s}) + O(\log n)$ bits of space and supports *rank* in $O(\log s)$ time. We call this structure $GR$.

***The wavelet tree*** [13] $wt(S)$ over a sequence $S[1,n]$ is a perfect binary tree of height $\lceil \log \sigma \rceil$, built on the alphabet symbols, such that the root represents the whole alphabet and each leaf represents a distinct alphabet symbol. If a node $v$ represents alphabet symbols in the range $A^v = [i,j]$, then its left child $v_l$ represents $A^{v_l} = [i, \frac{i+j}{2}]$ and its right child $v_r$ represents $A^{v_r} = [\frac{i+j}{2}+1, j]$. We associate to each node $v$ the subsequence $S^v$ of $S$ formed by the characters in $A^v$. However, sequence $S^v$ is not really stored at the node. Instead, we store a bit sequence $B^v$ telling whether characters in $S^v$ go left or right, that is, $B^v_i = 1$ if $S^v_i \in A^{v_r}$. The wavelet tree has all its levels full, except for the last one that is filled left to right.

In this paper $S$ will be $T^{bwt}$. A plain wavelet tree of $S$ requires $n \log \sigma$ bits of space. If we compress the wavelet tree using a numbering scheme [23] we obtain $nH_k(T) + o(n \log \sigma)$ bits of space for any $k \leq \alpha \log_\sigma n$ and any $0 < \alpha < 1$ [16].

The wavelet tree permits us to calculate $Occ(c,i)$ using binary ranks over the bit sequences $B^v$. Starting from the root $v$ of the wavelet tree, if $c$ belongs to the right side, we set $i \leftarrow rank_1(i)$ over vector $B^v$ and move to the right child of $v$. Similarly, if $c$ belongs to the left child we update $i \leftarrow rank_0(i)$ and go to the left child. We repeat this until reaching the leaf that represents $c$, where the current $i$ value is the answer to $Occ(c,i)$.

***The locally compressed suffix array (LCSA)*** [12], is built on well-known regularity properties that show up in suffix arrays when the text they index is compressible [22]. The LCSA uses differential encoding on $SA$, which converts those regularities into true repetitions. Those repetitions are then factored out using Re-Pair [15], a compression technique that builds a dictionary of phrases and permits fast local decompression using only the dictionary (whose size one can control at will, at the expense of losing some compression). Also, the Re-Pair dictionary is further compressed with a novel technique. The LCSA can extract any portion of the suffix array very fast by adding a small set of sampled absolute values. It is proved in [12] that the size of the LCSA is $O(H_k \log(1/H_k) n \log n)$ bits for any $k \leq \alpha \log_\sigma n$ and any constant $0 < \alpha < 1$.

The LCSA consists in three substructures: the sequence of phrases $SP$, the compressed dictionary $CD$ needed to uncompress the phrases and the absolute sample values to restore the suffix array values. One disadvantage of the original structure is the space and time needed to construct it. In [5] they present a

heuristic to overcome this, as it can run with limited main memory and performs sequential passes on disk. It might not choose the pairs to replace as well as the original algorithm, but it can trade construction time for precision.

## 3 A Compressed Secondary Memory Structure

We present a structure on secondary memory, which is able to answer count, locate and extract queries. It is composed of three substructures, each one responsible for one type of query, and allowing diverse trade-offs depending on how much main memory space they occupy.

### 3.1 Entropy-compressed rank dictionary on secondary memory

As we will require several bitmaps in our structure with few bits set, we describe an entropy-compressed rank dictionary, suitable for secondary memory, to represent a binary sequence $B_{1,n}$. In case it fits in main memory, we use $GR$ (Section 2). Otherwise we will use $DEB$, the $\delta$-encoded form of $B$: we encode the gaps between consecutive 1's in $B$ as variable-length integers, so that $x$ is represented using $\log x + 2\log\log x$ bits. $DEB$ uses at most $s\log\frac{n}{s} + 2s\log\log\frac{n}{s} + O(\log n)$ bits of space [16, Sec. 3.4.1]. Let $\bar{b}$ be the number of bits contained in a disk block. We split $DEB$ into blocks of at most $\bar{b}$ bits: if a $\delta$-encoding spans two blocks we move it to the next block. Each block is stored in secondary memory and, at the beginning of block $i$, we also store the number of 1's accumulated up to block $i-1$; we call this value $OB_i$. To access $DEB$, we use in main memory an array $B^a$, where $B^a[i]$ is the number of bits of $B$ represented in blocks 1 to $i-1$. $B^a$ uses $(s\log\frac{n}{s} + 2s\log\log\frac{n}{s} + O(\log n))\frac{\log n}{\bar{b}}$ bits of space.

To answer $rank_1(B,i)$ with this structure, we carry out the following steps: (1) We binary search $B^a$ to find $j$ such that $B^a[j] \le i < B^a[j+1]$. (2) We read block $j$ from disk. (3) We decompress the $\delta$-encodings in block $j$ until reaching position $i$, summing up the bits set. (4) $rank_1(B,i)$ will be the previous sum plus $OB_i$, the accumulator of 1's stored in the block.

Overall this costs $O(\log\frac{s}{\bar{b}} + \log\log\frac{n}{s} + \bar{b}) = O(\log s + \log\log n + \bar{b})$ CPU time and just one disk access. When we use these structures in the paper, $s$ will be $\Theta(n/b)$. Table 1 shows some real sizes and times obtained for the structures, when $s = n/b$. As it can be seen, we require very little main memory for the second scheme, and for moderate-size bitmaps even the $GR$ option is good.

### 3.2 Counting

We run the algorithm of Fig. 1 to answer a counting query. Table $C$ uses $\sigma\log n$ bits and easily fits in main memory, thus the problem is how to calculate $Occ$ over $T^{bwt}$.

We describe four different structures to count depending on how we represent $T^{bwt}$. We enumerate the versions from 1 to 4. In versions 1 and 2, we use an uncompressed form of $T^{bwt}$ and pay one I/O per call to $Occ$. In versions 3 and

**Table 1.** Different sizes and times obtained to answer *rank*, for some relevant choices of $n$ and $b$. $DEB$ is stored in secondary memory and is accessed using $B^a$. $B^a$ and $GR$ reside in main memory. Tb, Gb, etc. mean terabits, gigabits, etc. TB, GB, etc. mean terabytes, gigabytes, etc.

| Structure | Space (bits) | CPU time for *rank* | Real space if $s = n/b$ | | | |
|---|---|---|---|---|---|---|
| | | | $n =$ 1 Tb $b =$ 32 KB | 1 Gb 8 KB | 1 Gb 4 KB | 1 Mb 4 KB |
| $GR$ | $s \log \frac{n}{s} + s\frac{\log n}{\log s} + 2s \log\log \frac{n}{s} + O(\log n)$ | $O(\log s)$ | 100 MB | 354 KB | 667 KB | 677 B |
| $DEB+$ $B^a$ | $s \log \frac{n}{s} + 2s \log\log \frac{n}{s} + O(\log n) +$ $(s \log \frac{n}{s} + 2s \log\log \frac{n}{s} + O(\log n))\frac{\log n}{b}$ | $O(\log s + b$ $+ \log\log n)$ | 93 MB 14 KB | 326 KB 153 B | 613 KB 575 B | 600 B 1 B |

4, we use a compressed form of $T^{bwt}$ and pay one or two I/Os per call to *Occ*. In versions 1 and 3, we spend $O(b)$ CPU operations per call to *Occ*. In versions 2 and 4, this is reduced to $O(\log \sigma)$. Version 4 is omitted from now on as it is not competitive.

To calculate $Occ(c, i)$, we need to know the number of occurrences of symbol $c$ before each block on disk. To do so, we store a two-level structure: the first level stores for every $t$-th block the number of occurrences of every $c$ from the beginning, and the second level stores the number of occurrences of every $c$ from the last $t$-th block. The first level is maintained in main memory and the second level on disk, together with the representation of $T^{bwt}$ (i.e., the entry of each block is stored within the block). Let $K$ be the total number of blocks. We define:

- $E_c(j)$: number of occurrences of symbol $c$ in blocks 0 to $(j-1)\cdot t$, with $E_c(0) = 0$, $1 \leq j < \lfloor K/t \rfloor$.
- $E'_c(j)$: $j$ goes from 0 to $K-1$. For $j \bmod t = 0$ we have $E'_c(j) = 0$, and for the rest we have that $E'_c(j)$ is the number of occurrences of symbol $c$ in blocks from $\lfloor j/t \rfloor \cdot t$ to $j-1$.

Summing up all the entries, $E$ uses $\lceil K/t \rceil \cdot \sigma \log n$ bits and $E'$ uses $K\sigma \log \frac{t \cdot n}{K}$ bits of space in versions 1 and 2. In version 3, the numbering scheme [23] has a compression limit $n/K \leq b \cdot \log n/(2 \log\log n)$. Thus, for version 3, $E'$ uses at most $K \cdot \sigma \log(t \cdot \frac{b \log n}{2 \log\log n})$ bits.

To use these structures, we first need to know in which block lies $T^{bwt}[i]$:

- In versions 1 and 2, where the block size is constant, we know directly that $T^{bwt}[i]$ belongs to block $\lfloor i/b \rfloor$, where $b$ is the number of symbols that fit in a disk block.
- In version 3, the block size is variable. Compression ensures that there are at most $n/b$ blocks. We use a binary sequence $EB_{1,n}$ to mark where each block starts. Thus the block of $T^{bwt}[i]$ is $rank_1(EB, i)$. We use an entropy-compressed rank dictionary (Section 2) for $EB$. If we need to use the $DEB$ variant, we add up one more I/O per access to $T^{bwt}$ (Section 3.1) .

With this sorted out, we can compute $Occ(c, i) = E_c(j \text{ div } t) + E'_c(j) + Occ'(B_j, c, \mathit{offset})$, where $j$ is the block where $i$ belongs, *offset* is the position of $i$ within block $j$, and $Occ'(B_j, c, \mathit{offset})$ is the number of occurrences of symbol $c$

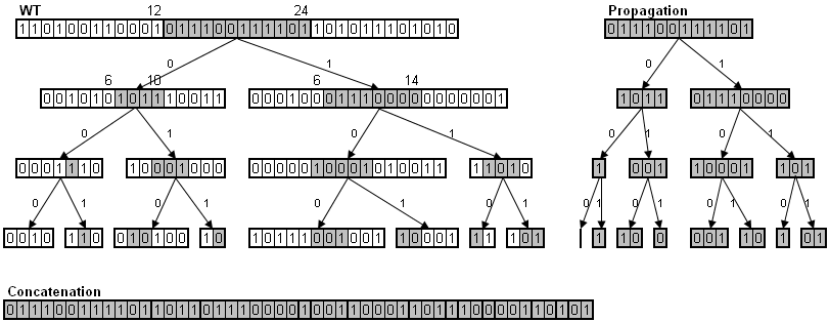



**Fig. 2.** Block propagation over the wavelet tree. Making ranks over the first level of $WT$ ($rank_0(12) = 6$, $rank_0(24) = 10$ and $rank_1(i) = i - rank_0(i)$), we determine propagation over the second level of $WT$, and so on.

within block $B_j$ up to *offset*. Now we explain the three ways to represent $T^{bwt}$, and this will give us three different ways to calculate $Occ'(B_j, c, \mathit{offset})$.

***Version 1.*** We use directly $T^{bwt}$ without any compression. If a disk block can store $b$ symbols (ie, $b \log \sigma$ bits), we will have $K = \lceil n/b \rceil$ blocks. $Occ'(B_j, c, \mathit{offset})$ is calculated by traversing the block and counting the occurrences of $c$ up to *offset*.

***Version 2.*** Let $b$ be the number of symbols within a disk block. We divide the first level of $WT = wt(T^{bwt})$ into blocks of $b$ bits. Then, for each block, we gather its propagation over $WT$ by concatenating the subsequences in breadth-first order, thus forming a sequence of $b \log \sigma$ bits. See Fig. 2. Note that this propagation generates $2^{j-1}$ intervals at level $j$ of $WT$. Some definitions:

- $B_i^j$: the $i$-th interval of level $j$, with $1 \leq j \leq \lceil \log \sigma \rceil$ and $1 \leq i \leq 2^{j-1}$.
- $L_i^j$: the length of interval $B_i^j$.
- $O_i^j / Z_i^j$: the number of 1's/0's in interval $B_i^j$.
- $D_j = B_1^j \dots B_{2^{j-1}}^j$ with $1 \leq j \leq \lceil \log \sigma \rceil$: all concatenated intervals from level $j$.
- $B = D_1 D_2 \dots D_{\lceil \log \sigma \rceil}$: concatenation of all the $D_j$, with $1 \leq j \leq \lceil \log \sigma \rceil$.

Some relationships hold: (1) $L_i^j = O_i^j + Z_i^j$. (2) $Z_i^j = rank_0(B_i^j, L_i^j)$. (3) $L_i^j = Z_{(i+1)/2}^{j-1}$ if $i$ is odd ($B_i^j$ is a left child); $L_i^j = O_{i/2}^{j-1}$ otherwise. (4) $|D_j| = L_1^1 = b$ for $j < \lfloor \log \sigma \rfloor$, the last level can be different if $\sigma$ is not a power of 2. With those properties, $L_i^j$, $O_i^j$ and $Z_i^j$ are determined recursively from $B$ and $b$. We only store $B$ plus the structures to answer $rank_1$ on it in constant time [10]. Note that any $rank_1(B_i^j)$ is answered via two ranks over $B$.

Fig. 3 shows how we calculate $Occ'$ in $O(\log \sigma)$ constant-time steps. To navigate the wavelet tree, we use some properties:

1. Block $D_j$ begins at bit $(j-1) \cdot b + 1$ of $B$, and $|B| = b \log \sigma$.

**Algorithm** $Occ'(B, c, j)$
$node \leftarrow 1; ans \leftarrow j; des \leftarrow 0; B_1^1 = B[1, b];$
**for** $level \leftarrow 1$ **to** $\lceil \log \sigma \rceil$ **do**
  **if** $c$ belongs to the left subtree of $node$ **then**
    $ans \leftarrow rank_0(B_{node}^{level}, ans);$
    $len \leftarrow Z_{node}^{level};$
    $node \leftarrow 2 \cdot node - 1;$
  **else** $ans \leftarrow rank_1(B_{node}^{level}, ans);$
    $len \leftarrow O_{node}^{level}; \quad des \leftarrow Z_{node}^{level};$
    $node \leftarrow 2 \cdot node;$
  $B_{node}^{level} = B[level \cdot b + des + 1, level \cdot b + des + len];$
**return** $ans$;

**Fig. 3.** Algorithm to obtain the number of occurrences of $c$ inside a disk block, for version 2.

**Table 2.** Different sizes and times obtained to answer $count(P_{1,m})$.

| Version | Main Memory | Secondary Memory | I/O | CPU |
|---|---|---|---|---|
| 1 | $O(\frac{n}{bt} \cdot \sigma \log n)$ | $n \log \sigma + O(\frac{n}{b} \cdot \sigma \log(t \cdot b))$ | $2(m-1)$ | $O(m \cdot b)$ |
| 2 | $O(\frac{n}{bt} \cdot \sigma \log n)$ | $n \log \sigma + O(\frac{n}{b} \cdot \sigma \log(t \cdot b))$ | $2(m-1)$ | $O(m \log \sigma)$ |
| 3a | $O(\frac{n}{bt} \cdot \sigma \log n + \frac{n}{b} \log n)$ | $nH_k(T) + O(\sigma^{k+1} \log n)$ $+O(\frac{n}{b} \cdot \sigma \log(t \cdot b \log n))$ | $2(m-1)$ | $O(m(b + \log n))$ |
| 3b | $O(\frac{n}{bt} \cdot \sigma \log n + \frac{n}{b^2} \log n \log b)$ | $nH_k(T) + O(\sigma^{k+1} \log n)$ $+O(\frac{n}{b} \cdot \sigma \log(t \cdot b \log n))$ | $4(m-1)$ | $O(m(b + \log n))$ |

2. To know where $B_i^j$ begins, we only need to add to the beginning of $D_j$ the length of $B_1^j, \ldots, B_{i-1}^j$. Each $B_k^j$, with $1 \leq k \leq i-1$, belongs to a left branch that we do not follow to reach $B_i^j$ from the root. So, when we descend through the wavelet tree to $B_i^j$, every time we take a right branch we accumulate the number of bits of the left branch (zeroes of the parent).
3. $node$ is the number of the current interval at the current $level$.
4. We do not calculate $B_{node}^{level}$, we just maintain its position within $B$.

***Version 3.*** We compress block $B$ from version 2 using a numbering scheme [23], yet without any structure for $rank$. In this case the division of $T^{bwt}$ is not uniform, but we add symbols from $T^{bwt}$ to the disk block as long as its compressed $WT$ fits in the block. By doing this, we compress $T^{bwt}$ to $nH_k + O(\sigma^{k+1} \log n + n \log \log n / \log n)$ bits for any $k$ [16]. To calculate $Occ'(B, c, \mathit{offset})$, we decompress block $B$ and apply the same algorithm of version 2, in $O(b)$ time.

In Table 2 we can see the different sizes and times needed for our three versions. We added the times to do $rank$ on the entropy-compressed bit arrays.

### 3.3 Locating

Our locating structure will be a variant of the LCSA, see Section 2. The array $SP$ from LCSA will be split into disk blocks of $\tilde{b}$ integers. Also, we will store in each block the absolute value of the suffix array at the beginning of the block. For

optimization of I/O, the dictionary will be maintained in main memory. So we compress the differential suffix array until we reach the desired dictionary size. Finally, we need a compressed bitmap $LB$ to mark the beginning of each disk block. $LB$ is entropy-compressed and can reside in main or secondary memory.

For locating every match of a pattern $P_{1,m}$, first we use our counting substructure to obtain the interval [First, Last] of the suffix array of $T$ (see Section 2). Then we find the block First belongs to, $j = rank_1(LB, \text{First})$. Finally, we read the necessary blocks until we reach Last, uncompressing them using the dictionary of the LCSA.

We define $occ = \text{Last} - \text{First} + 1$ and $occ' = cr \cdot occ$, where $0 < cr \leq 1$ is the compression ratio of $SP$. This process takes, without counting, $\lceil occ'/\tilde{b} \rceil$ I/O accesses, plus one if we store $LB$ in secondary memory. This I/O cost is optimal and improves thanks to compression. We perform $O(occ + \tilde{b})$ CPU operations to uncompress the interval of $SP$.

### 3.4 Extracting

To extract arbitrary portions of the text we use a variant of [11], which compresses $T$ blockwise using a semistatic statistical modeler of order $k$ plus an encoder $EN$. This compresses the text to $nH_k(T) + f_{EN}(n)$, where $f_{EN}(n)$ is the redundancy of the encoder. For example, a Huffman coder has redundancy $n$, whereas an arithmetic encoder has redundancy 2. The data generated by the modeler, $DM$, is maintained in main memory, which requires $\sigma^{k+1} \log n$ bits. This restricts the maximum possible $k$ to be used: If we have $ME$ bits to store the data generated by the modeler then $k \leq \log_\sigma(ME/\log n) - 1$. To store the structure in secondary memory we split the compressed text into disk blocks of size $\bar{b}$ bits (thus the overhead over the entropy is $\frac{n}{\bar{b}} f_{EN}(\bar{b})$). If we store less than $b = \bar{b}/\log \sigma$ symbols in a particular disk block, we rather store it uncompressed. An extra bit per block indicates whether this was the case. Also each block will contain the context of order $k$ of the first symbol stored in the block ($k \log \sigma$ bits). To know where a symbol of $T$ is stored we need a compressed rank dictionary $ER$, in which we mark the beginning of each block. This can be chosen to be in main memory or in secondary memory, the latter requiring one more I/O access.

The algorithm to extract $T_{l,r}$ is: (1) Find the block $j = rank_1(ER, l)$ where $T_l$ is stored. (2) Read block $j$ and decompress it using $DM$ and the context of the $k$ first symbols. (3) Continue reading blocks and decompressing them until reaching $T_r$.

Using this scheme we have at most $\lceil (j - i + 1)/b \rceil$ I/O operations, which on average is $\lceil (j-i+1)H_k(T)/\bar{b} \rceil$. We add one I/O operation if we use the secondary memory version of the rank dictionary. The total CPU time is $O(j-i+b+\log n)$.

## 4 Experiments

We consider two text files for the experiments: the text WSJ (Wall Street Journal) from the TREC collection from year 1987, of 126 MB, and the 200 MB XML file

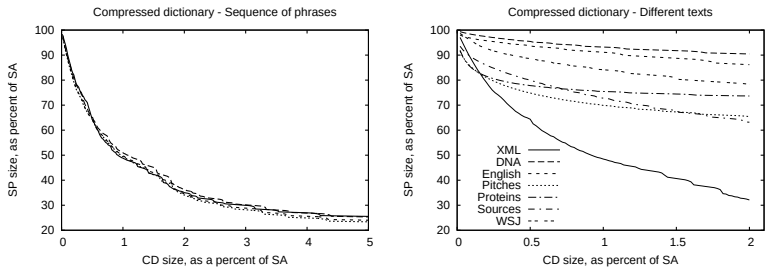



**Fig. 4.** Compression ratio achieved on XML as a function of the percentage allowed to the dictionary ($CD$). Both are percentage over the size of $SA$, the right plot shows other texts.

provided in the *Pizza&Chili Corpus* (`http://pizzachili.dcc.uchile.cl`). We searched for 5,000 random patterns, of length from 5 to 50, generated from these files. As in [6] and [1], we assume a disk page size of 32 KB.

We first study the compressibility we achieve as a function of the dictionary size, $|CD|$ (as $CD$ must reside in RAM). Fig. 4 shows that the compressibility depends on the percentage $|CD|/|SA|$ and not on the absolute size $|CD|$. In the following, we let our $CD$ use 2% of the suffix array size. For counting we use version 1 ($GR$, Section 3.2) with $t = \log n$. With this setting our index uses 19.15 MB of RAM for XML, and 12.54 MB for WSJ (for $GR$, $CD$, and $DM$). It compresses the $SA$ of XML to 34.30% and that of WSJ to 80.28% of its original size.

We compared our results against String B-tree [7], Compact Pat Tree (CPT) [4], disk-based Suffix Array (SA) [2] and disk-based LZ-Index [1]. We add our results to those of [1, Sec. 4]. We omit the disk-based CSA [17] as it is not implemented, but that one is strictly worse than ours.

Fig. 5 (left) shows counting experiments. Our structure needs at most $2(m-1)$ disk accesses. We show our index with and without the substructures for locating. Fig. 5 (right) shows locating experiments. For $m = 5$, we report *more* occurrences than those the block could store in raw format.

We can see that the result depends a lot on the compressibility of the text. In the highly-compressible XML our index occupies a very relevant niche in the tradeoff curves, whereas in WSJ it is subsumed by String B-trees. Thus, our index is very competitive on compressible texts. We have used texts up to 200 MB, but our results show that the outcome scales up linearly for the RAM needed, while the counting cost is at most $2(m-1)$, the locating cost depends on the number of occurrences of $P$. Thus it is very easy to predict other scenarios.

## References

1. D. Arroyuelo and G. Navarro. A Lempel-Ziv text index on secondary storage. In *Proc. CPM*, LNCS 4580, pages 83–94, 2007.
2. R. Baeza-Yates, E. F. Barbosa, and N. Ziviani. Hierarchies of indices for text searching. *Inf. Systems*, 21(6):497–514, 1996.

3. M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Tech.Rep. 124, DEC, 1994.
4. D. Clark and I. Munro. Efficient suffix trees on secondary storage. In *Proc. SODA*, pages 383–391, 1996.
5. F. Claude and G. Navarro. A fast and compact web graph representation. In *Proc. SPIRE*, pages 105–116, 2007. LNCS 4726.
6. P. Ferragina and R. Grossi. Fast string searching in secondary storage: theoretical developments and experimental results. In *Proc. SODA*, pages 373–382, 1996.
7. P. Ferragina and R. Grossi. The string B-tree: A new data structure for string search in external memory and its applications. *J. ACM*, 46(2):236–280, 1999.
8. P. Ferragina and G. Manzini. Indexing compressed texts. *J. ACM*, 52(4):552–581, 2005.
9. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM TALG*, 3(2):article 20, 2007.
10. R. González, Sz. Grabowski, V. Mäkinen, and G. Navarro. Practical implementation of rank and select queries. In *Proc. Posters WEA*, pages 27–38, Greece, 2005. CTI Press and Ellinika Grammata.
11. R. González and G. Navarro. Statistical encoding of succinct data structures. In *Proc. CPM*, LNCS 4009, pages 295–306, 2006.
12. R. González and G. Navarro. Compressed text indexes with fast locate. In *Proc. CPM*, LNCS 4580, pages 216–227, 2007.
13. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. SODA*, pages 841–850, 2003.
14. A. Gupta, W.-K. Hon, R. Shah, and J. Vitter. Compressed data structures: dictionaries and data-aware measures. In *Proc WEA*, pages 158–169, 2006.
15. J. Larsson and A. Moffat. Off-line dictionary-based compression. *Proc. IEEE*, 88(11):1722–1732, 2000.
16. V. Mäkinen and G. Navarro. Implicit compression boosting with applications to self-indexing. In *Proc. SPIRE*, pages 214–226, 2007. LNCS 4726.
17. V. Mäkinen, G. Navarro, and K. Sadakane. Advantages of backward searching — efficient secondary memory and distributed implementation of compressed suffix arrays. In *Proc. ISAAC*, LNCS 3341, pages 681–692, 2004.
18. G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.
19. P. Miltersen. Lower bounds on the size of selection and rank indexes. In *Proc. SODA*, pages 11–12, 2005.
20. I. Munro. Tables. In *Proc. FSTTCS*, LNCS 1180, pages 37–42, 1996.
21. G. Navarro. Indexing text using the Ziv-Lempel trie. *J. Discrete Algorithms*, 2(1):87–114, 2004.
22. G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.
23. R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding $k$-ary trees and multisets. In *Proc. SODA*, pages 233–242, 2002.
24. K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *J. Algor.*, 48(2):294–313, 2003.
25. N. Ziviani, E. Moura, G. Navarro, and R. Baeza-Yates. Compression: A key for next-generation text retrieval systems. *IEEE Computer*, 33(11):37–44, 2000.

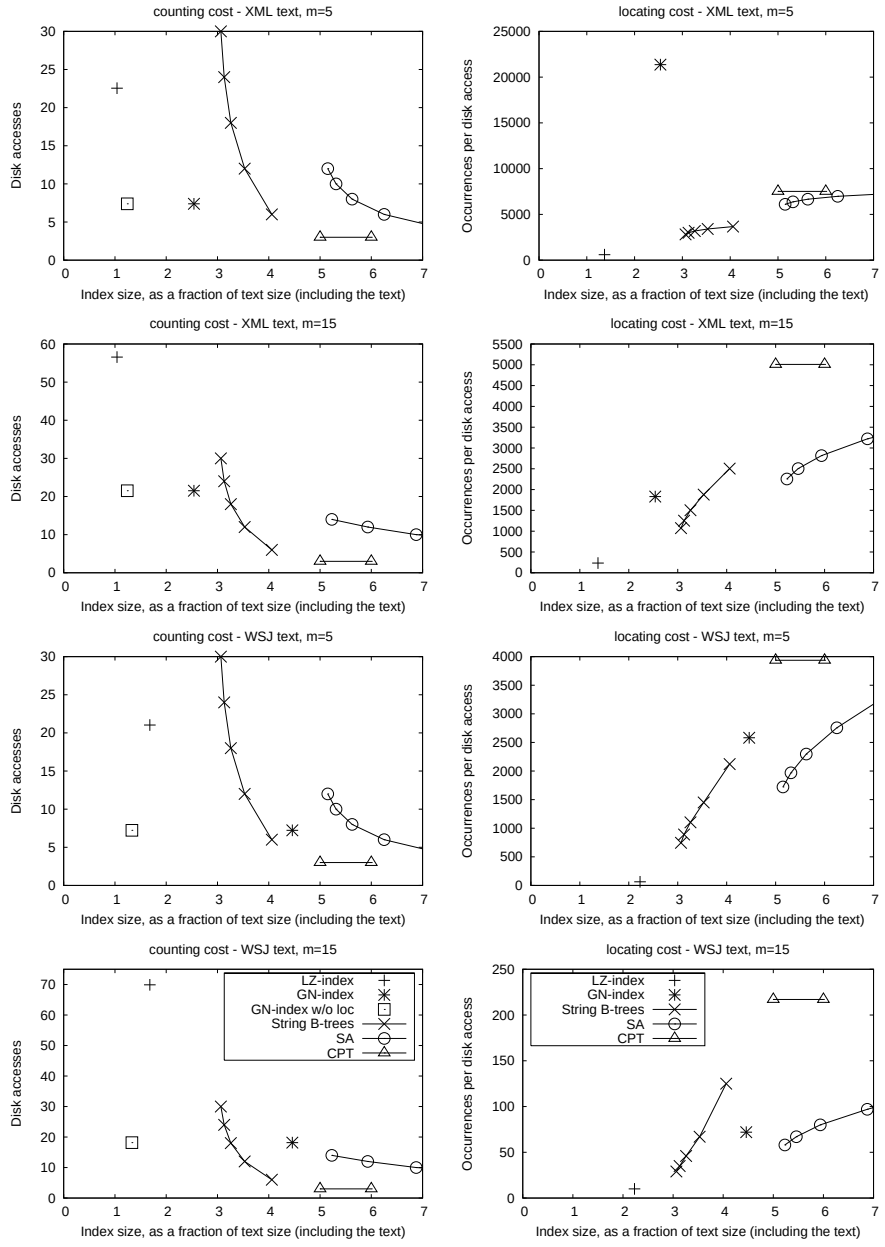



**Fig. 5.** Search cost vs. space requirement for the different indexes we tested. Counting on the left and locating on the right.

# Star-shaped Drawings of Planar Graphs

⋆

Seok-Hee Hong[1] and Hiroshi Nagamochi[2]

[1] School of Information Technologies, University of Sydney
shhong@it.usyd.edu.au
[2] Department of Applied Mathematics and Physics,
Kyoto University,
nag@amp.i.kyoto-u.ac.jp

**Abstract.** A star-shaped drawing of a plane graph is a straight-line drawing such that each inner facial cycle is drawn as a star-shaped polygon and the outer facial cycle is drawn as a convex polygon. Given a biconnected *planar* graph, we consider the problem of finding a star-shaped drawing of the graph with the minimum number of concave corners. We derive an effective lower bound on the number of concave corners, and prove that the problem can be solved in linear time.

## 1 Introduction

Graph drawing has attracted much attention over the last ten years due to its wide range of applications, such as VLSI design, software engineering and bioinformatics. Two- or three-dimensional drawings of graphs with a variety of aesthetics and edge representations have been extensively studied (see [1]).

One of the most popular drawing conventions is the *straight-line drawing*, where all the edges of a graph are drawn as straight-line segments. Every planar graph is known to have a planar straight-line drawing [5]. A straight-line drawing is called a *convex drawing* if every facial cycle is drawn as a convex polygon. Note that not all planar graphs admit a convex drawing.

Tutte [14] gave a necessary and sufficient condition for a plane graph to admit a convex drawing. He also showed that every *triconnected* plane graph with a given boundary drawn as a convex polygon admits a convex drawing using the polygonal boundary. Later, Thomassen [13] gave a necessary and sufficient condition for a *biconnected* plane graph to admit a convex drawing. Based on this result, Chiba et al. [4] presented a linear time algorithm for finding a convex drawing (if any) for a biconnected plane graph with a specified convex boundary.

In general, the convex drawing problem has been well investigated for the last ten years. Recently Hong and Nagamochi gave conditions for hierarchical plane

⋆ This is an extended abstract. This research was done when the second author was visiting NICTA (National ICT Australia) and partially supported by the Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

graphs to admit a convex drawing [7], and for c-planar clustered graphs to admit a convex drawing in which every cluster is also drawn as a convex polygon [8]. Another variation of convex drawing with minimum outer apices was studied by Miura et al. [12]. They gave a linear time algorithm for finding a convex drawing with minimum outer apices for an internally triconnected plane graph.

However, not much attention has been paid to the problem of finding a convex drawing with a *non-convex boundary* or *non-convex faces*. Recently, in our companion paper [6], we proved that every triconnected plane graph whose boundary is fixed with a star-shaped polygon has an inner-convex drawing (a drawing in which every inner face is drawn as a convex polygon), if its kernel has a positive area. Note that this is an extension of the classical result by Tutte [14], since any convex polygon is a star-shaped polygon.

In this paper, we deal with biconnected planar graph, and consider how to draw these graphs nicely. However, Kant [11] already proved that the problem of deciding whether a biconnected planar graph can be drawn with at most $k$ nonconvex faces is NP-complete. We define a "star-shaped drawing" of a plane graph to be a straight-line drawing such that each inner facial cycle is drawn as a star-shaped polygon and the outer facial cycle is drawn as a convex polygon. We consider the problem of finding a star-shaped drawing of a biconnected planar graph with the minimum number of concave corners (including outer apices as concave corners). We first derive an effective lower bound on the number of concave corners by identifying two characteristic structures "multipaths" and "bi-facial cycles" that require concave corners in any straight-line drawing. Based on this, we design linear time algorithms for finding an optimal plane embedding $F$ of $G$ and for computing a star-shaped drawing of the embedding $F$.

**Theorem 1.** *Let $G$ be a biconnected planar graph. A star-shaped drawing with the minimum number of concave corners among all straight-line drawings of $G$ can be obtained in linear time.* □

## 2 Preliminaries

graph. Let $G = (V, E)$ be a graph. The set of edges incident to a vertex $v \in V$ is denoted by $E(v)$. The degree of a vertex $v$ in $G$ is denoted by $d_G(v)$ (i.e., $d_G(v) = |E(v)|$). For a subset $X \subseteq E$ (resp., $X \subseteq V$), $G - X$ denotes the graph obtained from $G$ by removing the edges in $X$ (resp., the vertices in $X$ together with the edges in $\cup_{v \in X} E(v)$).

A graph $G = (V, E)$ is called *planar* if its vertices and edges are drawn as points and curves in the plane so that no two curves intersect except for their endpoints, where no two vertices are drawn at the same point. In such a drawing, the plane is divided into several connected regions, each of which is called a *face*. A face is characterized by the cycle of $G$ that surrounds the region. Such a cycle is called a *facial cycle*, A set $F$ of facial cycles in a drawing is called a *plane embedding* of a planar graph $G$, where a face is specified as the outer face in a plane embedding, and is denoted by $f_F^o$. Let $\mathcal{F}(G)$ denote the set of all plane

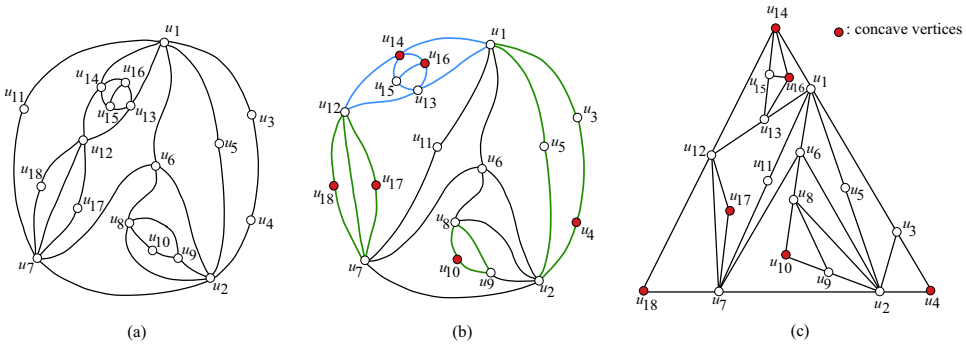



**Fig. 1.** Example of a biconnected planar graph $G = (V, E)$; (a) A plane embedding $F_1 \in \mathcal{F}(\{u_1, u_2, u_7\}; G)$; (b) A plane embedding $F_2 \in \mathcal{F}(\{u_1, u_2, u_7\}; G)$; (c) A star-shaped drawing of embedding $F_2 \in \mathcal{F}(\{u_1, u_2, u_7\}; G)$ with six concave corners.

embeddings of $G$. Figure 1(a) and (b) show two different plane embeddings of the same planar graph.

A planar graph $G = (V, E)$ with a fixed embedding $F$ of $G$ is called a *plane* graph. The set of vertices, set of edges and set of facial cycles of a plane graph $H$ may be denoted by $V(H)$, $E(H)$ and $F(H)$, respectively. A vertex (resp., an edge) in the outer facial cycle is called an *outer vertex* (resp., an *outer edge*), while a vertex (resp., an edge) not in the outer facial cycle is called an *inner vertex* (resp., an *inner edge*). For a subset $W \subseteq V$, let $\mathcal{F}(W; G)$ denote the set of all plane embeddings $F$ of $G$ such that $W \subseteq V(f_F^o)$.

A biconnected plane graph $G$ is called *internally triconnected* if, for any cut-pair $\{u, v\}$, $u$ and $v$ are outer vertices and each component in $G - \{u, v\}$ contains an outer vertex.

For two points $p_1, p_2$ in the plane, $[p_1, p_2]$ denotes the line segment with end points $p_1$ and $p_2$, and for three points $p_1, p_2, p_3$, $[p_1, p_2, p_3]$ denotes the triangle with three corners $p_1, p_2, p_3$. For a polygon $P$, let $V(P)$ denote the set of vertices of $P$. A *kernel* $K(P)$ of a polygon $P$ is the set of all points from which all points in $P$ are visible. The boundary of a kernel, if any, is a convex polygon. A polygon $P$ is called *star-shaped* if $K(P) \neq \emptyset$.

A *straight-line drawing* of a graph $G = (V, E)$ in the plane is an embedding of $G$ in the two dimensional space $\Re^2$, such that each vertex $v \in V$ is drawn as a point $\psi(v) \in \Re^2$, and each edge $(u, v) \in E$ is drawn as a straight-line segment $(\psi(u), \psi(v))$, where $\Re$ is the set of reals. A straight-line drawing $F$ of a plane graph $G = (V, E, F)$ is called a *convex drawing*, if every facial cycle is drawn as a convex polygon. We say that a drawing $F$ of a graph $G$ is *extended* from a drawing $\psi'$ of a subgraph $G'$ of $G$, if $\psi(v) = \psi'(v)$ for all $v \in V(G')$. A convex polygon drawn for the outer facial cycle in a biconnected plane graph can be extended to a convex drawing when the next condition holds.

**Theorem 2.** [4, 13] *Let $G = (V, E, F)$ be a biconnected plane graph. Then a drawing of $f_F^o$ on a convex polygon $P$ can be extended to a convex drawing of $G$ if and only if the following conditions* (i)-(iii) *hold:*

(i) *For each inner vertex $v$ with $d_G(v) \geq 3$, there exist three paths disjoint except for $v$, each connecting $v$ and an outer vertex;*

(ii) *Every cycle of $G$ which has no outer edge has at least three vertices $v$ with $d_G(v) \geq 3$; and*

(iii) *Let $Q_1, Q_2, \ldots, Q_k$ be the subpaths of $f_F^o$, each corresponding to a side of $P$. The graph $G - V(f_F^o)$ has no component $H$ such that all the outer vertices adjacent to vertices in $H$ are contained in a single path $Q_i$, and there is no inner edge $(u, v)$ whose end vertices are contained in a single path $Q_i$.* □

In a straight-line drawing of a planar graph $G$, the whole angle around a vertex $v$ is split into $d_G(v)$ angles, each of which is formed by two consecutive edges in the drawing and is called a *corner*. A corner is called *concave* if its angle is greater than $\pi$. A vertex $v$ in a straight-line drawing is called *concave* if one of the corners around $v$ is concave. For a straight-line drawing $D$ of a biconnected plane graph $G$, let $\Lambda(D)$ denote the set of all concave vertices in $D$.

A *star-shaped drawing* of a plane graph is a straight-line drawing such that each inner facial cycle is drawn as a star-shaped polygon and the outer facial cycle is drawn as a convex polygon. An outer vertex in a straight-line drawing of a plane graph is called an *apex* if it is concave in the drawing and its concave corner appears in the outer face. Figure 1(c) shows an example of a straight-line drawing $D$ of the plane graph in Figure 1(b), where $\Lambda(D) = \{u_4, u_{10}, u_{14}, u_{16}, u_{17}, u_{18}\}$ and $u_4$, $u_{14}$, and $u_{18}$ are the apices.

We easily observe the following.

**Lemma 1.** *Any straight-line drawing of a biconnected graph with at least three vertices requires at least three apices on its boundary.* □

We call a graph *trivial* if $G$ is a triconnected planar graph or a cycle. By Theorem 2, a trivial graph admits a convex drawing with a specified convex boundary. If we specify the boundary as a triangle, it has a convex drawing with three concave corners (three apices), and this is an optimal solution to our problem. In the following sections, we deal with nontrivial biconnected planar graphs.

The SPQR tree of a biconnected graph $G = (V, E)$ represents the adjacency the triconnected components of $G$ (see [2, 3] for detail). Each node $\nu$ in the SPQR tree is associated with a graph $\sigma(\nu) = (V_\nu, E_\nu)$ ($V_\nu \subseteq V$), called the *skeleton* of $\nu$, which corresponds to a triconnected component of $G$. Figure 2 shows the SPQR tree of the biconnected planar graph in Figure 1.

We treat the SPQR tree of a graph $G$ as a rooted tree $\mathcal{T}_{\nu^*}$ by choosing an arbitrary node $\nu^*$ as its root. The parent virtual edge of a node $\nu$ is denoted by $parent(\nu)$ (we let $parent(\nu) = \emptyset$ if $\nu$ is the root). We define a *parent cut-pair* of $\nu$ as the two endpoints of a parent virtual edge $e$. We denote the graph formed from $\sigma(\nu)$ by deleting its parent virtual edge as $\sigma^-(\nu) = (V_\nu, E_\nu^-)$, where $E_\nu^- = E_\nu - \{parent(\nu)\}$. Let $G^-(\nu)$ denote the subgraph of $G$ which consists

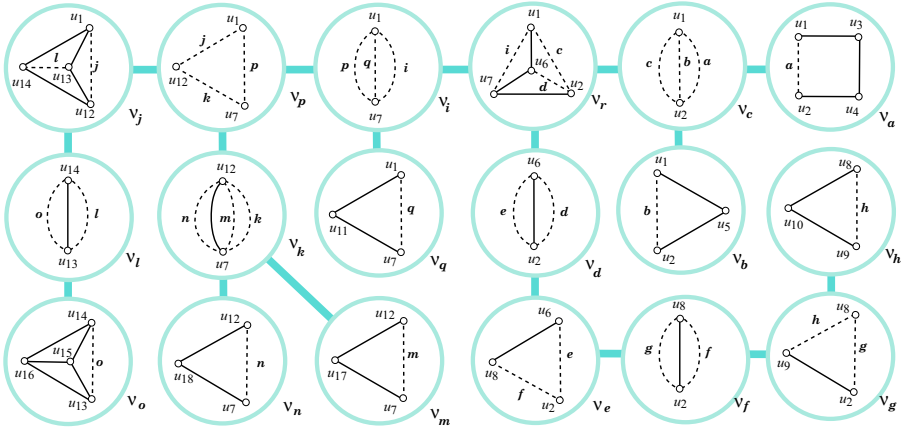


**Fig. 2.** The SPQR tree of the biconnected planar graph $G$ in Fig. 1.

of the vertices and real edges in the graphs $\sigma^-(\mu)$ for all descendants $\mu$ of $\nu$, including $\nu$ itself.

For a cut-pair $\{u, v\}$ of a biconnected graph $G$, a *$u, v$-component* $H$ is a connected subgraph of $G$ that either consists of a single edge $(u, v)$ or is a maximal subgraph such that $H - \{u, v\}$ remains connected. We may treat a $u, v$-component $H$ of a plane graph $G$ as a plane graph under the same embedding of $G$. In this case, we define the *$u, v$-boundary path* $f^o_{uv}(H)$ of $H$ to be the path obtained by traversing the boundary of $H$ from $u$ to $v$ in the clockwise order. A simple path with end vertices $u$ and $v$ of a graph $G$ is called a *$u, v$-path*, and is called an *induced $u, v$-path* if the every internal vertex (i.e., non end vertex) is of degree 2.

Let $A \subseteq V$ in a plane graph $G = (V, E, F)$. For a plane subgraph $H$ of $G$ and two outer vertices $u$ and $v$ of $H$, we denote $A_{uv}(H) = A \cap (V(Q_{uv}(H)) - \{u, v\})$. If $H$ is graph $G^-(\nu)$ for a node $\nu$ in the SPQR tree and $(u, v) = parent(\nu)$, then we denote $A_{uv}(H)$ by $A_{uv}(\nu)$ and $A_{uv}(\nu) \cup A_{vu}(\nu)$ by $A(\nu)$, where for the root $\nu^*$, we denote by $A(\nu^*)$ $A \cap V(f^o_F)$.

## 3 Structure of Multipaths and Bi-facial Cycles

This section identifies two structures "multipaths" and "bi-facial cycles" which cannot admit a straight-line drawing without introducing concave corners.

A set of at least two induced $u, v$-paths is a *multipath* (a path may be a single edge). We call vertices $u$ and $v$ the *terminals* of a multipath, and denote by $T(M)$ the set of terminals of a multipath $M$. A multipath between terminals $u$ and $v$ is called *maximal* if it contains all induced $u, v$-paths. For a multipath $M$, we denote by $|M|$ the number of paths in $M$. For example, the graph $G$ in Figure 1(a) has three maximal multipaths $M_1 = \{Q_1 = (u_1, u_5, u_2), Q_2 =$

$(u_1, u_3, u_4, u_2)\}$, $M_2 = \{Q_3 = (u_7, u_{12}), Q_4 = (u_7, u_{18}, u_{12}), Q_5 = (u_7, u_{17}, u_{12})\}$ and $M_3 = \{Q_6 = (u_8, u_9), Q_7 = (u_8, u_{10}, u_9)\}$. We here observe the following.

**Lemma 2.** *Assume that a nontrivial biconnected planar graph $G$ contains a multipath $M$. In any straight-line drawing of $G$, each of at least $|M|-1$ paths in $M$ must have a concave vertex among its internal vertices.* □

Then we see by definition that for any multipath, its terminals appear as the vertices in the skeleton $\sigma(\nu)$ of a P-node $\nu$ (recall that $G$ is not a cycle). Hence we observe the following.

**Lemma 3.** *Let $\mathcal{T}_{\nu^*}$ be the rooted SPQR-tree of a nontrivial biconnected planar graph $G$, and $f$ be a face of the skeleton $\sigma(\nu^*)$ of the root $\nu^*$ in $\mathcal{T}_{\nu^*}$. Then the set of all maximal multipaths is unique for all plane embeddings in $\mathcal{F}(V(f); G)$.* □

In a fixed plane embedding $F$ of a planar graph $G$, a cycle $C$ is called a *bi-facial cycle* if it is the boundary of a $u, v$-component $H$ of $G$ with $f_F^o \notin F(H)$ ($H$ is considered as a plane graph), where vertices $u$ and $v$ are called the *terminals* of the bi-facial cycle $C$. We denote by $T(C)$ the set of terminals of a bi-facial cycle $C$. For example, in the plane embedding $F_1$ in Figure 1(a), $C_1 = (u_{14}, u_{15}, u_{13}, u_{16})$, $C_2 = (u_1, u_{14}, u_{12}, u_{13})$, $C_3 = (u_8, u_2, u_9)$, and $C_4 = (u_6, u_8, u_2)$ are bi-facial cycles. We here observe the following.

**Lemma 4.** *Assume that a plane embedding $F$ of a nontrivial biconnected graph $G$ contains a bi-facial cycle $C$. In any straight-line drawing of $F$, at least one of non terminal vertices in $C$ is concave.* □

Note that a bi-facial cycle in a plane embedding of a graph $G$ may not be a bi-facial cycle in a different embedding of $G$. For example, cycle $C_4 = (u_6, u_8, u_2)$ is a bi-facial cycle in the plane embedding $F_1$ in Figure 1(a) but not in the plane embedding $F_2$ in Figure 1(b). A bi-facial cycle $C$ with $T(C) = \{u, v\}$ is called *minimal* if the corresponding $u, v$-component $H$ contains no other bi-facial cycle $C'$ which shares an edge with $C$ and no multipath $M$ sharing an edge with $C$. For our running example, $C_1$ and $C_2$ are minimal bi-facial cycles in both embeddings $F_1$ and $F_2$ in Figure 1(a) and (b), but bi-facial cycles $C_3$ and $C_4$ in embedding $F_1$ are not minimal, since $C_3$ and $C_4$ share an edge with bi-facial cycle $C_4$ and multipath $M_3$, respectively. It is not difficult to observe that The next property holds.

**Lemma 5.** *Let $\mathcal{T}_{\nu^*}$ be the rooted SPQR-tree of a nontrivial biconnected planar graph $G$, and $f$ be a face of $\sigma(\nu^*)$ of the root $\nu^*$ in $\mathcal{T}_{\nu^*}$. Let $C$ be a bi-facial cycle for some plane embedding $F \in \mathcal{F}(V(f); G)$. Then $C$ remains to be a bi-facial cycle for all plane embeddings in $\mathcal{F}(V(f); G)$ if and only if $C$ is a minimal bi-facial cycle in the plane embedding $F$.* □

By definition and Lemmas 3 and 5, we have the following.

**Lemma 6.** *Let $\mathcal{T}_{\nu^*}$ be the rooted SPQR-tree of a nontrivial biconnected planar graph $G$, $f$ be a face of $\sigma(\nu^*)$ of the root $\nu^*$ in $\mathcal{T}_{\nu^*}$, and $F \in \mathcal{F}(V(f); G)$. Let $L = \{C_1, \ldots, C_p, M_1, \ldots, M_q\}$ be the set of minimal bi-facial cycles $C_i$ and maximal multipaths $M_j$ such that $(V(H) - T(H)) \cap V(f) = \emptyset$ for all $H \in L$. Then:*

(i) *For every two subgraphs $H, H' \in L$, $H$ and $H'$ are edge-disjoint, and $(V(H) - T(H)) \cap (V(H') - T(H')) = \emptyset$.*
(ii) *$L$ is unique for all plane embeddings in $\mathcal{F}(V(f); G)$.* □

For a face $f$ in $\sigma(\nu^*)$ of the root $\nu^*$, we denote by $L(f; G)$ the set $L$ of minimal bi-facial cycles and maximal multipaths such that $(V(H) - T(H)) \cap V(f) = \emptyset$ for all $H \in L$. In our running example, for root $\nu^* = \nu_r$ and face $f = (u_1, u_2, u_7) \in F(\sigma(\nu_r))$ in the SPQR tree in Figure 2, we have $L(f; G) = \{C_1, C_2, M_1, M_2, M_3\}$.

We are now ready to derive a lower bound on the number of concave corners in a straight-line drawing of a plane embedding $F \in \mathcal{F}(V(f); G)$. Denote $L(f; G) = \{C_1, \ldots, C_p, M_1, \ldots, M_q\}$ as in Lemma 6, and define

$$a(f) := p + (|M_1| - 1) + \cdots + (|M_q| - 1). \tag{1}$$

Note that any straight-line drawing of $F \in \mathcal{F}(V(f); G)$ must have at least $a(f)$ concave corners on those subgraphs in $L(f; G)$. Fix an embedding $F \in \mathcal{F}(V(f); G)$, and denote by $b(F; f)$ the number of concave corners that appear on the boundary of $F$ among such concave corners. More specifically, $b(F; f)$ is given as follows. If the boundary of $F$ consists of two induced paths $Q, Q' \in M_j$ for some $M_j \in L(f; G)$, then we define

$$b(F; f) := \begin{cases} 2 \ (|M_j| \geq 3), \\ 1 \ (|M_j| = 2); \end{cases} \tag{2}$$

otherwise we define

$$b(F; f) := |\{H \in L(f; G) \mid (V(H) - T(H)) \cap V(f_F^o) \neq \emptyset\}|. \tag{3}$$

Hence we have the following.

**Lemma 7. (lower bound)** *Let $\mathcal{T}_{\nu^*}$ be the rooted SPQR-tree of a nontrivial biconnected planar graph $G$, and $f$ be a face of $\sigma(\nu^*)$ of the root $\nu^*$ in $\mathcal{T}_{\nu^*}$. Then for any straight-line drawing $D$ of a plane embedding $F \in \mathcal{F}(V(f); G)$, it holds*

$$|\Lambda(D)| \geq a(f) + \max\{0, 3 - b(F; f)\}. \tag{4}$$

□

In general, equality in (4) does not hold. In fact, if $F$ contains a bi-facial cycle which is edge-disjoint with any of all minimum bi-facial cycles, then equality in (4) does not hold. For example, embedding $F_1$ in Figure 1(a) contains bi-facial cycle $C_4 = (u_6, u_8, u_2)$ which is edge disjoint with any subgraph in $L(f; G)$ with $f = (u_1, u_2, u_7)$.

We now consider a plane embedding $F$ of a planar graph $G = (V, E)$ and a candidate set $A \subseteq V$ of concave vertices to compute a straight-line drawing of $F$ whose concave vertices are given by $A$.

When an embedding $F$ of $G$ is given, we assume that, for a P-node $\nu$, the indices of the edges in $E_\nu^- = \{e_1, e_2, \ldots, e_k\}$ is given by the order in which the corresponding subgraphs of $G$ are embedded in $F$ (the subgraphs for $e_1$ and $e_k$ enclose other subgraphs).

**Definition 1.** *For a plane embedding $F \in \mathcal{F}(V(f); G)$, a subset $A \subseteq V$ is called* proper *if the following is satisfied.*

(i) *For each minimal bi-facial cycle $C \in L(f; G)$, $|A \cap (V(C) - T(C))| = 1$;*
(ii) *For each maximal multipath $M = \{Q_1, Q_2, \ldots, Q_k\} \in L(f; G)$, there is a path $Q_\ell$ such that $|A \cap (V(Q_i) - T(Q_i))| = 1$ for $i \in \{1, 2, \ldots, k\} - \{\ell\}$, and $A \cap (V(Q_\ell) - T(Q_\ell)) = \emptyset$, where if $M$ contains a single edge, then $Q_\ell$ is the edge;*
(iii) *$A$ contains $\max\{0, 3 - b(L; F)\}$ vertices from $V_{\nu^*}$;*
(iv) *For each P-node $\nu$ in the rooted SPQR-tree $\mathcal{T}_{\nu^*}$ and $E_\nu^- = \{e_1, e_2, \ldots, e_k\}$, there are indices $h'$ and $h$ as follows: $h' = h$ or $h' = h - 1$, and if $h' = h - 1$ then $e_h$ represents an induced $u, v$-path $Q$ such that $A \cap (V(Q) - \{u, v\}) = \emptyset$, where $(u, v) = parent(\nu)$. Moreover, $e_i$, $1 \le i \le h'$ is a virtual edge and its corresponding child node $\mu_i$ satisfies $A_{vu}(\mu_i) \neq \emptyset$ while $e_j$, $h + 1 \le j \le k$ is a virtual edge and its corresponding child node $\mu_j$ satisfies $A_{uv}(\mu_j) \neq \emptyset$ (see Figure 3(a)).*

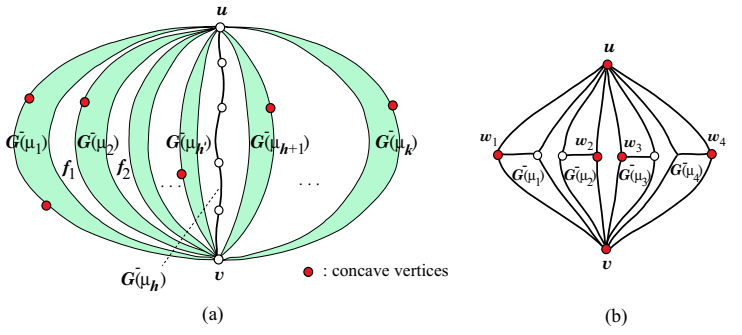



**Fig. 3.** (a)Definition 1(iv) for a P-node $\nu$; (b) An example of a non-proper set $A$.

Note that the total number of those vertices in $A \cap (V(C) - T(C))$ in (i) and $A \cap (V(Q_i) - T(Q_i))$ in (ii) is $a(f)$. Hence by (iii), a proper pair $(A, F)$ satisfies $|A| = a(f) + \max\{0, 3 - b(F, f)\}$. Condition (iv) is necessary for an embedding $F$ to have a straight-line drawing without introducing other concave vertices than those in $A$. For example, Figure 3(b) shows a subset $A = \{w_1, \ldots, w_4, u, v\}$

in a plane graph $G = (V, E, F)$ from which no straight-line drawing can be constructed without introducing some other concave vertices in $V - A$.

## 4 Finding a Best Embedding

This section shows how to maximize $b(F; f)$ over all $F \in \mathcal{F}(V(f), G)$ for a given face $f \in F(\sigma(\nu^*))$ of the skeleton of the root $\nu^*$ of the rooted SPQR tree of $G$.

**Lemma 8.** *Let $\mathcal{T}_{\nu^*}$ be the rooted SPQR-tree of a nontrivial biconnected planar graph $G$, and $f$ be a face of the skeleton $\sigma(\nu^*)$ of the root $\nu^*$ in $\mathcal{T}_{\nu^*}$. Then an embedding $F \in \mathcal{F}(V(f); G)$ that maximizes $b(F; f)$ and a proper set $A \subseteq V$ with $|A| = a(f) + \max\{b(F; f) \mid F \in \mathcal{F}(V(f); G)\}$ can be found in linear time.* □

We can compute an embedding $F$ that maximizes $b(F; f)$ in a bottom-up manner along the rooted SPQR tree $\mathcal{T}_{\nu^*}$. For a non-root node $\nu$ in the rooted SPQR tree $\mathcal{T}_{\nu^*}$ and its parent cut-pair $\{u, v\}$ of $\nu$, let $L(\nu; \nu^*)$ denote the set of minimal bi-facial cycles and maximal multipaths $H \in L(f; G)$ such that $(V(H) - T(H)) \cap \{u, v\} = \emptyset$ (note that $L(\nu; \nu^*)$ remains unchanged even if other face of $\sigma(\nu^*)$ is chosen as $f$). Let $b(\nu; \nu^*)$ be the maximum number of subgraphs $H \in L(\nu; \nu^*)$ that share an edge with the *$u, v$-boundary path* $Q_{uv}(G^-(\nu))$ of $G^-(\nu)$ (the maximum is taken over all embeddings in $\mathcal{F}(\{u, v\}; G^-(\nu))$). By definition, it is not difficult to see that $L(\nu; \nu^*)$ and $b(\nu; \nu^*)$ for all non root nodes $\nu$ can be computed in $O(|V|)$ time by a dynamic programming algorithm based on a recursive formula (the detail is omitted due to the space limitation). From $L(\nu; \nu^*)$ and $b(\nu; \nu^*)$, $\nu \in Ch(\nu^*)$, we can actually compute $a(f) + \max\{b(F; f) \mid F \in \mathcal{F}(V(f); G)\}$ for all faces $f \in F(\sigma(\nu^*))$ in $O(|V_{\nu^*}| + |E_{\nu^*}|)$ time. Let Hence a face $f_{\nu^*} \in F(\sigma(\nu^*))$ that maximizes $a(f) + \max\{b(F; f) \mid F \in \mathcal{F}(V(f); G)\}$ can be obtained in the same time complexity.

We can apply the above dynamic programming algorithm for other choice of root $\nu$ in the SPQR tree to find such a maximizer $f_\nu$ in $O(|V|)$ time. However, we can compute such $f_\nu$ for all nodes $\nu$ in $O(|V|)$ time by reusing the information used to compute $b(\nu; \nu^*)$ for all descendants $\nu$ of the first choice of root $\nu^*$. When we choose a node $\nu_1$ adjacent to the current root $\nu^*$ as a new root, we can compute $b(\nu; \nu_1)$, $\nu \in Ch(\nu_1)$ in $O(|V_{\nu_1}| + |E_{\nu_1}|)$ time. Therefore, we obtain a face $f^*$ of a skeleton $\sigma(\nu)$ of a node that maximizes $b(f^*)$ in $O(|V|)$ time, and we can conclude that the minimum number of concave corners in any straight-line drawing is at least $a(f^*) + \max\{0, 3 - b(f^*)\}$. For the optimal face $f^*$, we can also compute an embedding $F \in \mathcal{F}(V(f^*); G)$ and a proper subset $A \subseteq V$ in $F$ such that $|A| = a(f^*) + \max\{0, 3 - b(f^*; F)\}$ in $O(|V|)$ time.

The remaining task to prove Theorem 1 is to show that, given a proper set $A \subseteq V$ in a plane embedding $F$ of $G$, a star-shaped drawing $D$ with $\Lambda(D) = A$ under the specified embedding $F$ always exists and that such a drawing $D$ can be computed in linear time.

## 5 Constructing Star-shaped Drawings

This section describes how to construct a star-shaped drawing of a given proper pair $(A, F)$. We prove the following.

**Lemma 9.** *Let $\mathcal{T}_{\nu^*}$ be the rooted SPQR-tree of a biconnected planar graph $G$, and $f$ be a face of $\sigma(\nu^*)$ of the root $\nu^*$ in $\mathcal{T}_{\nu^*}$. Let $L = L(f; G)$, and $(A, F)$ be a proper pair. Then there exists a star-shaped drawing $D$ of $F$ such that $c(D) = a(L) + \max\{0, 3 - b(L; F)\}$, and such a drawing $D$ can be constructed in linear time.* □

Let $F$ be a plane embedding of $G$ and $A$ be a proper subset. For a node $\nu$ in the SPQR tree $\mathcal{T}_{\nu^*}$, let $f_F^o(H)$ denote the boundary of an induced subgraph $H$ of $G$ (where $H$ are regarded as plane graphs induced from $G$ by embedding $F$). We compute a star-shaped drawing $D$ with $\Lambda(D) = A$ in a top-down manner along the SPQR tree $\mathcal{T}_{\nu^*}$. We first explain key strategies to maintain a star-shaped drawing recursively:

1. After choosing an arbitrary $|A(\nu^*)|$-gon $B_{\nu^*}$ with $V(B_{\nu^*}) = A(\nu^*)$ for the root $\nu^*$, we process all nodes $\nu$ in $\mathcal{T}_{\nu^*}$ from the root to the leaves by repeatedly computing a drawing $D_\nu$ of skeleton $\sigma^-(\nu)$ (or $\sigma(\nu)$), where the line segments in $D_\nu$ for virtual edges will be replaced with new drawings $D_\mu$ of the nodes corresponding to the virtual edges.
2. When we process a non-root R-node $\nu$ whose parent is an R-node, we fix the boundary of $G^-(\nu)$ as an $(|A(\nu)| + 2)$-gon $B_\nu$, and then compute a convex drawing $D_\nu$ of skeleton $\sigma^-(\nu)$ as an extension of $B_\nu$ (we use the linear time convex drawing algorithm due to Chiba et al. [4] to compute such a convex extension).
3. When we process a non-root R-node $\nu$ whose parent is an R- (resp., S-node), we compute a convex drawing $D_\nu$ of skeleton $\sigma^-(\nu)$ (resp., $D_\nu$ of skeleton $\sigma(\nu)$), where the boundary of the skeleton has been fixed as a convex polygon $B_\nu$, and we b(we use the linear time convex drawing extend $B_\nu$ to such a drawing $D_\nu$.

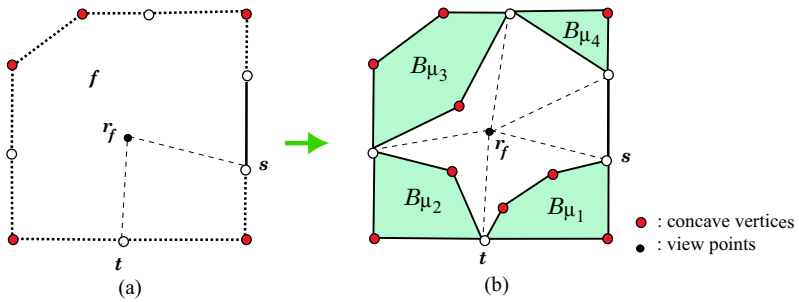



**Fig. 4.** (a) Fixing the boundary $B_\nu$ of $G^-(\nu)$ of an S-node $\nu$ of type I; (b) Fixing the boundaries $B_{\mu_i}$ of $G^-(\mu_i)$ of the child nodes $\mu_i \in Ch(\nu)$ of an S-node $\nu$ of type I.

4. When we process a non-root S-node $\nu$ whose parent is an R-node (resp., a P-node), the virtual edge corresponding to $\nu$ is drawn as a line segment $L_\nu$ (resp., a convex polygon $B_\nu$), and we then compute an $(|A(\mu)|+2)$-gon $B_\mu$ of its child node $\mu \in Ch(\nu)$, and replace $L_\nu$ with (resp., extend $B_\nu$ to) the sequence of these convex polygons $B_\mu$, $\mu \in Ch(\nu)$.
5. When we process a non-root P-node $\nu$ with $\{u, v\} = parent(\nu)$, the boundary of $G^-(\nu)$ has been fixed as $B_\nu$, and we fix the boundary $G^-(\mu)$ of each child node $\mu \in Ch(\nu)$ as an $(|A_{vu}(\mu)|+2)$-, $(|A_{uv}(\mu)|+2)$- or $(|A(\mu)|+2)$-gon $B_\nu$ depending on the position of the corresponding virtual edge in the indexing of Definition 1(iv).
6. When new inner faces are introduced after computing a drawing $D_\nu$ of $\sigma^-(\nu)$ (or $\sigma(\nu)$) of an R-node $\nu$, we choose a point $r_f$ inside each new face $f$ as the *view point* of $f$, which will be kept as a visible point in the kernel of the face $f$ until a final drawing is obtained.
7. A convex polygon $B_\nu$ for a node $\nu$ will be chosen so that the view point(s) $r_f$ of the face(s) adjacent to the corresponding virtual edge remain visible from everywhere in the face(s).

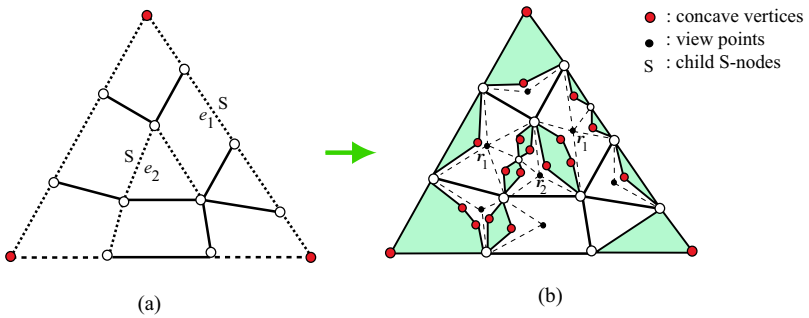



**Fig. 5.** (a) Fixing the boundary $B_{\nu^*}$ of $G$ for the root R-node $\nu^*$; (b) Fixing the boundaries of $G^-(\mu)$ and $G^-(\mu')$ for the child P- and R-nodes $\mu \in Ch(\nu)$ and the child P- and R-nodes $\mu' \in Ch(\mu)$ with child S-nodes $\mu \in Ch(\nu)$.

We distinguish type I with type II for each of S- and R-nodes. We briefly explain what we need to compute for each type of nodes. (A full description of the algorithm is omitted due to the space limitation.) When a vertex $u$ is drawn as a point in the plane, the point may be denoted by $u$ for notational simplicity.

**type I S-node** A non-root S-node $\nu$ is called *type I* if the virtual edge $e_\nu$ corresponding $\nu$ is an outer edge (resp., an edge) in the drawing for its parent R-node (resp., its parent P-node).
Input: The view point $r_f$ of the face incident to $e_\nu$ and a convex polygon $B_\nu$ drawn for $f^o_{vu}(G^-(\nu))$ (or $f^o_{uv}(G^-(\nu))$) are given.

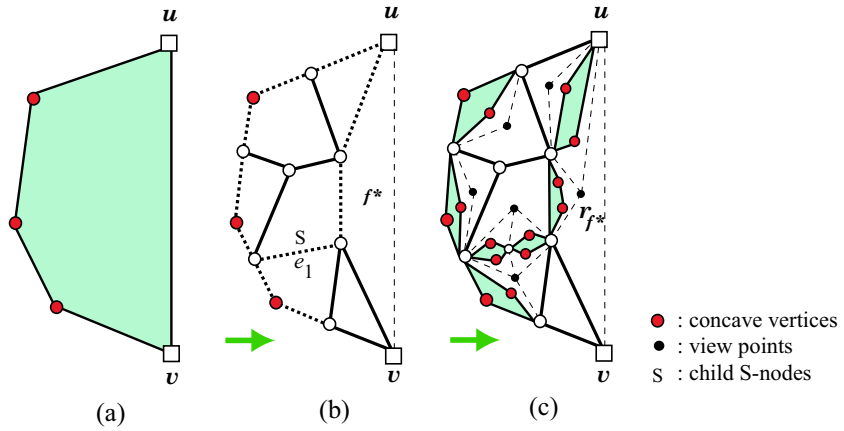



**Fig. 6.** (a) For an R-node $\nu$ of type I, a convex $(|A_{vu}(\nu)|+2)$-gon $B_\nu$ is drawn for $f^o_{uv}(\nu)$; (b) Extending $B_\nu$ into a convex drawing of $\sigma_{vu}(\nu)$; (c) Fixing the boundaries of $G^-(\mu)$ and $G^-(\mu')$ for the child P- and R-nodes $\mu \in Ch(\nu)$ and the child P- and R-nodes $\mu' \in Ch(\mu)$ with child S-nodes $\mu \in Ch(\nu)$.

Output: For each child node $\mu \in Ch(\nu)$, we place the parent cut-pair $(s,t)$ on $B_\nu$ and fix the boundary of $G^-(\mu)$ as a convex $(|A(\mu)|+2)$-gon $B_\mu$ with $V(B_\mu) = A(\mu) \cup \{s,t\}$ by combining the line segments of $B_\nu$ between $s$ and $t$ and a new line segments between $s$ and $t$ inside $B_\nu$ (see Figure 4, and the virtual edge $e_1$ in Figure 5).

**type II S-node** A non-root S-node $\nu$ is called *type II* if its parent node is an R-node, and the virtual edge $e_\nu$ corresponding $\nu$ is an inner edge in the drawing of the parent node.
Input: The view points $r_1$ and $r_2$ of the two faces incident to $e_\nu$ and a line segment $B_\nu = [u,v]$ drawn for $parent(\nu) = (u,v)$ are given.
Output: For each (R- or P-node) $\mu$, we place the parent cut-pair $(s,t)$ on $B_\nu$ and fix the boundary of $G^-(\mu)$ as a convex $(|A(\mu)|+2)$-gon $B_\mu$ with $V(B_\mu) = A(\mu) \cup \{s,t\}$ inside $[s,t,r_1] \cup [s,t,r_2]$ (see the virtual edge $e_2$ in Figure 5).

For a non-root node $\nu$ with $(u,v) = parent(\nu)$, two plane drawings for $\sigma(\nu)$ can be obtained from the plane graph $\sigma^-(\nu)$; one still has $f^o_{uv}(\sigma^-(\nu))$ as part of its boundary, and the other $f^o_{vu}(\sigma^-(\nu))$, where we denote the former and latter plane graphs by $\sigma_{uv}(\nu)$ and $\sigma_{vu}(\nu)$, respectively.

**type I R-node** A non-root R-node $\nu$ is called *type I* if a convex drawing of plane graph $\sigma_{vu}(\nu)$ (resp., $\sigma_{uv}(\nu)$) is required to be computed. Let $(u,v) = parent(\nu)$.
Input: A convex $(|A_{vu}(\nu)|+2)$-gon $B_\nu$ for $f^o_{vu}(\nu)$ (resp., $(|A_{vu}(\nu)|+2)$-gon $B_\nu$ for $f^o_{vu}(\nu)$) is given (see Figure 6(a)).

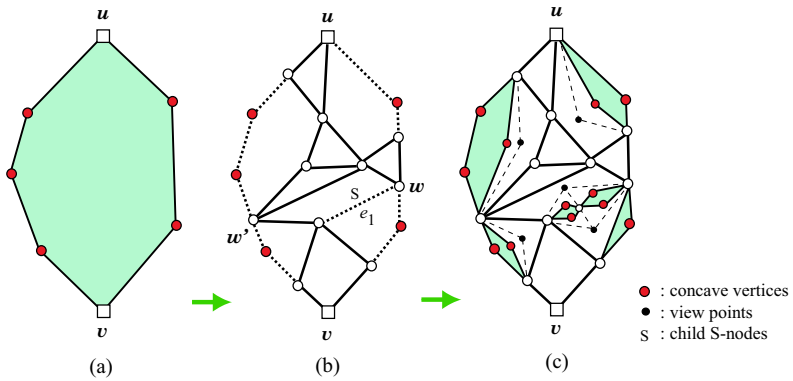



**Fig. 7.** (a) For an R-node $\nu$ of type II, a convex $(|A(\nu)|+2)$-gon $B_\nu$ is given as the boundary of $G^-(\nu)$; (b) Extending $B_\nu$ into a convex drawing of $\sigma^-(\nu)$; (c) Fixing the boundaries of $G^-(\mu)$ and $G^-(\mu')$ for the child P- and R-nodes $\mu \in Ch(\nu)$ and the child P- and R-nodes $\mu' \in Ch(\mu)$ with child S-nodes $\mu \in Ch(\nu)$.

Output: A convex drawing of $\sigma_{vu}(\nu)$ (resp., $\sigma_{uv}(\nu)$) as an extension of $B_\nu$ (see Figure 6(b)).

**type II R-node** An R-node $\nu$ is called *type II* if a convex drawing of $\sigma^-(\nu)$ is required to be computed.
Input: A convex $(|A(\nu)|+2)$-gon $B_\nu$ for the boundary of $G^-(\nu)$ is given (see Figure 7(a)), where a convex $|A(\nu^*)|$-gon $B_{\nu^*}$ for the boundary of $G$ is given if $\nu$ is the root $\nu^*$.
Output: A convex drawing of $\sigma^-(\nu)$ as an extension of $B_\nu$ (see Figure 7(b)).

**P-node** Let $\nu$ be a P-node. Let $u, v$ be the vertices in $V_{\nu^*}$ if $\nu$ is the root $\nu^*$ or the vertices in $parent(\nu)$ otherwise.
Input: A convex boundary $B_\nu$ of $G^-(\nu)$, where $(e_1, e_2, \ldots, e_{j^*}, \ldots, e_k)$ denotes the sequence of $E^-_\nu$, where $e_{j^*} = c(\nu)$ is the central edge, and $f_i$ denotes the face between two edges $e_i$ and $e_{i+1}$ in the plane graph $\sigma^-(\nu)$ (see Figure 8(a)).
Output: Drawings of $\sigma^-(\mu)$ of child nodes $\mu \in Ch(\nu)$. We process left edges in $E^-_\nu$ from $e_1$ to $e_{j^*-1}$ and right edges in $E^-_\nu$ from $e_k$ to $e_{j^*+1}$ before the central edge $e_{j^*}$ is processed. Left edges: If left edge $e_i$ corresponds to an S-node $\mu_i$, then we choose a view point $r_{f_i}$ and treat $\mu_i$ as a type I S-node . If left edge $e_i$ corresponds to an R-node $\mu_i$, then treat $\mu_i$ as a type I R-node, and compute a convex drawing $D_\mu$ of $\sigma_{vu}(\mu_i)$ (including virtual edge $parent(\nu)$), where the view point $r_{f_i}$ is computed during the computation of $D_\mu$ (see Figure 9(a) and (b)).
Right edges: we apply the above procedure symmetrically to right edges $e_k, e_{k-1}, \ldots, e_{j^*+1}$.
We treat the central edge $e_{j^*}$ as follows. If $e_{j^*}$ corresponds to an R-node $\mu_{j^*}$, then we then treat $\mu_{j^*}$ as a type II R-node.

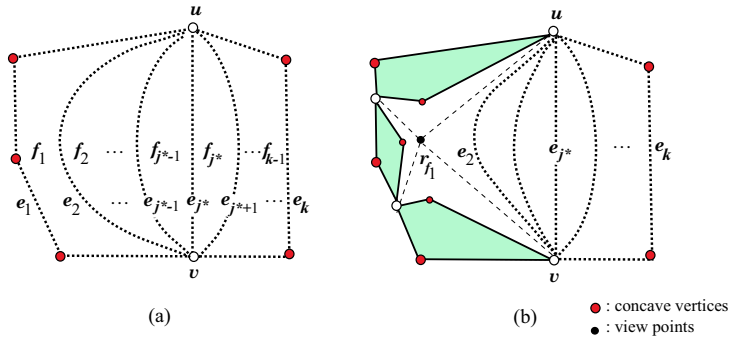



**Fig. 8.** (a) $B_\nu$ for the root P-node or an internal P-node $\nu$; (b) $e_1$ corresponds to an S-node $\mu_1$, where SNODE($\mu_1, B_{\mu_1}, r_{f_1}, \emptyset$,I) is executed after choosing view point $r_{f_1}$.

Since we can show that a boundary of $B_\nu$ of a node $\nu$ can be fixed in linear time of the size of $B_\nu$, the entire algorithm can be implemented to run in $O(|V|+|E|)$ time. This proves Lemma 9.

## 6 Concluding Remarks

In this paper, we considered the problem of finding a star-shaped drawing of a given biconnected *planar* graph with the minimum number of concave corners. By deriving an effective lower bound on the number of concave corners, we proved that the problem can be solved in linear time. A natural question related to the problem is whether the problem of finding a straight-line drawing that minimizes the number of concave corners for a *given* plane embedding $F$ of a biconnected planar graph is hard or not. Remember that (4) does not hold by equality in general, as observed in the example in Fig. 1(a). However, recently we showed that the problem can be solved in linear time [9].

## References

1. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice-Hall, 1998.
2. G. Di Battista and R. Tamassia, On-line planarity testing, SIAM J. on Comput. 25(5), pp. 956-997, 1996.
3. G. Di Battista and R. Tamassia, On-line maintenance of triconnected components with SPQR-trees, Algorithmica, 15, pp. 302-318, 1996.
4. N. Chiba, T. Yamanouchi and T. Nishizeki, Linear algorithms for convex drawings of planar graphs, Progress in Graph Theory, Academic Press, pp. 153-173, 1984.
5. I. Fáry, On straight line representations of planar graphs, Acta Sci. Math. Szeged, 11, pp. 229-233, 1948.

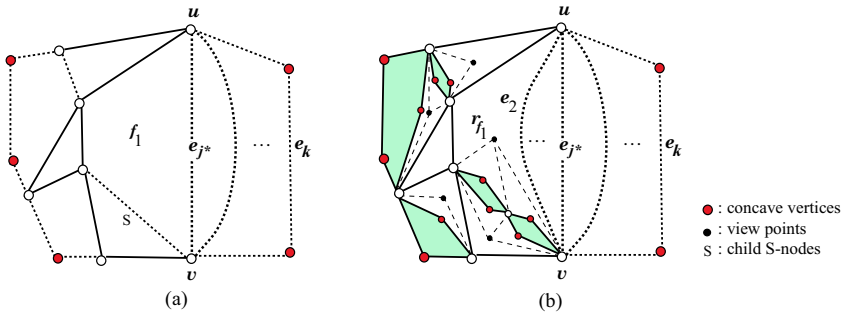



**Fig. 9.** (a) $e_1$ corresponds to an R-node $\mu$ of type I, where RNODE($\mu_1, B_{\mu_1}, \sigma_{vu}(\mu_1)$,I) computes a convex drawing $D_\mu$ of $\sigma_{vu}(\mu)$ and returns a view point $r_{f_1}$; (b) RNODE($\mu_1, B_{\mu_1}, \sigma_{vu}(\mu_1)$,I) draws the boundaries of child nodes of $\mu_1$.


6. S.-H. Hong and H. Nagamochi, Convex drawings of graphs with non-convex boundary, Proc. of WG 2006, Lecture Notes in Computer Science, vol. 4271, Springer-Verlag, pp. 113-124, 2006.
7. S.-H. Hong and H. Nagamochi, Convex drawings of hierarchical plane graphs, 17th Australasian Workshop on Combinatorial Algorithms (AWOCA 2006) Uluru, NT, Australia, July 13-16, 2006.
8. S.-H. Hong and H. Nagamochi, Fully convex drawings of clustered planar graphs, Proc. of Korea-Japan Joint Workshop on Algorithms and Computation, August 9-10, 2007, Chonnam National University, Gwangju, Korea, pp. 32–39.
9. S.-H. Hong and H. Nagamochi, Star-shaped drawings of plane graphs, Workshop on Algorithms and Computation 2008 (WALCOM 2008) February 7-8, 2008 at Dhaka, Bangladesh (submitted).
10. J. E. Hopcroft and R. E. Tarjan, Dividing a graph into triconnected components, SIAM J. on Comput., 2, pp. 135-158, 1973.
11. G. Kant, Algorithms for Drawing Planar Graphs, Ph.D. Dissertation, Department of Computer Science, University of Utrecht, Holland, 1993.
12. K. Miura, M. Azuma and T. Nishizeki, Convex drawings of plane graphs of minimum outer apices, Int. J. Found. Comput. Sci., 17, pp. 1115-1128, 2006.
13. C. Thomassen, Planarity and duality of finite and infinite graphs, J. of Combinatorial Theory, Series B 29, pp. 244-271, 1980.
14. W. T. Tutte, Convex representations of graphs, *Proc. of London Math. Soc.*, 10, no. 3, pp. 304-320, 1960.
15. W. T. Tutte, Graph Theory, Encyclopedia of Mathematics and Its Applications, Vol. 21, Addison-Wesley, Reading, MA, 1984.

# Algorithms for Two Versions of LCS Problem for Indeterminate Strings*

Costas S. Iliopoulos[1,4,**], M. Sohel Rahman[1,4,***,†], and Wojciech Rytter[2,3,5,‡]

[1] Algorithm Design group
Department of Computer Science
King's College London
Strand, London WC2R 2LS, England
http://www.dcs.kcl.ac.uk/adg
[2] Institute of Informatics
Warsaw University, Warsaw, Poland,
[3] Department of Mathematics and Informatics
Copernicus University, Torun, Poland
[4] {sohel,csi}@dcs.kcl.ac.uk
[5] rytter@mimuw.edu.pl

**Abstract.** We study the complexity of the longest common subsequence (LCS) problem from a new perspective. By an indeterminate string (i-string, in short) we mean a sequence $\widetilde{X} = \widetilde{X}[1]\widetilde{X}[2]\ldots\widetilde{X}[n]$, where $\widetilde{X}[i] \subseteq \Sigma$ for each $i$, and $\Sigma$ is a given alphabet of potentially large size. A subsequence of $\widetilde{X}$ is any usual string over $\Sigma$ which is an element of the finite (but usually of exponential size) language $\widetilde{X}[i_1]\widetilde{X}[i_2]\ldots\widetilde{X}[i_p]$, where $1 \leq i_1 < i_2 < i_3 \ldots < i_p \leq n, p \geq 0$. Similarly, we define a supersequence of $x$. Our first version of the LCS problem is Problem ILCS: for given i-strings $\widetilde{X}$ and $\widetilde{Y}$, find their longest common subsequence. From the complexity point of view, new parameters of the input correspond to $|\Sigma|$ and maximum size $\ell$ of the subsets in $\widetilde{X}$ and $\widetilde{Y}$. There is also a third parameter $\mathcal{R}$, which gives a measure of similarity between $\widetilde{X}$ and $\widetilde{Y}$. The smaller the $R$, the lesser is the time for solving Problem ILCS. Our second version of the LCS problem is Problem CILCS (constrained ILCS): for given i-strings $\widetilde{X}$ and $\widetilde{Y}$ and a plain string $Z$, find the longest common subsequence of $\widetilde{X}$ and $\widetilde{Y}$ which is, at the same time, a supersequence of $Z$. In this paper, we present several efficient algorithms to solve both ILCS and CILCS problems. The efficiency in our algorithms are obtained in particular by using an efficient data structure for special type of range maxima queries and fast multiplication of boolean matrices.

* Part of this research work was carried out when Costas Iliopoulos and M. Sohel Rahman were visiting Institute of Informatics, Warsaw University.
** Supported by EPSRC and Royal Society grants.
*** Supported by the Commonwealth Scholarship Commission in the UK under the Commonwealth Scholarship and Fellowship Plan (CSFP).
† On Leave from Department of CSE, BUET, Dhaka-1000, Bangladesh.
‡ Supported by the grant of the Polish Ministery of Science and Higher Education N 206 004 32/0806.

## 1 Introduction

This paper deals with two interesting variants of the classical and well-studied longest common subsequence (LCS) problem: LCS for indeterminate strings (i-string) and Constrained LCS (CLCS) problem, also for i-strings. In i-strings, at each position, the string may contain a set of characters. The LCS problem and variants thereof have been focus of extensive research in the computer science literature since long. Given two strings, the LCS problem consists of computing a subsequence of maximum length common to both strings. In CLCS, the computed longest common subsequence must also be a supersequence of a third given string. The motivation of CLCS problem comes from bioinformatics: in the computation of the homology of two biological sequences it is important to take into account a common specific or putative structure [25].

The longest common subsequence problem for $k$ strings ($k > 2$) was first shown to be NP-hard [17] and later proved to be hard to be approximated [15]. The restricted but probably the more studied problem that deals with two strings has been studied extensively. The classic dynamic programming solution to LCS problem, invented by Wagner and Fischer [27], has $O(n^2)$ worst case running time, where $n$ is the length of the two strings. Masek and Paterson [18] improved this algorithm using the "Four-Russians" technique [1] to reduce the worst case running time to $O(n^2/\log n)$[6]. Since then not much improvement in terms of $n$ can be found in the literature. However, several algorithms exist with complexities depending on other parameters. For example, Myers in [20] and Nakatsu et al. in [21] presented an $O(nD)$ algorithm, where the parameter $D$ is the simple Levenshtein distance between the two given strings [16]. Another interesting and perhaps more relevant parameter for this problem is $\mathcal{R}$, where $\mathcal{R}$ is the total number of ordered pairs of positions at which the two strings match. Hunt and Szymanski [11] presented an algorithm running in $O((\mathcal{R}+n)\log n)$ time. They also cited applications where $\mathcal{R} \sim n$ and thereby claimed that for these applications the algorithm would run in $O(n \log n)$ time. Very recently, Rahman and Iliopoulos presented an improved LCS algorithm running in $O(\mathcal{R} \log\log n + n)$ time [23]. Notably, an $O(\mathcal{R} \log\log n)$ time algorithm for LCS was also reported in [19]; but their running time excludes a costly preprocessing time of $O(n^2 \log n)$. For a comprehensive comparison of the well-known algorithms for LCS problem and study of their behaviour in various application environments the readers are referred to [5].

The CLCS problem, on the other hand, has been introduced quite recently by Tsai in [25]. In [25], a dynamic programming formulation for CLCS was presented leading to a $O(pn^4)$ time algorithm to solve the problem, where $p$ is the length of the third string which applies the constraint. Later, Chin et al. [7] and independently, Arslan and Eğecioğlu [2] presented improved CLCS algorithm with $O(pn^2)$ time and space complexity. The problem was also studied

[6] Employing different techniques, the same worst case bound was achieved in [10]. In particular, for most texts, the achieved time complexity in [10] is $O(hn^2/\log n)$, where $h \leq 1$ is the entropy of the text.

very recently in [13], where an algorithm running in $O(p\mathcal{R}\log\log n+n)$ time was devised.

In this paper, we revisit the LCS and CLCS problems, but in a different setting: instead of standard strings, we consider i-strings, where at each position the string may contain a set of characters. The motivation of our study comes from the fact that i-strings are extensively used in molecular biology to express polymorphism in DNA sequences, e.g. the polymorphism of protein coding regions caused by redundancy of the genetic code or polymorphism in binding site sequences of a family of genes. To the best of our knowledge, the only work in the literature in this context is the recent paper [14], where a finite automata based solution for the CLCS problem for i-strings was presented with worst case running time $O(|\Sigma|pn^2)$. Here we present a number of improved algorithms to solve both LCS and CLCS problems for i-strings. In particular, we have used some novel techniques to preprocess the given i-strings, which let us use the corresponding solutions for normal strings to get efficient solution for i-strings.

The rest of the paper is organized as follows. In Section 2, we present the preliminary concepts and formally define the problems handled in this paper. In Section 3, we handle Problem LCS for i-strings (Problem ILCS). To elaborate, in Sections 3.1 to 3.3, we present three different preprocessing steps to get efficient algorithms to solve Problem ILCS. In particular, in Section 3.1, we reduce the problem at hand to boolean matrix multiplication problem and uses the fast multiplication of boolean matrices to gain efficiency. In Sections 3.2 and 3.3 we take a different approach and consider algorithms with running time dependent on $\mathcal{R}$. Particularly in Section 3.3, we have (implicitly) used efficient data structure of [23] for special type of range maxima queries to get efficient algorithms for ILCS. Notably, the use of RMQ in solving LCS problems and variants thereof was explored in [19] and later in [12, 22, 23]. Thus, the preprocessing step in Section 3.3, could be viewed as an extension of the work of [19, 12, 22, 23]. We handle the CLCS problem for i-strings (Problem ICLCS) in Section 4. In particular, we present two different preprocessing steps for ICLCS in Sections 4.1 and 4.2 extending the ideas and techniques used in Sections 3.1 and 3.2 respectively. Finally, we briefly conclude in Section 5.

## 2 Preliminaries

We use $LCS(X,Y)$ to denote a longest common subsequence of $X$ and $Y$. We denote the length of $LCS(X,Y)$ by $\mathcal{L}(X,Y)$. Given two strings $X[1..n]$ and $Y[1..n]$ and a third string $Z[1..p]$, a common subsequence $S$ of $X,Y$ is said to be *constrained* by $Z$ if, and only if, $Z$ is a subsequence of $S$. We use $LCS_Z(X,Y)$ to denote a longest common subsequence of $X$ and $Y$ that is constrained by $Z$. We denote the length of $LCS_Z(X,Y)$ by $\mathcal{L}_Z(X,Y)$.

*Example 1.* Suppose $X = TCCACA$, $Y = ACCAAG$ and $Z = AC$. As is evident from Fig. 1, $S^1 = CCAA$ is an $LCS(X,Y)$. However, $S^1$ is not an $LCS_Z(X,Y)$ because $Z$ is not a subsequence of $S^1$. On the other hand, $S^2 = ACA$ is an $LCS_Z(X,Y)$. Note that, in this case $r_Z(X,Y) < r(X,Y)$.


**Fig. 1.** $|LCS(X,Y)| = 4$ and $|LCS_Z(X,Y)| = 2$.

In this paper, we are interested in *indeterminate* strings (*i-strings*, in short). In contrast, usual strings are called here *standard* strings. A string $\widetilde{X}[1..n]$ is said to be indeterminate, if it is built over the potential $2^{|\Sigma|}-1$ non-empty sets of letters belonging to $\Sigma$. Each $\widetilde{X}[i], 1 \leq i \leq n$ can be thought of as a set of characters and we have $|\widetilde{X}[i]| \geq 1, 1 \leq i \leq n$. The length of the i-string $\widetilde{X}$, denoted by $|\widetilde{X}|$, is the number of sets (of characters) in it, i.e., $n$. In this paper, the set containing the letters $A$ and $C$ will be denoted by $[AC]$ and the singleton $[C]$ will be simply denoted by $C$ for ease of reading. Also, we use the following convention: we use plain letters like $X$ to denote normal strings. The same letter may be used to denote a i-string if written as $\widetilde{X}$. For i-strings, the notion of symbol equality is extended to single-symbol match between two (indeterminate) letters in the following way. Given two subsets $A, B \subseteq \Sigma$ we say that $A$ matches $B$ and write $A \approx B$ iff $A \cap B \neq \emptyset$. Note that, the relation $\approx$, referred to as the 'indeterminate equality' henceforth, is not transitive.

*Example 2.*
Suppose we have i-strings $\widetilde{X} = AC[CTG]TG[AC]C$ and $\widetilde{Y} = TC[AT][AT]TTC$. Here we have $\widetilde{X}[3] \approx \widetilde{Y}[3]$ because $\widetilde{X}[3] = [CTG] \cap \widetilde{Y}[3] = [AT] = T \neq \emptyset$. Similarly we have, $\widetilde{X}[3] \approx \widetilde{Y}[1]$, and also $\widetilde{X}[3] \approx \widetilde{Y}[2]$ etc.

We can extend the notion of a subsequence for i-strings in a natural way replacing the equality of symbols by the relation $\approx$ as follows. A subsequence of $\widetilde{X}$ is a plain string $U$ over $\Sigma$ which is an element of the finite (but usually of exponential size) language $\widetilde{X}[i_1]\widetilde{X}[i_2]\ldots\widetilde{X}[i_p]$, where $1 \leq i_1 < i_2 \ldots < i_p \leq n, p \geq 0$.
Similarly, we define a supersequence of $\widetilde{X}$. The notion of common and longest common subsequence for i-strings can now be extended easily. In what follows, for the ease of exposition, we assume that $|\widetilde{X}| = |\widetilde{Y}| = n$. But our results can be easily extended when $|\widetilde{X}| \neq |\widetilde{Y}|$. We are interested in the following two problems.

***Problem "ILCS" (LCS for Indeterminate Strings).*** *Given* 2 *i-strings* $\widetilde{X}$ *and* $\widetilde{Y}$ *we want to compute an* $LCS(\widetilde{X}, \widetilde{Y})$.

***Problem "CILCS" (CLCS for Indeterminate Strings).*** *Given* 2 *i-strings* $\widetilde{X}$ *and* $\widetilde{Y}$ *and another (plain) string* $Z$*, we want to compute an* $LCS_Z(\widetilde{X}, \widetilde{Y})$.

*Example 3.* Suppose, we are given the i-strings

$$\widetilde{X} = [AF]BDDAAA,\ \widetilde{Y} = [AC]BA[CD]AA[DF],\ Z = BDD$$

Figure 2 shows an $LCS(\widetilde{X}, \widetilde{Y})$ and an $LCS_Z(\widetilde{X}, \widetilde{Y})$. Note that, although $\mathcal{L}(\widetilde{X}, \widetilde{Y}) = 5$, $\mathcal{L}_Z(\widetilde{X}, \widetilde{Y}) = 4$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\widetilde{X}$ | $[AF]$ | $B$ | | $D$ | | $D$ | $A$ | $A$ | $A$ | |
| $\widetilde{Y}$ | $[AC]$ | $B$ | $A$ | $[CD]$ | | | $A$ | $A$ | $[DF]$ | |
| An $LCS(\widetilde{X}, \widetilde{Y})$ | $A$ | $B$ | | $D$ | | | $A$ | $A$ | | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\widetilde{X}$ | $[AF]$ | $B$ | | $D$ | | | $D$ | $A$ | $A$ | $A$ |
| $\widetilde{Y}$ | $[AC]$ | $B$ | $A$ | $[CD]$ | $A$ | $A$ | $[DF]$ | | | |
| $Z$ | | $B$ | | $D$ | | | $D$ | | | |
| An $CLCS_Z(\widetilde{X}, \widetilde{Y})$ | $A$ | $B$ | | $D$ | | | $D$ | | | |

**Fig. 2.** $LCS(\widetilde{X}, \widetilde{Y})$ and $LCS_Z(\widetilde{X}, \widetilde{Y})$ of Example 3.

## 3 Algorithm for ILCS

In this section, we devise efficient algorithms for Problem ILCS. We start with a brief review of the traditional dynamic programming technique employed to solve LCS [27] for standard strings. Here the idea is to determine the longest common subsequences for all possible prefix combinations of the input strings. The recurrence relation for extending the length of LCS for each prefix pair $(X[1..i], Y[1..j])$, i.e. $\mathcal{L}(X[1..i], Y[1..j])$, is as follows [27]:

$$\mathcal{T}[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ max(\mathcal{T}[i-1, j-1] + \delta(i,j),\ \mathcal{T}[i-1, j],\ \mathcal{T}[i, j-1]), & \text{otherwise,} \end{cases} \tag{1}$$

where $\delta(i,j) = 0$ if $X[i] = Y[j]$; otherwise $\delta(i,j) = 1$.

Here we have used the tabular notion $\mathcal{T}[i,j]$ to denote $\mathcal{L}(X[1..i], Y[1..j])$. After the table has been filled, $\mathcal{L}(X,Y)$ can be found in $\mathcal{T}[n,n]$ and $LCS(X,Y)$ can be found by backtracking from $\mathcal{T}[n,n]$ (for detail please refer to [27] or any textbook on algorithms, e.g. [9]). It is easy to see that Equation 1 can be realized easily in $O(n^2)$ time.

Interestingly, Equation 1 can be adapted to solve Problem ILCS quite easily. The only thing we need to do is to replace the equality check ('=') in Equation 1 with the indeterminate equality ('$\approx$'). However, this 'simple' change affects the running time of the algorithm because instead of the constant time equality check we need to perform intersection between two sets. To deduce the precise running time of the resulting algorithm, we make the assumption that the sets of characters for each input i-strings are given in sorted order. Under this assumption, we can perform the intersection operation in $O(|\Sigma|)$ time in the worst case, since there can be at most $|\Sigma|$ characters in a set of a i-string. So we have the following theorem.

**Theorem 1.** *Problem ILCS can be solved in $O(|\Sigma|n^2)$ time.*

In the rest of this section, we try to devise algorithms giving better running times than what is reported in Theorem 1. We assume that the alphabet $\Sigma$ is

indexed, which is the case in most practical situations. We also assume that the sets of characters for each input i-strings are given in sorted order. Recall that the latter assumption is required to get the running time of Theorem 1. To improve the running time, we plan to do some preprocessing to realize Equation 1 more efficiently. In particular, we want to preprocess the two given strings so that the indeterminate equality check can be realized in $O(1)$ time. In the next subsections, we present three different preprocessing steps and analyze the time and space complexity of the resulting algorithms.

### 3.1 Preprocessing 1 for ILCS

Here the idea is to first compute a table $\mathcal{I}[i,j], 1 \leq i,j \leq n$ as defined below:

$$\mathcal{I}[i,j] = \begin{cases} 1 & \text{If } \widetilde{X}[i] \bigcap \widetilde{Y}[j] \neq \emptyset \\ 0 & \text{Otherwise.} \end{cases} \tag{2}$$

It is easy to realize that, with that table $\mathcal{I}$ in our hand, we can realize Equation 1 in $O(n^2)$ time because the indeterminate equality check reduces to the constant time checking of the corresponding entry in $\mathcal{I}$-table. Now it remains to see how efficiently we can compute the table $\mathcal{I}$. Recall that, our ultimate goal is to get a overall running time better than the one reported in Theorem 1. To compute the table $\mathcal{I}$, we first encode each $\widetilde{X}[i], 1 \leq i \leq n$ and $\widetilde{Y}[j], 1 \leq j \leq n$ as a binary vector of size $|\Sigma|$ as follows. We use $\widetilde{X}_e$ and $\widetilde{Y}_e$ to denote the encodings for $\widetilde{X}$ and $\widetilde{Y}$, respectively. For all $1 \leq i \leq n$ and $c \in |\Sigma|$, the encoding for $\widetilde{X}[i]$ is defined below:

$$\widetilde{X}_e[i][c] = \begin{cases} 1 & \text{If } c \in \widetilde{X}[i] \\ 0 & \text{Otherwise.} \end{cases} \tag{3}$$

The encoding for $\widetilde{Y}$ is defined analogously. Now, we can view $\widetilde{X}_e$ and $\widetilde{Y}_e$ as two ordered lists having $n$ binary vectors each, where each vector is of size $|\Sigma|$. And it is easy to realize that the computation of $\mathcal{I}$ reduces to the matrix multiplication of $\widetilde{X}_e$ and $\widetilde{Y}_e$. To speed-up this computation, we perform the following trick. Without loss of generality, we assume that $n$ is divisible by $|\Sigma|$ We divide both the boolean matrices $\widetilde{X}_e$ and $\widetilde{Y}_e$ in square partitions, each partition having size $|\Sigma| \times |\Sigma|$. Now we can perform the matrix multiplication by performing square matrix multiplication of the constituent square blocks.

Next, we analyze the running time of the preprocessing discussed above. The encoding of the two input i-strings $\widetilde{X}$ and $\widetilde{Y}$ require $O(n|\Sigma|)$ time and space. For square matrix multiplication, the best known algorithm is due to Coppersmith and Winograd [8]. Their algorithm works in $O(\mathcal{N}^{2.376})$ time, where the involved matrices are of size $\mathcal{N} \times \mathcal{N}$. Now, recall that in our case the square matrices are of size $|\Sigma| \times |\Sigma|$. Also, it is easy to see that, in total, we need $(n/|\Sigma|)^2$ such computation. Therefore, the worst case computational effort required is $O((n/|\Sigma|)^2 \times |\Sigma|^{2.376}) = O(n^2|\Sigma|^{0.376})$. To sum up, the total time required to solve Problem ILCS is $O(n|\Sigma| + n^2|\Sigma|^{0.376}) + n^2) = O(n|\Sigma| + n^2|\Sigma|^{0.376}))$ in the worst case. This implies the following result.

**Theorem 2.** *Problem ILCS can be solved in* $O(n|\Sigma| + n^2|\Sigma|^{0.376})$ *time.*

Before concluding this section, we briefly review some other boolean matrix multiplication results that may be used in our algorithm. There is a simple so called "Four Russians" algorithm of Arlazarov et al. [1], which performs Boolean $\mathcal{N} \times \mathcal{N}$ matrix multiplication in $O(\mathcal{N}^3/\log \mathcal{N})$ time. This was eventually improved slightly to $O(\mathcal{N}^3/\log^{1.5} \mathcal{N})$ time by Atkinson and Santoro [3]. Rytter [24] and independently Basch, Khanna, and Motwani [4] gave an $O(\mathcal{N}^3/\log^2 \mathcal{N})$ algorithm for Boolean matrix multiplication on the $(\log n)$-word RAM. Similar result also follows from a very recent paper [28]. Therefore problem ILCS can be solved in $O(n|\Sigma| + n^2|\Sigma|/\log^2 |\Sigma|))$ time without algebraically sophisticated matrix multiplication.

### 3.2 Preprocessing 2 for ILCS

We first present some notations needed to discuss the next preprocessing phase. We define the term $\ell$ as follows:

$$\ell = \max\{|\widetilde{X}[i]|, |\widetilde{Y}[i]| \mid 1 \leq i \leq n\}.$$

Also, we say a pair $(i, j), 1 \leq i, j \leq n$ defines a match, if $\widetilde{X}[i] \approx \widetilde{Y}[j]$. The set of all matches, $\mathcal{M}$, is defined as follows:

$$\mathcal{M} = \{(i, j) \mid \widetilde{X}[i] \approx \widetilde{Y}[j], 1 \leq i, j \leq n\}.$$

Observe that, $\mathcal{R} = |\mathcal{M}|$.

**Algorithm 1** Computation of the Table $\mathcal{I}$

```
1: for each i, j do  I[i, j] = 0;
2: for each a ∈ Σ do
3:     L_X̃[a] := ∅; L_Ỹ[a] := ∅;
4: for i = 1 to n do
5:     for each a ∈ X̃[i] do   insert(i, L_X̃[a])
6:     for each b ∈ Ỹ[i] do   insert(i, L_Ỹ[b])
7: for each a ∈ Σ do
8:     for each i ∈ L_X̃[a] do
9:         for each j ∈ L_Ỹ[a] do   I[i, j] = 1.
```

In the algorithm we pre-compute $\mathcal{M}$, and then fill up the table $\mathcal{I}$. We proceed as follows. We construct, for each symbol $a \in \Sigma$, two separate lists, $L_{\widetilde{X}}[a]$ and $L_{\widetilde{Y}}[a]$. For each $a \in \Sigma$, $L_{\widetilde{X}}[a]$ ($L_{\widetilde{Y}}[a]$) stores the positions where $a$ appears in $\widetilde{X}$ ($\widetilde{Y}$), if any. We have for $1 \leq i, j \leq n$

$$\mathcal{I}[i, j] \;=\; 1 \;\Leftrightarrow\; \exists\, (a \in \Sigma) \text{ such that } (i \in L_{\widetilde{X}}[a]) \text{ and } (j \in L_{\widetilde{Y}}[a])$$

The initialization of the table $\mathcal{I}$ requires $O(n^2)$ time. The constructions of the lists $L_{\widetilde{X}}[a], L_{\widetilde{Y}}[a], a \in \Sigma$ can be done in $(n\ell)$ time simply by scanning $\widetilde{X}$ and $\widetilde{Y}$ in turn. Traversing the two lists $L_X[a]$ and $L_Y[a]$ for each $a \in \Sigma$ to fill up $\mathcal{I}$ requires $O(\mathcal{R}\ell)$ time. This follows from the fact that there are in total $\mathcal{R}$ positions where we can have a match and at each such position we can have up to $\ell$ matches. Thus the total running time required for the preprocessing is $O(n\ell + n^2 + \mathcal{R}\ell)$.

**Theorem 3.** *Problem ILCS can be solved in $O(n\ell + n^2 + \mathcal{R}\ell)$ time.*

We remark that, in the worst case we have $\ell = |\Sigma|$. However, $\ell$ can be much smaller than $|\Sigma|$ in many cases. Also, $\mathcal{R}\ell$ and $n\ell$ are 'pessimistic' upper bounds in the sense that rarely for all $1 \leq i \leq n$, we will have $|\widetilde{X}[i]| = \ell$ ($|\widetilde{Y}[i]| = \ell$). Also, in the worst case we have $\mathcal{R} = O(n^2)$. But in many practical cases $\mathcal{R}$ turns out to be $o(n^2)$, or even $O(n)$. Another remark is that to compute an actual LCS, we will additionally require $O(n\ell)$ time in the worst case.

### 3.3 Preprocessing 3 for ILCS

The preprocessing of Section 3.2 can be slightly modified to devise an efficient algorithm based on the parameter $\mathcal{R}$. There exist efficient algorithms for computing LCS depending on parameter $\mathcal{R}$. The recent work of Rahman and Iliopoulos [23] presents an $O(\mathcal{R}\log\log n + n)$ algorithm (referred to as LCS-II in [23]) for computing the LCS. LCS-II computes the set $\mathcal{M}$, sorts it in a 'prescribed' order and then considers each $(i, j) \in \mathcal{M}$ and do some useful computation (instead of performing computation for all $n^2$ entries in the usual dynamic programming matrix). The efficient computation in LCS-II is based on the use of the famous vEB data structure invented by van Emde Boas [26] to solve a restricted dynamic version of the Range Maxima Query problem. The vEB data structure allows us to maintain a sorted list of integers in the range $[1..n]$ in $O(\log\log n)$ time per insertion and deletion. In addition to that it can return $next(i)$ (successor element of $i$ in the list) and $prev(i)$ (predecessor element of $i$ in the list) in constant time. On the other hand, the range maxima query (RMQ) problem is to preprocess an array of numbers to answer queries to find the maximum in a given range of the array. In [23], the authors observed that to compute the LCS, one needs only solve a restricted but dynamic version of RMQ problem[7]. Using the vEB structure, they then solved this restricted RMQ problem in $O(\mathcal{R}\log\log n)$ time per update and per query, which leads to an overall $O(\mathcal{R}\log\log n + n)$ time algorithm for computing the LCS. Now the essential thing about LCS-II is that if $\mathcal{M}$ is computed and supplied to it, it can compute the LCS in $O(\mathcal{R}\log\log n)$ time. Based on the results of [23], we get the following theorem.

**Theorem 4.** *Problem ILCS can be solved in $O(\mathcal{R}\ell + \mathcal{R}\log\log n + n)$ time.*

[7] Similar ideas were also utilized in [19] to solve LCS although employing a slightly different strategy and using a different data structure.

*Proof. (Sketch)*
We can slightly change Algorithm 1 to compute the set $\mathcal{M}$ instead of computing the table $\mathcal{I}$. This can be done simply by replacing Step 13 of Algorithm 1 with the following statement:

$\mathcal{M} = \mathcal{M} \bigcup (i, j)$.

We also need to initialize $\mathcal{M}$ to $\emptyset$ just before the for loop of Step 9.

Now, for plain strings, $\mathcal{M}$ can be computed in $O(\mathcal{R})$ time. But, for i-strings, it requires $O(\mathcal{R}\ell)$ time, because at each match position we may have up to $\ell$ matches in the worst case. However, recall that our goal is to use LCS-II for which we need $\mathcal{M}$ to be in a prescribed order. This however, can be done using the same preprocessing algorithm (Algorithm Pre) used in [23]. Algorithm Pre computes $\mathcal{M}$ requiring $O(\mathcal{R} + n)$ time (for normal strings) and using the vEB structures maintain the prescribed order spending $O(\mathcal{R} \log \log n)$ time. In our case, we have already computed $\mathcal{M}$ for the degenerate strings, and hence, we use Algorithm Pre, only to maintain the orders. Therefore, in total our preprocessing requires $O(\mathcal{R}\ell + \mathcal{R} \log \log n)$ time. Finally, once $\mathcal{M}$ (for the degenerate strings) is computed in the prescribed order, we can employ LCS-II directly to solve Probelem ILCS, requiring a further $O(\mathcal{R} \log \log n)$ time. Therefore, in total, the running time to solve Problem ILCS remains $O(\mathcal{R}\ell + \mathcal{R} \log \log n + n)$ in the worst case. □

*Remark 1.* LCS-II can compute the actual LCS in $O(\mathcal{L}(X, Y))$ time. However, in our adaptation of that algorithm for i-strings, we will need $O(\mathcal{L}(X, Y) \times \ell)$ time because we don't keep track of the matched character and therefore, are required to do the intersection operations to find a match. However, this can be reduced to $O(\mathcal{L}(X, Y))$ simply by keeping track of the matched character (at least one of them if there exists more) in the set $\mathcal{M}$.

## 4 Algorithm for CILCS

In this section, we present algorithms to solve Problem CILCS, i.e. Constrained LCS problem for i-strings. We follow the same strategy of Section 3: we use the best known dynamic programming algorithm for CLCS and try to devise an efficient algorithm for CILCS by doing some preprocessing. We use the dynamic programming formulation for CLCS presented in [2]. Extending our tabular notion from Equation 1, we use $\mathcal{T}[i, j, k], 1 \leq i, j \leq n, 0 \leq k \leq p$ to denote $\mathcal{L}_{Z[1..k]}(X[1..i], Y[1..j])$. We have the following formulation for Problem CLCS from [2].

$$\mathcal{T}[i, j, k] = \max\{\mathcal{T}'[i, j, k], \mathcal{T}''[i, j, k], \mathcal{T}[i, j-1, k], \mathcal{T}[i-1, j, k]\} \quad (4)$$

where

$$\mathcal{T}'[i,j,k] = \begin{cases} 1+\mathcal{T}[i-1,j-1,k-1] & \text{if}(k=1 \text{ or} \\ & (k>1 \text{ and } \mathcal{T}[i-1,j-1,k-1]>0)) \\ & \text{and } X[i]=Y[j]=Z[k]. \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

and

$$\mathcal{T}''[i,j,k] = \begin{cases} 1+\mathcal{T}[i-1,j-1,k] & \text{if } (k=0 \text{ or } \mathcal{T}[i-1,j-1,k]>0) \\ & \text{and } X[i]=Y[j]. \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The following boundary conditions are assumed in Equations 4 to 6:

$$\mathcal{T}[i,0,k] = \mathcal{T}[0,j,k] = 0,\ 0 \le i,j \le n, 0 \le k \le p.$$

It is straightforward to give an $O(n^2p)$ algorithm realizing the dynamic programming formulation presented in Equations 4 to 6. Now, for CILCS, we have to make the following changes. First of all, all equality checks of the form $X[i] = Y[j]$ have to be replaced by:

$$\widetilde{X}[i] \approx \widetilde{Y}[j]. \tag{7}$$

Here, the constant time operation is replaced by an $O(|\Sigma|)$ time operation in the worst case. On the other hand, all the triple equality checks of the form $X[i] = Y[j] = Z[k]$ have to be replaced by the check:

$$Z[k] \in \widetilde{X}[i] \bigcap \widetilde{Y}[j]. \tag{8}$$

Once again, the constant time operations are replaced by $O(|\Sigma| + \log|\Sigma|)$ time operations. So we have the following theorem.

**Theorem 5.** *Problem CILCS can be solved in $O(|\Sigma|n^2p)$ time.*

As before, our goal is to do some preprocessing to facilitate $O(1)$ time realization of the Check 7 and 8 and thereby improve the running time reported in Theorem 5.

### 4.1 Preprocessing 1 for CILCS

It is clear that we need the table $\mathcal{I}$ as defined in Section 3.1 for constant time realization of Check 7. In addition to that, to realize Check 8 in constant time, we compute two more tables $\mathcal{B}_{\widetilde{X}}[i,k]$, for $1 \le i \le n, 1 \le k \le p$ and $\mathcal{B}_{\widetilde{Y}}[j,k], 1 \le j \le n, 1 \le k \le p$, as defined below:

$$\mathcal{B}_{\widetilde{X}}[i,k] = \begin{cases} 1 & \text{If } Z[k] \in \widetilde{X}[j] \\ 0 & \text{Otherwise.} \end{cases} \tag{9}$$

$$\mathcal{B}_{\widetilde{Y}}[j,k] = \begin{cases} 1 & \text{If } Z[k] \in \widetilde{Y}[j] \\ 0 & \text{Otherwise.} \end{cases} \tag{10}$$

It is easy to realize that Check 8 evaluates to be true if, and only if, we have $\mathcal{B}_{\widetilde{X}}[i,k] = \mathcal{B}_{\widetilde{Y}}[j,k] = \mathcal{I}[i,j] = 1$. Therefore, with all three tables pre-computed, we can evaluate Check 8 in constant time. We have already discussed the construction of table $\mathcal{I}$ in Section 3.1. The other two tables can be computed exactly in the same way requiring $O(np|\Sigma|^{0.376})$ time each. Note that $p \leq n$. So we have:

**Theorem 6.** *Problem CILCS can be solved in $O(n|\Sigma| + n^2|\Sigma|^{0.376} + n^2p)$ time.*

**Theorem 7.** *Problem CILCS can be solved in $O(n|\Sigma| + n^2|\Sigma|/\log^2|\Sigma|) + n^2p)$ time.*

However, instead of applying the complex matrix multiplication algorithm to compute $\mathcal{B}_{\widetilde{X}}$ and $\mathcal{B}_{\widetilde{Y}}$ we can do it more simply by employing the set-membership check for each entry in the $n \times p$ table. This would require $O(np \log|\Sigma|)$ time to compute $\mathcal{B}_{\widetilde{X}}$ and $\mathcal{B}_{\widetilde{Y}}$. However, the overall asymptotic running time remains unimproved.

### 4.2 Preprocessing 2 for CILCS

In this section, we adapt the preprocessing of Section 3.2 to solve Problem CILCS. We first define the set of all 'triple' matches, $\mathcal{M}_3$, as follows:

$$\mathcal{M}_3 = \{(i,j,k) \mid \widetilde{X}[i] \approx \widetilde{Y}[j] \bigwedge Z[k] \in \widetilde{X}[i] \bigcap \widetilde{Y}[j], 1 \leq i,j \leq n, 1 \leq k \leq p\}.$$

We assume that $\mathcal{R}_3 = |\mathcal{M}_3|$. Now our goal is to compute the table $\mathcal{I}_3[i,j,k], 1 \leq i,j \leq n, 1 \leq k \leq p$ as defined below:

$$\mathcal{I}_3[i,j,k] = \begin{cases} 1 & \text{If } Z[k] \in \widetilde{X}[i] \bigcap \widetilde{Y}[j] \\ 0 & \text{Otherwise.} \end{cases} \tag{11}$$

It is easy to realize that, with table $\mathcal{I}$ and $\mathcal{I}_3$, we can evaluate, respectively, Check 7 and 8 in constant time each. And it is quite straightforward to adapt Algorithm 1 to compute $\mathcal{I}_3$ (please see Algorithm 2). All we need to do is to construct lists $L_Z[a]$, (similar to $L_{\widetilde{X}}[a]$ and $L_{\widetilde{Y}}[a]$) for each symbol $a \in \Sigma$ and incorporate it in the for loop of Step 9. The preprocessing time to compute $\mathcal{I}_3$ can be easily deduced following the analysis of Algorithm 1. To compute $\mathcal{I}_3$ we need to create $3 * |\Sigma|$ lists. These can be constructed by simply scanning $\widetilde{X}$, $\widetilde{Y}$ and $Z$ in turn requiring in total $O(2 \times n\ell + n) = O(n\ell)$ time in the worst case. The initialization of $\mathcal{I}_3$ requires $O(n^2p)$ time. The filling up of $\mathcal{I}_3$ requires $O(\mathcal{R}_3\ell)$ time. Note that we also need to compute $\mathcal{I}$. Thus the total running time required for the preprocessing is $O(n\ell + n^2 + n^2p + \mathcal{R}\ell + \mathcal{R}_3\ell) = O(n\ell + n^2p + \ell(\mathcal{R} + \mathcal{R}_3))$. Since, we already have an $n^2p$ component in the above running time, the total running time for the CILCS problem remains same as above.

**Algorithm 2** Computation of the Table $\mathcal{I}_3$

```
1: for a ∈ Σ do
2:     Insert the positions of a in X̃ in L_X̃[a] in sorted order
3:     Insert the positions of a in Ỹ in L_Ỹ[a] in sorted order
4:     Insert the positions of a in Z in L_Z[a] in sorted order
5: for i = 1 to n do
6:     for j = 1 to n do
7:         for k = 1 to p do
8:             I[i, j, k] = 0.
9: for a ∈ Σ do
10:    for i ∈ L_X̃[a] do
11:        for j ∈ L_Ỹ[a] do
12:            for k ∈ L_Z[a] do
13:                I[i, j, k] = 1.
```

**Theorem 8.** *Problem CILCS can be solved in* $O(n\ell + n^2p + \ell(\mathcal{R} + \mathcal{R}_3))$ *time.*

We remind that the remarks in Section 3.2, regarding $\ell$ and $\mathcal{R}$, applies here as well. We further remark that, in the worst case we have $\mathcal{R}_3 = n^2p$. But in many practical cases $\mathcal{R}_3$ may turn out to be $o(n^2p)$. Also, since $\ell$ can be much smaller than $|\Sigma|$ in many cases, $\mathcal{R}_3\ell$ remains as a 'pessimistic' upper bound.

## 5 Conclusion

In this paper, we have studied the classic and well-studied longest common subsequence (LCS) problem and a recent variant of it namely the constrained LCS (CLCS) problem, when the inputs are i-strings. In LCS, given two strings, we want to find the common subsequence having the highest length; in CLCS, in addition to that, the solution to the problem must also be a supersequence of a third given string. We have presented efficient algorithms to solve both LCS and CLCS for i-strings. In particular, we have used some novel techniques to preprocess the given strings, which lets us use the corresponding DP solutions for normal string to get efficient solution for i-strings. It would be interesting to see how well the presented algorithms behave in practice and compare them among themselves on the basis of their practical performance.

## Acknowledgement

We would like to express our gratitude to the anonymous reviewers for their helpful comments and especially for pointing out that the similar techniques used in [12, 23] to solve LCS have also been used in [19] to get efficient algorithms.

## References


1. V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev. On economic construction of the transitive closure of a directed graph (english translation). *Soviet Math. Dokl.*, 11:1209–1210, 1975.
2. A. N. Arslan and Ö. Eğecioğlu. Algorithms for the constrained longest common subsequence problems. *Int. J. Found. Comput. Sci.*, 16(6):1099–1109, 2005.
3. M. D. Atkinson and N. Santoro. A practical algorithm for boolean matrix multiplication. *Inf. Process. Lett.*, 29(1):37–38, 1988.
4. J. Basch, S. Khanna, and R. Motwani. On diameter verification and boolean matrix multiplication. *Technical Report, Department of Computer Science, Stanford University*, (STAN-CS-95-1544), 1995.
5. L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *String Processing and Information Retrieval (SPIRE)*, pages 39–48. IEEE Computer Society, 2000.
6. H. Broersma, S. S. Dantchev, M. Johnson, and S. Szeider, editors. *Algorithms and Complexity in Durham 2007 - Proceedings of the Third ACiD Workshop, 17-19 September 2007, Durham, UK*, volume 9 of *Texts in Algorithmics*. King's College, London, 2007.
7. F. Y. L. Chin, A. D. Santis, A. L. Ferrara, N. L. Ho, and S. K. Kim. A simple algorithm for the constrained sequence problems. *Inf. Process. Lett.*, 90(4):175–179, 2004.
8. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.
9. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1989.
10. M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. In *Symposium of Discrete Algorithms (SODA)*, pages 679–688, 2002.
11. J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest subsequences. *Commun. ACM*, 20(5):350–353, 1977.
12. C. S. Iliopoulos and M. S. Rahman. Algorithms for computing variants of the longest common subsequence problem. *Theoretical Computer Science*, 2007. To Appear.
13. C. S. Iliopoulos and M. S. Rahman. New efficient algorithms for LCS and constrained LCS problem. In Broersma et al. [6], pages 83–94.
14. C. S. Iliopoulos, M. S. Rahman, M. Voracek, and L. Vagner. Computing constrained longest common subsequence for degenerate strings using finite automata. In Broersma et al. [6], pages 95–106.
15. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal of Computing*, 24(5):1122–1139, 1995.
16. V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Problems in Information Transmission*, 1:8–17, 1965.
17. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.
18. W. J. Masek and M. Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.
19. V. Mkinen, G. Navarro, and E. Ukkonen. Transposition invariant string matching. *Journal of Algorithms*, 56:124–153, 2005.

20. E. W. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.
21. N. Nakatsu, Y. Kambayashi, and S. Yajima. A longest common subsequence algorithm suitable for similar text strings. *Acta Inf.*, 18:171–179, 1982.
22. M. S. Rahman and C. S. Iliopoulos. Algorithms for computing variants of the longest common subsequence problem. In T. Asano, editor, *ISAAC*, volume 4288 of *Lecture Notes in Computer Science*, pages 399–408. Springer, 2006.
23. M. S. Rahman and C. S. Iliopoulos. A new efficient algorithm for computing the longest common subsequence. In M.-Y. Kao and X.-Y. Li, editors, *AAIM*, volume 4508 of *Lecture Notes in Computer Science*, pages 82–90. Springer, 2007.
24. W. Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control*, 67(1-3):12–22, 1985.
25. Y.-T. Tsai. The constrained longest common subsequence problem. *Inf. Process. Lett.*, 88(4):173–176, 2003.
26. P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6:80–82, 1977.
27. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.
28. R. Williams. Matrix-vector multiplication in sub-quadratic time (some preprocessing required). In *SODA*, pages 1–11. ACM Press, 2007.

# Three NP-Complete Optimization Problems in Seidel's Switching

Eva Jelínková

Department of Applied Mathematics
Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
eva@kam.mff.cuni.cz

**Abstract.** Seidel's switching is a graph operation which makes a given vertex adjacent to precisely those vertices to which it was non-adjacent before, while keeping the rest of the graph unchanged. Two graphs are called switching-equivalent if one can be made isomorphic to the other by a sequence of switches.
In this paper, we show the NP-completeness of the problem Switch-$cn$-Clique for each $c \in (0, 1)$: determine if a graph $G$ is switching-equivalent to a graph containing a clique of size at least $cn$, where $n$ is the number of vertices of $G$. We also prove the NP-completeness of the problems Switch-Max-Edges and Switch-Min-Edges which decide if a given graph is switching-equivalent to a graph having at least or at most a given number of edges, respectively.

## 1 Introduction

The concept of Seidel's switching was introduced by a Dutch mathematician J. J. Seidel in connection with algebraic structures, such as systems of equiangular lines, strongly regular graphs, or the so-called two-graphs. For more structural properties of two-graphs, cf. [11–13]. Since then, switching has been studied by many others. Apart from the algebraic structures, consequences of switching arise in other research fields as well; for example, Seidel's switching plays an important role in Hayward's polynomial-time algorithm for solving the $P_3$-structure recognition problem [6].

As proved by Colbourn and Corneil [4] (and independently by Kratochvíl et al. [9]), deciding whether two given graphs are switching equivalent is an isomorphism-complete problem.

In this paper, we prove the NP-completeness of several problems related to Seidel's switching. We examine the complexity of deciding if a given graph is switching-equivalent to a graph having a certain desired property. This paradigm has already been addressed by several authors. As observed by Kratochvíl et al. [9] and also by Ehrenfeucht at al. [5], there is no correlation between the complexity of the problem and the complexity of the property $P$ itself. For example, Kratochvíl et al. [9] proved that any graph is switching-equivalent to a graph containing a Hamiltonian path, and it is polynomial to decide if a graph

is switching-equivalent to a graph containing a Hamiltonian cycle. However, the problems to decide if a graph itself contains a Hamiltonian path or cycle are well known to be NP-complete [7].

On the other hand, the problem of deciding switching-equivalence to a regular graph was proven to be NP-complete by Kratochvíl [10], and switching-equivalence to a $k$-regular graph for a fixed $k$ is polynomial, while both the regularity and $k$-regularity of a graph can be tested polynomially. Three-colorability and switching-equivalence to a three-colorable graph are both NP-complete [5].

One of the problems we address in this paper is deciding switching-equivalence to a graph containing a clique of a certain size. It is well known that deciding for instances $(G, k)$ if the graph $G$ itself contains a clique of size at least $k$ is NP-complete [7]. The corresponding switching problem, to decide for instances $(G, k)$ if $G$ is switching-equivalent to a graph with a clique of size at least $k$, was shown by Ehrenfeucht et al. [5] to be NP-complete as well. If $k$ is fixed (not part of the input), then the problem can be solved by testing all induced subgraphs of size $k$. The whole graph $G$ is switching-equivalent to a graph with a $k$-clique if and only if at least one induced subgraph of $G$ on $k$ vertices is switching-equivalent to a clique, and that can be determined in polynomial time. In Section 3 we extend the results of Ehrenfeucht et al. [5] by proving the NP-completeness of deciding switching-equivalence to a graph with a clique of size at least $cn$, where $n$ is the number of vertices of $G$, for every fixed constant $c$ in $(0, 1)$.

We further examine the complexity of problems Switch-Min-Edges and Switch-Max-Edges which for instances $(G, k)$ decide if $G$ is switching-equivalent to a graph with at most or at least $k$ edges, respectively. We prove that both problems are NP-complete. Such a result may be unexpected, because switching a vertex affects the number of edges in a simple way.

On the other hand, Suchý [14] recently proved that the problems Switch-Min-Edges and Switch-Max-Edges are fixed-parameter tractable. Hence, for fixed $k$ they are polynomial, which complements our result.

This paper is organized as follows. In Section 2, we introduce the notation and definitions used throughout the paper. In Section 3, we prove the NP-completeness of Switch-$cn$-Clique. In Section 4 we prove the NP-completeness of Switch-Min-Edges and Switch-Max-Edges, and describe a connection of these problems to graph theoretic codes and the Maximum Likelihood Decoding problem.

## 2 Basic Definitions

### 2.1 Preliminaries

In this paper, we use the standard graph theoretic notation. Unless defined otherwise, by $n$ we denote the number of vertices of the currently discussed graph. The graph $G = (V, \binom{V}{2})$ is called a *complete graph* and denoted by $K_n$. A complete subgraph on $k$ vertices is called a *$k$-clique*. A path with $n$ vertices is denoted by $P_n$, and a graph with $n$ vertices and no edges is called *discrete* and denoted by $I_n$. The symmetric difference of sets $A$ and $B$ is denoted by $A \triangle B$.

### 2.2 Seidel's Switching

**Definition 1.** *Let $G$ be a graph.* Seidel's switch of a vertex $v \in V_G$ *results in the graph called $S(G, v)$ whose vertex set is the same as of $G$ and the edge set is the symmetric difference of $E_G$ and the full star centered in $v$, i. e.,*

$$V_{S(G,v)} = V_G$$

$$E_{S(G,v)} = E_G \setminus \{xv : x \in V_G,\ xv \in E_G\}) \cup \{xv : x \in V_G,\ x \neq v,\ xv \notin E_G\}.$$

It is easy to observe that the result of a sequence of vertex switches in $G$ depends only on the parity of the number of times each vertex is switched. This allows generalizing switching to vertex subsets of $G$.

**Definition 2.** *Let $G$ be a graph. Then the* Seidel's switch of a vertex subset $A \subseteq V_G$ *is called $S(G, A)$ and*

$$S(G, A) = (V_G, E_G \triangle \{xy : x \in A,\ y \in V_G \setminus A\}).$$

*We say that two graphs $G$ and $H$ are* switching equivalent *(denoted by $G \sim H$) if there is a set $A \subseteq V_G$ such that $S(G, A)$ is isomorphic to $H$.*

## 3 Searching for a Switch with a *cn*-Clique

In this section, we consider the following problem.

**Problem:** SWITCH-*cn*-CLIQUE
**Input:** A graph $G$ on $n$ vertices
**Question:** Is $G$ switching-equivalent to a graph containing a clique of size at least $cn$?

**Theorem 1.** *The problem* SWITCH-*cn*-CLIQUE *is NP-complete for any $c \in (0, 1)$.*

*Proof.* We prove the theorem in two steps: first we prove the statement for rational numbers $c$ only; then we extend it to numbers $c$ which are irrational. For rational $c$, it is clear that the problem is in NP—a polynomial-size certificate contains vertex subsets $A$ and $C$ such that $S(G, A)[C]$ is a clique of the desired size. In the case of irrational $c$, we assume that an oracle can be used for querying the bits of $c$ in constant time. This ensures that the certificate can be checked in time polynomial in $n$.

In the first step, we show the NP-hardness of the problem by reducing SAT to it, whereas in the second step we reduce 3-SAT. Both SAT and 3-SAT are well known to be NP-complete [7].

Suppose that $c$ is rational and equal to $\frac{p}{q}$, where $p, q \in \mathbb{N}$, and $p < q$. We have an instance of SAT: a formula $\varphi$ in CNF with $k$ clauses and $l$ occurrences of literals, and ask if $\varphi$ is satisfiable. Without loss of generality we can assume that $k < l$ and $k \geq 2$.

Let $G = G_{p,q}(\varphi)$ be a graph constructed in the way illustrated in Fig. 1. The vertices of $G$ are $V_G = L \cup K \cup Z$, where $L$, $K$, $Z$ are pairwise disjoint and

$$|L| = l,$$

$$|K| = pl + p - k,$$

$$|Z| = (q - p - 1)(l + 1) + k + 1.$$

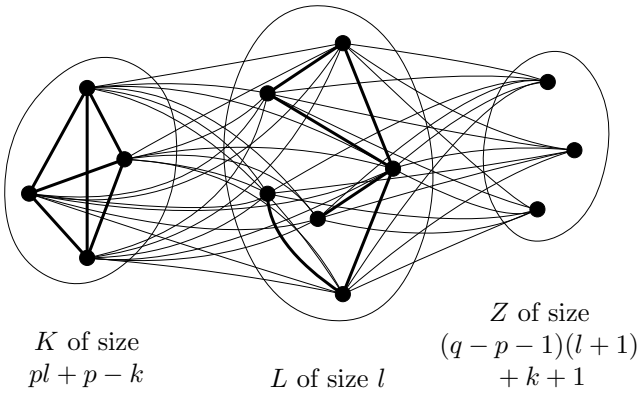



**Fig. 1.** The graph $G_{p,q}(\varphi)$.

The edges of $G$ are defined as follows:

- $K$ induces a clique and every vertex in $K$ is adjacent to all vertices in $L$ and no vertex in $Z$.
- Every vertex in $Z$ is adjacent to all vertices in $L$ and nothing more.
- Vertices of $L$ represent occurrences of literals in $\varphi$. Two vertices $l_1, l_2 \in L$ are adjacent if and only if
  - $l_1$ and $l_2$ occur in different clauses and
  - they are not in the form $l_1 = \neg l_2$ nor $l_2 = \neg l_1$.

**Lemma 1.** *Let $j$ be an integer. The graph $G_{p,q}(\varphi)[L]$ contains a $j$-clique if and only if the formula $\varphi$ contains $j$ simultaneously satisfiable clauses.*

*Proof.* Mutually adjacent vertices of $G_{p,q}(\varphi)[L]$ correspond to simultaneously satisfiable literals in distinct clauses. □

**Corollary 1.** *The formula $\varphi$ is satisfiable if and only if $G_{p,q}(\varphi)[L]$ contains a clique of size $k$ (where $k$ is the number of clauses in $\varphi$).*

Let us now consider cliques of size $pl + p$ in the whole graph—either in the original graph $G$ or in its switches. The reader can verify that $n = |L \cup K \cup Z| = ql + q$ and $(pl + p)/n = c$, therefore cliques of size $pl + p$ are exactly $cn$-cliques.

**Lemma 2.** *The following statements are equivalent for $G = G_{p,q}(\varphi)$.*

*(a) The graph $G[L]$ contains a $k$-clique.*
*(b) The graph $G$ contains a $(pl+p)$-clique.*
*(c) There exists a set $A \subseteq V_G$ such that $S(G,A)$ contains a $(pl+p)$-clique.*

*Proof.* First we prove that (a) implies (b). Any clique in $G[L]$ forms a larger clique together with all vertices of $K$. So, if $G[L]$ contains a $k$-clique, then $G[L \cup K]$ contains a clique of size $k + (pl + p - k) = pl + p$.

The implication from (b) to (c) is obvious.

To prove that (c) implies (a), suppose that there is a set $A \subseteq V_G$ such that $S(G,A)$ contains a $(pl+p)$-clique on a vertex set $C$.

The set $C$ does not contain more than two vertices of $Z$, because they are pairwise non-adjacent in $G$ and in $S(G,A)$ they induce a bipartite graph. From the assumptions $k < l$ and $k \geq 2$ it follows that $l > 2$, and $p \geq 1$, so $pl + p > 2$. Therefore $C$ contains some vertices of $L$ or $K$. But all vertices of $Z$ are non-adjacent in $G$ and have the same neighborhood in $G[L \cup K]$; surely all vertices in $Z \cap C$ have the same neighborhood in $S(G,A)[C]$ (otherwise $C$ would not induce a clique). But then either $(Z \cap C) \subseteq A$ or $(Z \cap C) \cap A = \emptyset$, so switching $A$ does not affect edges inside $S(G,A)[Z \cap C]$ and any two vertices in $S(G,A)[Z \cap C]$ are non-adjacent. Therefore $C$ contains at most one vertex of $Z$.

Since $1 + |K| = 1 + (pl + p - k) < pl + p$, the clique $C$ contains at least one vertex of $L$. But then it cannot contain both vertices of $K$ and $Z$, because in the graph $G$ they have the same neighborhood in $L$ and there is no edge between $K$ and $Z$. Also, the set $C$ cannot consist only of vertices of $L$, because $pl + p > l$. Therefore $C$ contains one of the following:

- $pl + p - 1$ (which is at least $k$) vertices of $L$ and one vertex of $Z$
- at least $k$ vertices of $L$ and at least one vertex of $K$.

In both cases, $C$ contains $k$ vertices of $L$, and a vertex $v$ of $K \cup Z$. Since $C$ induces a clique in $S(G,A)$, the vertex $v$ is adjacent to all other vertices in $C$. But in $G$, by definition, the vertex $v$ is adjacent to all vertices of $L$, too. So switching $A$ cannot have changed any edge connecting $v$ and the $k$ vertices, which means that either all these $k+1$ vertices are in $A$ or none of them is. But then they induce a $(k+1)$-clique in $G$ as well, and $G[L]$ contains a $k$-clique, which we wanted to prove. □

Corollary 1 and Lemma 2 together give us that $\varphi$ is satisfiable if and only if there exists a set $A \subseteq V_G$ such that $S(G,A)$ contains a $(pl+p)$-clique. But we have already shown that $pl + p = cn$; and clearly a graph contains a clique of size *exactly* $cn$ if and only if it contains a clique of size *at least* $cn$. That concludes the reduction. The graph $G_{p,q}(\varphi)$ with $q(l+1) = \mathcal{O}(l)$ vertices and $\mathcal{O}(l^2)$ edges can be constructed in time polynomial in the size of $\varphi$. Hence the problem Switch-$cn$-Clique is NP-complete for every rational constant $c \in (0,1)$.

Proving the NP-hardness of Switch-$cn$-Clique for irrational numbers $c$ is slightly more complicated. We use a theorem of Arora et al. [1] and certain

number theoretic results to get suitable numbers $p, q$, and then, analogously to the rational case, we reduce an instance of 3-SAT to Switch-$cn$-Clique for the graph $G_{p,q}$. Due to space limitations, the rest of the proof is placed in the Appendix. □

## 4 Minimizing the Number of Edges

In this section, we prove the NP-completeness of the following problem:

**Problem:** Switch-Min-Edges
**Input:** A graph $G$, an integer $k$.
**Question:** Is $G$ switching-equivalent to a graph with at most $k$ edges?

The problem Switch-Max-Edges is defined analogously with "at most" replaced by "at least". It is easy to observe that these two problems are polynomially equivalent. Therefore it suffices to show the NP-completeness of Switch-Min-Edges only, which is done in the proof of Theorem 3.

### 4.1 The Connection to Maximum Likelihood Decoding

Let $V$ be a fixed set of $n$ vertices. For an edge set $E \subseteq \binom{V}{2}$, by $\chi_E$ we denote the characteristic vector of $E$, i. e., the element of $\mathbb{Z}_2^{\binom{n}{2}}$ such that $\chi_E(e) = 1$ if and only if $e \in E$. Thus any graph on the vertex set $V$ can be represented by a vector of length $\binom{n}{2}$. The following observation expresses how switching works by means of characteristic vectors.

**Observation 2** *Let $K_n = (V, \binom{V}{2})$ be the complete graph, let $V_1$, $V_2$ be a partition of $V$ and let $S = \{\{x, y\}, x \in V_1,\ y \in V_2\}$ be the corresponding cut in $K_n$. Then for any $G = (V, E)$,*

$$\chi_{S(G,V_1)} = \chi_{S(G,V_2)} = \chi_E + \chi_S.$$

(Note that the summation is done over $\mathbb{Z}_2$.) Therefore, if we seek a switch of $G$ with the minimum number of edges, we seek a characteristic vector $\chi_E + \chi_S$ with the minimum Hamming weight. Or, equivalently, we seek a cut $S$ in $K_n$ with the minimum Hamming distance between $\chi_S$ and $\chi_E$.

It is a well-known fact that the cut space $\mathcal{C}^*(G)$ of a graph $G$ is a vector space, and the cycle space $\mathcal{C}(G)$ is also a vector space orthogonal to $\mathcal{C}^*(G)$. The dimension of $\mathcal{C}^*(G)$ is $|V| - 1$, and $\mathcal{C}^*(G)$ can also be viewed as a linear $[|E|, |V| - 1]$ code with a parity-check matrix $C$ whose rows are $|E| - |V| + 1$ linearly independent characteristic vectors of cycles in $G$. Such a code is called a *graph theoretic code*; the concept of graph theoretic codes has been introduced by Hakimi and Frank [8].

The problem of finding a codeword in a linear code that is closest to a given vector is an important problem in coding theory. It can be formulated as a decision problem in the following way.

**Problem:** Maximum Likelihood Decoding
**Input:** A binary $p \times q$ matrix $H$, a vector $r \in \mathbb{Z}_2^p$, and an integer $w > 0$.
**Question:** Is there a vector $e \in \mathbb{Z}_2^q$ of Hamming weight at most $w$ such that $He = r$?

This problem was proven to be NP-complete by Berlekamp et al. [2]. Note that Switch-Min-Edges is indeed its special case, which we formalize in the following lemma (proved in the Appendix).

**Lemma 3.** Switch-Min-Edges *is a special case of* Maximum Likelihood Decoding, *where $H$ is the parity check matrix of the code of cuts in a complete graph.*

Special cases of Maximum Likelihood Decoding have been studied. It is known that the problem is NP-complete even if we allow unbounded time for preprocessing the code $H$. This was proven by Bruck and Naor [3] by showing that Maximum Likelihood Decoding is NP-complete for the cut code of a special fixed graph, and therefore no preprocessing can help because this fixed code can be known in advance. Our proof in Subsection 4.2 provides an alternative proof of Bruck and Naor's result by using $K_n$ as the fixed graph.

### 4.2 Proof of NP-Completeness

We use a reduction of the following well-known NP-complete problem [7].

**Problem:** Simple-Max-Cut
**Input:** A graph $G$, an integer $j$.
**Question:** Does there exist a partition $V_1$, $V_2$ of $V_G$ such that the cut between $V_1$ and $V_2$ in $G$ contains at least $j$ edges?

**Theorem 3.** Switch-Min-Edges *is NP-complete.*

*Proof.* From an instance $(G, j)$ of Simple-Max-Cut we create an instance $(G', k)$ of Switch-Min-Edges in the following way. For each vertex of $G$ we create a corresponding non-adjacent vertex pair in $G'$. An edge in $G$ is represented by four edges completely interconnecting the two pairs, and a non-edge in $G$ is represented by only two edges connecting the two pairs in a parallel way. More formally, we set

$$V_{G'} = \{v', v'' : v \in V_G\}$$

$$E_{G'} = \{\{u', v'\}, \{u'', v''\} : u, v \in V_G, u \neq v\} \cup \{\{u', v''\}, \{u'', v'\} : \{u, v\} \in E_G\}.$$

The following lemma relates cuts in $G$ with switches of $G'$.

**Lemma 4.** *The following statements are equivalent:*

*(a) There is a cut in $G$ having at least $j$ edges,*
*(b) there exists a set $A \subseteq V_{G'}$ such that $S(G', A)$ contains at most $2\binom{|V_G|}{2} + 2|E_G| - 4j$ edges.*

*Proof.* For a cut $C$ in $G$ we define a corresponding vertex subset $A = A(C)$ of $G'$. Suppose that $C$ separates a vertex set $V_1$ from the remaining vertices of $G$. We set $A$ to be the set $\{v', v'' : v \in V_1\}$. Note that such a set $A$ satisfies the following condition.

**Definition 3.** *We say that a vertex subset of $G'$ is* legal *if it contains an even number of vertices out of the pair $v', v''$ for every $v \in V_G$. Otherwise, we say that it is* illegal.

Legality is a desired property, because there is an obvious correspondence between legal sets and cuts in $G$. Also, the number of edges in $S(G', A)$ is determined by the size of the cut $C$. The original graph $G'$ contains two edges per every vertex-pair of $G$ and two more edges per every edge in $G$, which is $2\binom{|V_G|}{2} + 2|E_G|$ edges altogether. Since $A$ is legal, it can easily be checked that every non-edge $\{u, v\}$ in $G$ corresponds to two edges in both $G'$ and $S(G', A)$, regardless of the cut $C$. For every edge $\{u, v\}$ in the cut $C$ we have $u'$, $u'' \in A$ and $v'$, $v'' \notin A$ (or vice versa), so switching $A$ destroys all the four corresponding edges and creates none. For an edge not present in the cut $C$, switching $A$ does not modify the corresponding edges, so there are still four of them in $S(G', A)$. To sum it up, $S(G', A)$ has

$$2\binom{|V_G|}{2} + 2|E_G| - 4|C|$$

edges. This proves that the statement (a) implies (b). As for the other implication, by reverting the construction of $A(C)$ from a cut $C$, we get that it holds for legal sets $A$. It remains to deal with possible illegal switches. For that purpose, we introduce another definition.

**Definition 4.** *We say that a vertex $u \in V_G$ is* broken in $A$ *if $A$ contains exactly one vertex of $u', u''$. We say that a vertex set $\{u, v\} \subseteq V_G$ is* broken in $A$ *if at least one of its vertices $u$, $v$ is broken. Otherwise, we say that it is* legal in $A$.

**Lemma 5.** *For every illegal set $A$ there is a legal set $A'$ such that $S(G', A')$ contains at most the same number of edges as $S(G', A)$.*

*Proof.* Let $A$ be an illegal set. As can be seen in Fig. 2, a broken non-edge never decreases the resulting number of edges, and thus is no more profitable than a legal non-edge. Therefore, if all the broken pairs in $A$ correspond to non-edges, then the set $A$ minus the union of all broken pairs is legal, and it yields a switch with at most the same number of edges as $A$ does.

Assume that there are $m$ broken *edges*, where $m > 0$. As shown in Fig. 2, a broken edge could in certain cases decrease the number of edges in $G'$ by more than a legal edge not present in the cut would. We create a legal set $A'$ from $A$ using the following greedy algorithm.

1. $A' := A$.

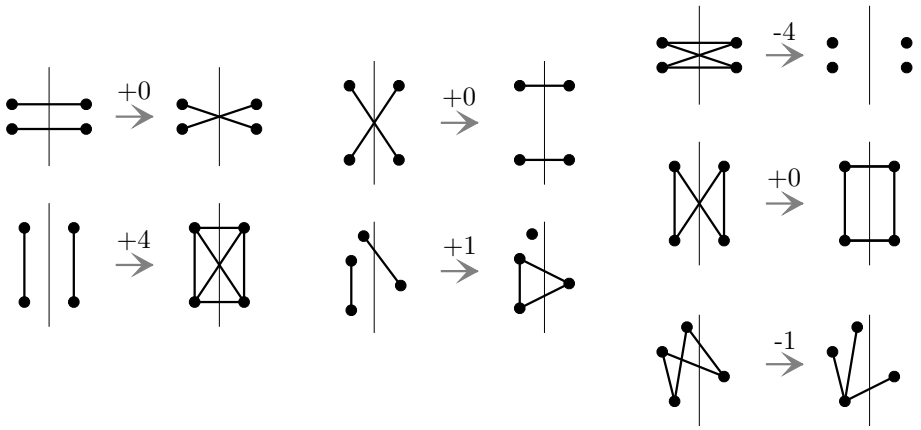



**Fig. 2.** The possible legal and illegal switches of non-edges (on the left, in the middle) and edges (on the right) in $G$ and their influence on the number of edges in $G'$.

2. Find a vertex $v$ broken in $A'$. If there is no such, STOP.
3. Look for vertices $u$ that are legal in $A'$ and such that $\{u, v\}$ is an edge in $G$. If there are more such vertices in $A'$ than in $V_G \setminus A'$, set $A' := A' \setminus \{v', v''\}$, otherwise set $A' := A' \cup \{v', v''\}$.
4. Go back to step 2.

It remains to prove that the algorithm finds a legal set that is better than $A$.

In each iteration of step 3, the algorithm legalizes one vertex. It clearly finishes after a finite number of steps and creates a legal set. It does not modify legal vertices nor legal edges in $A$, therefore the number of edges in $S(G', A')$ corresponding to legal sets in $A$ remains unchanged. Also, legalizing a non-edge does not increase the number of edges in $S(G', A')$ in comparison to $S(G', A)$.

As we already know, any legal set gives us a cut in $G$; consider the cut obtained from $A'$. In each iteration of Step 3, at least half of the newly legalized edges became cut edges with one endpoint in $A'$ and the other not in $A'$. Since each broken edge was legalized exactly once, we know that at least $m/2$ legalized edges became cut edges (and decreased the edge number by at least 3), and at most $m/2$ legalized edges became out of the cut (and increased the edge number by at most 1). Therefore, the edge number in $S(G', A')$ is lower by at least $3m/2 - m/2 = m$, which we assumed to be positive. □

To finish the proof of Lemma 4, it remains to prove that (b) implies (a). Let $S(G', A)$ be a switch with at most $2\binom{|V_G|}{2} + 2|E_G| - 4j$ edges. If $A$ is illegal, Lemma 5 assures that there is a legal set $A'$ such that $S(G', A')$ has even less edges. The legal set yields a partition $V_1 = \{v : v', v'' \in A'\}$ and $V_2 = V_G \setminus V_1$ such that the cut between $V_1$ and $V_2$ contains at least $j$ edges. □

According to Lemma 4, the graph $G$ contains a cut of size at least $j$ if and only if $G'$ is switching-equivalent to a graph with at most $k = 2\binom{|V_G|}{2} + 2|E_G| - 4j$

edges. The size of $(G', k)$ is surely polynomial in the size of $(G, j)$. That concludes the proof of NP-hardness of Switch-Min-Edges. To prove that Switch-Min-Edges is in NP, it suffices to note that the positive answer can be certified by a vertex set $A$ that gives us a switch with the desired number of edges. □

**Acknowledgments** I am grateful to Jan Kratochvíl for numerous useful tips and valuable discussions; and to Jiří Sgall for advice concerning some relevant references.

## References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy: *Proof verification and the hardness of approximation problems*, Journal of ACM **45(3)** (1998), pp. 501–555. An extended abstract appeared in FOCS 1992.
2. E. R. Berlekamp, R. J. McEliece, H. C. A. van Tilborg, *On the inherent intractability of certain coding problems*, IEEE Trans. Inform. Theory, vol **IT-24** (1978), pp. 384–386.
3. J. Bruck, M. Naor, *The hardness of decoding linear codes with preprocessing*, IEEE Trans. Inform. Theory, vol **36** (1990), pp. 381–384.
4. C. J. Colbourn, D. G. Corneil, *On deciding switching equivalence of graphs*, Discrete Appl. Math, vol **2** (1980), pp. 181–184.
5. A. Ehrenfeucht, J. Hage, T. Harju, G. Rozenberg: *Complexity issues in switching classes*, in: Theory and Application to Graph Transformations, LNCS 1764, Springer-Verlag (2000), pp. 59–70.
6. R. B. Hayward: *Recognizing $P_3$-structure: A switching approach*, J. Combin. Th. Ser. B **66** (1996), pp. 247–262.
7. M. R. Garey, D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.
8. S. L. Hakimi, H. Frank: *Cut-set matrices and linear codes*, IEEE Trans. Inform. Theory, vol. **IT-11** (1965), pp. 457–458.
9. J. Kratochvíl, J. Nešetřil, O. Zýka: *On the computational complexity of Seidel's switching*, Proc. 4th Czech. Symp., Prachatice 1990, Ann. Discrete Math. **51** (1992), pp. 161–166.
10. J. Kratochvíl: *Complexity of hypergraph coloring and Seidel's switching*, in: Graph Theoretic Concepts in Computer Science (H. Bodlaender, ed.), Proceedings WG 2003, Elspeet, June 2003, LNCS 2880, Springer Verlag (2003), pp. 297–308.
11. J. J. Seidel: *Graphs and two-graphs*, in: Proc. 5th. Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Winnipeg, Canada (1974).
12. J. J. Seidel: *A survey of two-graphs*, Teorie combinatorie, Atti Conv. Lincei, Vol 17, Academia Nazionale dei Lincei, Rome (1973), pp. 481–511.
13. J. J. Seidel, D. E. Taylor: *Two-graphs, a second survey*, Algebraic methods in graph theory, Vol. II, Conf. Szeged 1978, Colloq. Math. Janos Bolyai **25** (1981), pp. 689–711.
14. O. Suchý: *Some Parametrized Problems Related to Seidel's Switching*, to appear in Proceedings of IWOCA 2007.
15. V. V. Vazirani: *Approximation Algorithms*, Springer-Verlag, 2001.

## Appendix

*Proof (of Theorem 1, continued).* To prove the NP-hardness of Switch-*cn*-Clique for irrational numbers $c$, we use a theorem of Arora et al. proved in [1] (and restated in this way in [15] as Lemma 29.10).

**Theorem 4.** *(Arora, Lund, Motwani, Sudan, Szegedy) There exists a polynomial time transformation $T$ from 3-CNF to 3-CNF and a constant $\varepsilon > 0$ such that*

- *If $\psi$ is satisfiable, then $T(\psi)$ is satisfiable.*
- *If $\psi$ is not satisfiable, then at most $1 - \varepsilon$ fraction of the clauses of $T(\psi)$ are simultaneously satisfiable.*

Our aim is to do a reduction from an instance $\psi$ of 3-SAT. We will use the graph $G_{p,q}$, like in the previous part of the proof; this time for the transformed formula $T(\psi)$ and for numbers $p$ and $q$ such that $\frac{p}{q}$ is sufficiently close to the irrational number $c$. Then we examine the relationship between $cn$-cliques and $\frac{p}{q}$-cliques in the resulting graph. To show that some suitable numbers $p$ and $q$ exist, we will make use of Lemma 6, which is a variant of Dirichlet's Theorem, and Lemma 7.

**Lemma 6.** *For any real number $\alpha \in [0,1]$, any $\varepsilon > 0$ and $r \in \mathbb{R}$ there exists $n \in \mathbb{N}$ such that $n > r$ and $\{n\alpha\} < \varepsilon$ (where $\{n\alpha\}$ stands for the decimal fraction of $n\alpha$).*

*Proof.* Without loss of generality we can assume that $\varepsilon < \alpha$ and $\alpha \in (0,1)$. We prove by induction that for each $k \in \mathbb{N}_0$ there exists $n_k \in \mathbb{N}$ such that

$$\{n_k\alpha\} \leq \frac{\alpha}{2^k}$$

and $n_{k+1} > n_k$ for all $k$. Then we take $n = n_k$ for $k \geq \max\{r, \log_2(\frac{\alpha}{\varepsilon})\}$.

We set $n_0 = 1$, because $\{1 \cdot \alpha\} \leq \frac{\alpha}{1}$. Now assume that we already have $n_k$ for some $k \geq 0$ and want to find $n_{k+1}$. Let $\beta = \{n_k\alpha\}$; we want to get an integer $m$ so that $\{m\beta\} \leq \frac{\beta}{2}$ and $m > 1$. If $\beta = 0$, then clearly the inequality $\{m\beta\} \leq \frac{\beta}{2}$ holds for any integer $m$, so we can set $m = 2$. Otherwise we consider the number $\lfloor \frac{1}{\beta} \rfloor \beta$. It is clear that $\lfloor \frac{1}{\beta} \rfloor \beta \leq 1$; in case of an equality we have that $\{\lfloor \frac{1}{\beta} \rfloor \beta\} = 0$, while $\lfloor \frac{1}{\beta} \rfloor$ is nonzero. Hence $m$ can be either $\lfloor \frac{1}{\beta} \rfloor$ or any its integral multiple larger than 1.

The remaining case is that $\beta > 0$ and $\lfloor \frac{1}{\beta} \rfloor \beta < 1$. Then $1 < \lceil \frac{1}{\beta} \rceil \beta < \beta + 1$, and after subtracting 1 we get that

$$\left\{ \left\lceil \frac{1}{\beta} \right\rceil \beta \right\} < \beta. \tag{1}$$

We want $m$ to be an integer such that $\{m\beta\} \leq \frac{\beta}{2}$. Note that $\lceil \frac{1}{\beta} \rceil > 1$, since $\lfloor \frac{1}{\beta} \rfloor > 0$ for any $\beta \in (0,1)$. So suppose that $m$ cannot be $\lceil \frac{1}{\beta} \rceil$, because

$\{\lceil \frac{1}{\beta} \rceil \beta\} > \frac{\beta}{2}$. Then we define

$$\delta = \beta + 1 - \left\lceil \frac{1}{\beta} \right\rceil \beta.$$

The assumption $\{\lceil \frac{1}{\beta} \rceil \beta\} > \frac{\beta}{2}$ together with (1) imply that $\delta \in (0, \frac{\beta}{2})$. Similarly like before we obtain the inequalities $\lfloor \frac{\beta}{2\delta} \rfloor \delta \leq \frac{\beta}{2}$ and $\frac{\beta}{2} \leq \lceil \frac{\beta}{2\delta} \rceil \delta$. Moreover, it is surely true that $\lceil \frac{\beta}{2\delta} \rceil \leq (1 + \lfloor \frac{\beta}{2\delta} \rfloor)$; hence

$$\frac{\beta}{2} \leq \left\lceil \frac{\beta}{2\delta} \right\rceil \delta \leq \beta. \tag{2}$$

Now we set

$$m = \left\lceil \frac{\beta}{2\delta} \right\rceil \left( \left\lceil \frac{1}{\beta} \right\rceil - 1 \right) + 1.$$

It is clear that such an $m$ is larger than one, and by substituting $\delta$ according to its definition, it can be easily verified that

$$m\beta = \left\lceil \frac{\beta}{2\delta} \right\rceil - \left\lceil \frac{\beta}{2\delta} \right\rceil \delta + \beta. \tag{3}$$

By plugging the inequalities of (2) into (3), we obtain

$$\left\lceil \frac{\beta}{2\delta} \right\rceil \leq m\beta \leq \left\lceil \frac{\beta}{2\delta} \right\rceil + \frac{\beta}{2},$$

which immediately gives us that $\{m\beta\} \leq \frac{\beta}{2}$, and that is what we wanted. It now remains to set $n_{k+1} = mn_k$ and verify that $\{n_{k+1}\alpha\} \leq \frac{\alpha}{2^{k+1}}$. Indeed, we have that

$$\{n_{k+1}\alpha\} = \{mn_k\alpha\} = \{m\beta\} < \frac{\beta}{2} = \frac{\{n_k\alpha\}}{2} \leq \frac{\frac{\alpha}{2^k}}{2} = \frac{\alpha}{2^{k+1}},$$

where the last inequality holds by the induction hypothesis. In all cases considered we chose $m$ to be larger than one, hence $n_{k+1} > n_k$, and we are done. □

**Lemma 7.** *For each irrational $c \in (0,1)$ and $\varepsilon > 0$, there exist $p$, $q \in \mathbb{N}$ such that $\frac{p}{q} \in (0,1)$ and*

$$c \in \left( \left(1 - \frac{\varepsilon}{4p}\right) \frac{p}{q}, \frac{p}{q} \right).$$

*Proof.* We shall find an integer $p$ such that the interval $(\frac{p}{c} - \frac{\varepsilon}{4c}, \frac{p}{c})$ contains another integer $q$. We want $p$ to satisfy the condition

$$\left\{ \frac{p}{c} \right\} < \frac{\varepsilon}{4c}, \tag{4}$$

and additionally we request that

$$p > \frac{\varepsilon}{4(1-c)}. \tag{5}$$

It is true that $\{\frac{p}{c}\} = \{p\{\frac{1}{c}\}\}$, the number $\{\frac{1}{c}\}$ lies in the interval $(0,1)$, and surely $\frac{\varepsilon}{4c} > 0$; hence Lemma 6 for $\alpha = \{\frac{1}{c}\}$, $\varepsilon' = \frac{\varepsilon}{4c}$ and $r = \frac{\varepsilon}{4(1-c)}$ ensures the existence of such a $p$.

Then we set $q = \lfloor \frac{p}{c} \rfloor$ and verify that it is really an integer in the interval $\left(\frac{p}{c} - \frac{\varepsilon}{4c}, \frac{p}{c}\right)$. The number $\frac{p}{c}$ is irrational, so we have $q < \frac{p}{c}$. The fact that $\lfloor \frac{p}{c} \rfloor = \frac{p}{c} - \{p\{\frac{1}{c}\}\}$, and (4) together give us the other inequality $q > \frac{p}{c} - \frac{\varepsilon}{4c}$.

Moreover, from (5) we obtain

$$\frac{p}{c} - \frac{\varepsilon}{4c} > p,$$

so any integer $q$ in the interval $\left(\frac{p}{c} - \frac{\varepsilon}{4c}, \frac{p}{c}\right)$ is larger than $p$, and thus $\frac{p}{q} \in (0,1)$. Also, by rewriting the inequalities $q < \frac{p}{c}$ and $q > \frac{p}{c} - \frac{\varepsilon}{4c}$ we get the desired inequality

$$\left(1 - \frac{\varepsilon}{4p}\right)\frac{p}{q} < c < \frac{p}{q}.$$

□

Let $c$ be an irrational number in $(0,1)$, let $\varepsilon$ be the constant from Theorem 4, and $p$, $q$ the integers given by Lemma 7 for $\varepsilon$ and $c$. We take an instance $\psi$ of 3-SAT and construct the graph $G = G_{p,q}(T(\psi))$ in the same way as in the previous part of the proof. Let us again denote the number of clauses of $T(\psi)$ by $k$. The number of occurrences of literals is $l = 3k$ and $n$ stands for the number of vertices of $G$.

If $\psi$ is satisfiable, we have again by Corollary 1 that $G[L]$ contains a $k$-clique, and by Lemma 2 the graph $G$ contains a $(pl+p)$-clique, which is a $\frac{p}{q}n$-clique. We shall show that if $\psi$ is not satisfiable, then for any set $A \subseteq V_G$ the graph $S(G,A)$ does not contain a clique of size larger than $(1 - \frac{\varepsilon}{4p})\frac{p}{q}n$. We limit ourselves to instances $\psi$ such that $(1-\varepsilon)k > 1$ and $pl + p - \varepsilon k \geq 2$, which we can do without loss of generality. Let us first show the following lemma.

**Lemma 8.** *Let $\psi$ be a formula such that $(1-\varepsilon)k > 1$ and $pl + p - \varepsilon k \geq 2$. If $\psi$ is not satisfiable, then for any set $A \subseteq V_G$ the graph $S(G,A)$ does not contain a clique of size larger than $pl + p - \varepsilon k$.*

*Proof.* Suppose that for some $A \subseteq V_G$ we have a clique on a vertex set $C$ in $S(G,A)$ and the size of the clique is larger than $pl + p - \varepsilon k$. Then (similarly as in the proof of Lemma 2) we get that the set $C$ does not contain more than two vertices of $Z$, because they are pairwise non-adjacent in $G$ and in $S(G,A)$ they induce a bipartite graph.

Since $|C|$ is more than two, $C$ contains some vertices of $L$ or $K$. But all vertices of $Z$ are non-adjacent in $G$ and have the same neighborhood in $G[L \cup K]$; surely all vertices in $Z \cap C$ have the same neighborhood in $S(G,A)[C]$ (otherwise $C$ would not be a clique). But then either $(Z \cap C) \subseteq A$ or $(Z \cap C) \cap A = \emptyset$, so switching $A$ does not affect edges inside $S(G,A)[Z \cap C]$, and any two vertices in $S(G,A)[Z \cap C]$ are non-adjacent. Therefore $C$ contains at most one vertex of $Z$.

The set $C$ contains at least one vertex of $L$, because

$$1 + |K| = 1 + (pl + p - k) < (1 - \varepsilon)k + pl + p - k = pl + p - \varepsilon k.$$

But then $C$ cannot contain both vertices of $K$ and $Z$, because in $G$ they have the same neighborhood in $L$ and there is no edge between $K$ and $Z$.

By Lemma 1, every clique in $L$ corresponds to $|C \cap L|$ clauses which are simultaneously satisfiable. Hence by Theorem 4, the maximum clique size in $L$ is $(1 - \varepsilon)k$, which is not enough for $C$, since

$$(1 - \varepsilon)k < (1 - \varepsilon)k + pl + p - k = pl + p - \varepsilon k,$$

hence $C$ cannot consist only of vertices of $L$. Therefore $C$ consists of one of the following:

- more than $pl + p - \varepsilon k - 1$ (which is larger than $(1 - \varepsilon)k$) vertices of $L$, and one vertex of $Z$
- more than $(1 - \varepsilon)k$ vertices of $L$, and $pl + p - k$ vertices of $K$.

In both cases, $C$ contains more than $(1 - \varepsilon)k$ vertices of $L$, and a vertex $v$ from $K$ or $Z$. Since $C$ induces a clique in $S(G, A)$, the vertex $v$ is adjacent to all other vertices in $C$. But in $G$, by definition, $v$ is adjacent to all vertices of $L$, too. So switching $A$ cannot have changed any edge connecting $v$ and the other vertices, which means that either all the vertices are in $A$ or none of them is. But then they induce a clique of size larger than $(1 - \varepsilon)k$ in $G[L]$ as well. As we have already shown, the maximum clique size in $G[L]$ is $(1 - \varepsilon)k$, which is a contradiction. □

By Lemma 8, if $\psi$ is not satisfiable, then the maximum clique size in $S(G, A)$ for any $A$ is $pl + p - \varepsilon k$. But

$$\frac{\varepsilon k}{n} = \frac{\varepsilon k}{q(l+1)} = \frac{\varepsilon k}{q(3k+1)} \geq \frac{\varepsilon}{4q} = \frac{\varepsilon}{4p} \cdot \frac{p}{q},$$

so the maximum clique size divided by $n$ is

$$\frac{pl + p - \varepsilon k}{n} = \frac{p(l+1)}{q(l+1)} - \frac{\varepsilon k}{q(l+1)} \leq \frac{p}{q} - \frac{\varepsilon}{4p} \cdot \frac{p}{q} = \left(1 - \frac{\varepsilon}{4p}\right) \frac{p}{q}.$$

We have chosen the numbers $p$, $q$ so that

$$c \in \left( \left(1 - \frac{\varepsilon}{4p}\right) \frac{p}{q}, \frac{p}{q} \right),$$

hence the maximum clique ratio matches the lower bound of the interval containing $c$.

To sum it all up, we have shown that

- if $\psi$ is satisfiable, then there exists an $A \subseteq V_G$ such that $S(G, A)$ contains a clique of size $\frac{p}{q}n$, which is at least $cn$,

– if $\psi$ is not satisfiable, then for no set $A \subseteq V_G$ the graph $S(G, A)$ contains a clique of size more than $(1 - \frac{\varepsilon}{4p})\frac{p}{q}n$, especially of size at least $cn$.

Hence $\psi$ is satisfiable if and only if $G$ can be switched to contain a clique of size at least $cn$. The graph $G_{p,q}(T(\psi))$ with $q(l+1) = \mathcal{O}(l)$ vertices and $\mathcal{O}(l^2)$ edges can be constructed in polynomial time. That concludes the polynomial-time reduction of 3-SAT to Switch-$cn$-Clique for an irrational constant $c$, and also the proof that the problem is NP-complete. □

*Proof (of Lemma 3).* Having a graph $G$ with edge set $E$, we set $H = C(K_n)$ (the parity-check matrix of $\mathcal{C}^*(G)$), $r = H\chi_E$, and $w = k$. Then a vector $e$ is a solution of Maximum Likelihood Decoding if and only if $H(e + \chi_E) = \mathbf{0}$, which means that the vector $e + \chi_E$ is an element of the cut space $\mathcal{C}^*(G)$ and its Hamming distance from $\chi_E$ is at most $k$.

Therefore, by Observation 2 there exists a switch of $S(G, A)$ whose characteristic vector is $e$. Since the Hamming weight of $e$ is at most $k$, the switch $S(G, A)$ has at most $k$ edges and we are done.

# On Superconnectivity of $(4, g)$-Cages

Hongliang Lu[1], Yunjian Wu[1], Yuqing Lin[2], Qinglin Yu[1,3]
[1] Center for Combinatorics, LPMC
Nankai University, Tianjin, China
[2] School of Electrical Engineering and Computer Science
The University of Newcastle, Newcastle, Australia
[3] Department of Mathematics and Statistics
Thompson Rivers University, Kamloops, BC, Canada

**Abstract.** A $(k, g)$-*cage* is a graph that has the least number of vertices among all $k$-regular graphs with girth $g$. It was conjectured by Fu, Huang and Rodger [3] that all $(k, g)$-cages are $k$-connected for every $k \geq 3$. A $k$-connected graph $G$ is called *superconnected* if every $k$-cutset $S$ is trivial. Moreover, if $G - S$ has precisely two components, then $G$ is called *tightly superconnected*. In [9, 13], the authors showed that every $(4, g)$-cage is 4-connected. In this extended abstract, we proved that every $(4, g)$-cage is tightly superconnected when $g$ is odd.

*Key words:* cage, superconnected, tightly superconnected

## 1 Introduction

Throughout the paper, only undirected simple graphs are considered. Unless otherwise defined, we follow [1] for terminology and definitions.

Let $G = (V, E)$ be a graph with vertex set $V(G)$ and edge set $E(G)$. For $u, v \in V(G)$, $d_G(u, v)$ denotes the length of a shortest path in $G$. For vertex sets $T_1, T_2 \subseteq V(G)$, $E(T_1, T_2)$ is the set of the edges between $T_1$ and $T_2$, and $d(T_1, T_2) = d_G(T_1, T_2) = min\{d(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2\}$ denotes the *distance* between $T_1$ and $T_2$. For $S \subset V(G)$, $G - S$ is the subgraph of $G$ obtained by deleting the vertices in $S$ and all the edges incident with them. The set of vertices which are at distance $r$ to $S$ in $G$ is denoted by $N_r(S) = \{v \in V(G) \mid d(v, S) = r\}$, where $r$ is an integer. We write $N(S)$ instead of $N_1(S)$. The length of a shortest cycle in $G$ is called the *girth* of $G$, denoted by $g(G)$. The *diameter* of $G$ is the maximum distance between any two vertices in $G$.

A $k$-regular graph with girth $g$ is called a $(k, g)$-*graph*. A $(k, g)$-*cage* is a $(k, g)$-graph with the least number of vertices for given $k$ and $g$. We use $f(k, g)$ to denote the number of vertices in $(k, g)$-cages. A cutset $X$ of $G$ is called a *non-trivial cutset* if $X$ does not contain the neighborhood $N(u)$ of any vertex $u \notin X$. A $k$-connected (or $k$-vertex-connected) graph $G$ is called *superconnected* if for every vertex cutset $S \subseteq V(G)$ with $|S| = k$, $S$ is a trivial cutset. Moreover, if $G - S$

has precisely two components, then $G$ is called *tightly superconnected*. Provided there exists a non-trivial cutset, the *superconnectivity* of $G$ is denoted by $\kappa_1 = \kappa_1(G) = \min\{|X| : \text{X is a non-trivial cutset}\}$. The edge-superconnectivity $\lambda_1$ is defined similarly.

Cages were introduced by Tutte in 1947, and have been extensively studied. Most of the work carried out so far has focused on the existence problem, whereas very little is known about the structural properties of $(k, g)$-cages. For more information see survey [12]. Recently, several researchers have studied the connectivity of cages. Fu, Huang and Rodger [3] proved that all cages are 2-connected, and then subsequently showed that all cubic cages are 3-connected. They then conjectured that $(k, g)$-cages are $k$-connected. Daven and Rodger [2], and independently Jiang and Mubayi [4], proved that all $(k, g)$-cages are 3-connected for $k \geq 3$. In [9, 13], some authors also showed that every $(4, g)$-cage is 4-connected. Tang et al [10] conjectured that every $(4, g)$-cage with odd girth is tightly superconnected. In this paper, we show that this conjecture is true.

For the edge-connectivity of $(k, g)$-cage, Wang, Xu and Wang [11] showed that $(k, g)$-cages are $k$-edge-connected when $g$ is odd, and subsequently, Lin, Miller and Rodger [7] proved that $(k, g)$-cages are $k$-edge-connected when $g$ is even. Recently, Lin et al. [5, 8] proved that $(k, g)$-cages are edge-superconnected.

## 2 Main Results

First, we list several known results which will be used in proving our main theorem.

**Theorem 1** (see [3]) *Let $G$ be a $(k, g)$-cage with diameter $D$, where $k \geq 2$ and $g \geq 3$, then $f(k, g) < f(k, g + 1)$ and $D \leq g$.*

**Theorem 2** ([8]) *Every $(k, g)$-cage with odd girth $g \geq 5$ is edge-superconnected.*

For edge-connectivity, Tang et al. [10] conjectured the following:

**Conjecture 1** ([10]) *Every $(k, g)$-cage of odd girth $g \geq 5$ has $\lambda_1 = 2k - 2$.*

We can show that the conjecture is true for $k = 4$ and present it as a lemma below.

**Lemma 1** *Every $(4, g)$-cage of girth $g \geq 5$ has $\lambda_1 = 6$*

The following lemma has been proven in [10].

**Lemma 2** ([10]) *Let $G$ be a $(4, g)$-cage with odd girth $g \geq 5$. Assume that there exists a non-trivial cutset $X$, and $C$ is a component of $G - X$. Then there exists a vertex $u \in V(C)$ such that $d(u, X) \geq (g - 1)/2$.*

We now provide a stronger version of this lemma for $(4, g)$-cages.

**Lemma 3** *Let $G$ be a $(4,g)$-cage with odd girth $g \geq 5$. Assume that there exists a non-trivial cutset $X$, and $C$ is a component of $G-X$. Then* $\max\{d(u,X) : u \in V(C)\} = (g-1)/2$.

**Proof.** By contradiction. Let $G-X$ contains exactly two components $C$ and $C'$. Suppose the lemma is not true, then there is a vertex $u \in C$ such that $d_C(u,X) = (g+1)/2$. We know that the diameter of $G$ is at most $g$ and also by Lemma 2, there exists a vertex $v \in V(C')$ such that $d_{C'}(v,X) \geq (g-1)/2$. Denote $N_C(u) = \{u_1,u_2,u_3,u_4\}$, $N_{C'}(v) = \{v_1,v_2,v_3,v_4\}$ and $X = \{x_1,x_2,x_3,x_4\}$, then $d(u_i,v_j) \geq g-2$, where $i,j = 1,2,3,4$.

*Claim 1. There are at least two pairs of vertices $(u_i,v_j)$ such that $d(u_i,v_j) \geq g-1$.*

Otherwise there exists a vertex $s \in N(u) \cup N(v)$, and $d(u_i,v_j) = g-2$ if $u_i, v_j \neq s$. Then each vertex in $N(u)-s$ is at distance $(g-1)/2$ to each vertex in $X$, and there are at least twelve shortest paths at distance $(g-1)/2$ from $N(u)-s$ to $X$ which can not have a common vertex of $N(X)-X$, otherwise a cycle of length shorter than $g$ appears in $G$. So there are at least twelve edges from $X$ to $C$, then at most four edges left from $X$ to $C'$, a contradiction to Lemma 1.

Without loss of generality, assume $d(u_1,v_1) = d(u_2,v_2) = g-1$, then we can reconstruct a $(4,g')$-graph as follows: In $G' = G-u-v$, add a vertex $y$ and six edges $u_1v_1$, $u_2v_2$, $yu_3$, $yu_4$, $yv_3$ and $yv_4$. So $|V(G')| < |V(G)|$, and it is clear that $g' \geq g$, a contradiction to Theorem 1. □

Suppose $U$ and $W$ are two vertex sets with $|U| = |W|$. For a one-to-one map $f: U \mapsto W$, we define $E(f) = \{uw \mid f(u) = w, u \in U, w \in W\}$. The following lemma is a key technical tool for the construction of a new $(4,g)$-graph of smaller order in our proof.

**Lemma 4** *Let $H$ be a bipartite graph with bipartition $(U,W)$, where $|U| = |W| = 4$, such that $|E(H)| \leq 4$ and $\Delta(H) \leq 3$. Let $H^*$ be a copy of $H$ with bipartition $(U^*,W^*)$. Let $G = H \cup H^*$. Then there exist two one-to-one maps $f: W \mapsto U^*$ and $f^*: W^* \mapsto U$ such that no new 4-cycle created in graph $G \cup E(f) \cup E(f^*)$.*

**Proof.** It suffices to show that the result holds for $|E(H)| = 4$. Suppose that we can partition $H$ into two vertex disjoint subgraphs $H_1 = (U_1,W_1)$ and $H_2 = (U_2,W_2)$ such that no edge $e \in E(H_1,H_2)$, where $U_1 = \{a_1,b_1\}$, $U_2 = \{c_1,d_1\}$, $W_1 = \{a_2,b_2\}$ and $W_2 = \{c_2,d_2\}$. Let two maps be defined by $E(f) = \{a_2c_1^*, b_2d_1^*, c_2a_1^*, d_2b_1^*\}$ and $E(f^*) = \{a_2^*a_1, b_2^*b_1, c_2^*c_1, d_2^*d_1\}$. No new 4-cycles will be created in the graph $G \cup E(f) \cup E(f^*)$. If we can not partition $H$ into two vertex disjoint subgraphs as above, then $H$ must be one of graphs shown in Figure 1 since $E(H) = 4$ and $\Delta(H) \leq 3$. We give the two maps dotted line in Figure 1 for each case. □

To prove that all $(4,g)$-cages with odd girth $g \geq 11$ are tightly superconnected, we use contrapositive arguments by assuming that there exists a non-trivial cutset $S$ of order 4 in $G$. Let $G_1$ be the smaller component of $G-S$

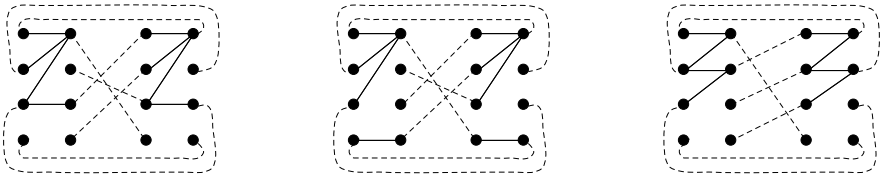


**Fig. 1.** Graph $G \cup E(f) \cup E(f^*)$

and $G_2 = G - S - G_1$, we know that there exists a vertex $u \in G_1$ such that $d(u, S) = (g-1)/2$ by Lemma 3, and $|V(G_1)| \leq |V(G)/2| - 2$. We proceed by constructing a $(4, g')$-graph of order less than $|V(G)|$, where $g' \geq g$, which then yields a contradiction against Theorem 1.

Let $S = \{s_1, s_2, s_3, s_4\}$. Based on the degree distribution of vertices of $S$ in the component $G_1$, we need to consider two major cases which are stated as lemmas below. Because of the page limit, in this extended abstract, we provide a detailed argument for one case in Lemma 5 and a sketch of the construction for the second case. Hopefully, the reader can get a taste of the main idea in both lemmas. For Lemma 6, we omit the entire proof.

**Lemma 5** *If $d_{G_1}(s_i) = 2$ and $d_{G_2}(s_i) = 2$ where $s_i \in S$ for $i = 1, 2, 3, 4$, then $G$ is not a $(4, g)$-cage.*

**Sketch of Proof.** Let $N(u) = \{u_1, u_2, u_3, u_4\}$ and $W_i = N(u_i) - u = \{u_{i1}, u_{i2}, u_{i3}\}$ for $i = 1, 2, 3, 4$. We consider two cases according to the neighbors of $u$, but firstly we show the following claim.

*Claim 1. For each $W_i$, $i = 1, 2, 3, 4$, if there is at most one vertex $x_j \in W_i$ such that $d(x_j, S) = (g-5)/2$, then $G$ is not a $(4, g)$-cage.*

No two vertices of $W_i$ will be at distance $(g-3)/2$ to the same vertex in $S$. Otherwise, a cycle of length $g-1$ appears. Similarly, it is impossible to have $d(u_i, s) = d(u_j, s) \leq (g-3)/2$ for any two different vertices $u_i$, $u_j$ and a vertex $s \in S$. It is clear that there is a vertex in $W_i$ at distance $(g-5)/2$ or $(g-3)/2$ to $S$. Otherwise, the vertex $u_i$ is at distance $(g+1)/2$ to $S$ which is impossible because of Lemma 3. Without loss of generality, assume $u_{i1} \in W_i$ is the vertex that satisfies $d(u_{i1}, S) = (g-5)/2$ or $(g-3)/2$. In the rest of this paper, connecting two vertices means joining the two vertices by an edge and connecting a vertex $x$ to a set $R$ means joining $x$ to every vertex in $R$.

Let $W = \{W_1^{'}, W_2^{'}, W_3^{'}, W_4^{'}\}$, where $W_i^{'} = W_i - u_{i1}$. We shall construct a bipartite graph $H = (W, S)$, where $|W| = |S| = 4$ and $W_i^{'} s_j \in E(H)$ if and only if $d_{G_1}(s_j, W_i - u_{i1}) \leq (g-3)/2$ in $G$. It is clear that there are at most eight paths in total of length at most $(g-3)/2$ from $\bigcup_{i=1,2,3,4} W_i$ to $S$, otherwise a cycle of length shorter than $g$ appears. This implies that there are at most four paths of length at most $(g-3)/2$ from $W$ to $S$. It is clear that at most two of the paths can share the same vertex in $S$ since $d_{G_1}(s_i) = 2$. Also it is easy to see

that these four paths can not start from the same $W_i^{'} = W_i - u_{i1}$, because, by the Pigeon Hole Principle, it would imply that $u_{i1}$ and another vertex from $W_i$ have distance at most $(g-3)/2$ to the same vertex in $S$, and this is impossible.

So we can conclude that $\triangle(H) \leq 3$ and $|E(H)| \leq 4$. Let $H^*$ be a copy of $H$. By Lemma 4, there are two one-to-one maps $f : S \mapsto W^*$ and $f^* : S^* \mapsto W$ such that no new 4-cycles created in $H \cup H^* \cup E(f) \cup E(f^*)$.

We consider a subgraph $N = G[(V(G_1) - u - N(u)) \cup S]$. Let $N^*$ be a copy of $N$. For every $x \in V(N)$, let $x^*$ denote its image in $N^*$. Now we can construct a 4-regular graph $G'$ (see Figure 2) with girth at least $g$ by using $N$ and $N^*$:

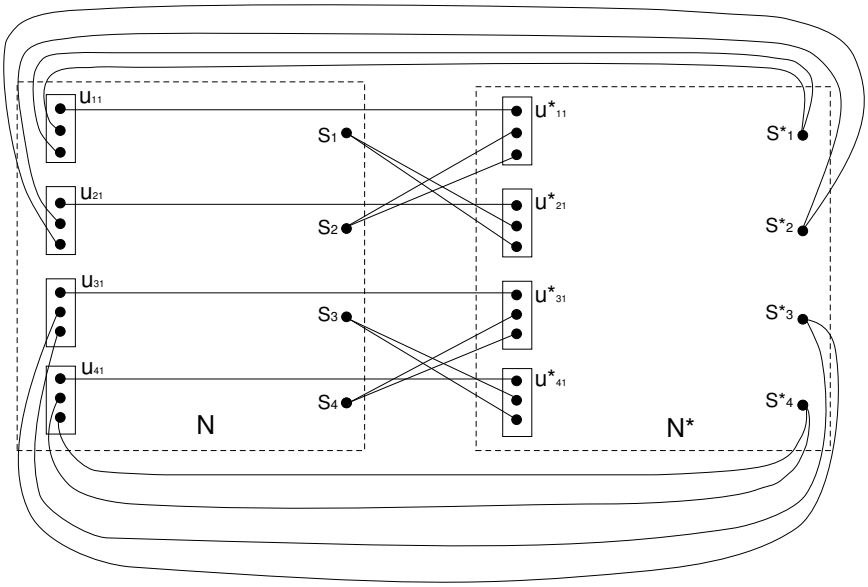



**Fig. 2.** Illustration of the construction for Claim 1, where $f^*(s_i^*) = W_i'$, $i = 1, 2, 3, 4$; $f(s_1) = W_2'^*$, $f(s_2) = W_1'^*$, $f(s_3) = W_4'^*$, $f(s_4) = W_3'^*$.

(a) connect $u_{i1}$ and $u_{i1}^*$ for $i = 1, 2, 3, 4$;

(b) $s_i$ are connected with $u_{j2}^*$ and $u_{j3}^*$ if and only if $f(s_i) = W_j'^*$ for $i, j = 1, 2, 3, 4$;

(c) $s_i^*$ are connected with $u_{j2}$ and $u_{j3}$ if and only if $f^*(s_i^*) = W_j'$ for $i, j = 1, 2, 3, 4$.

Consider the additional edges: any new cycle, say $C$, which was introduced in the construction, has to use at least two new edges. If $C$ goes through two edges in (a), then the cycle $C$ has length at least $2(g-4)+2 > g$ since $g \geq 11$. If $C$ contains two edges in (b) and (c), then the cycle $C$ has length at least $g$, since $H \cup H^* \cup E(f) \cup E(f^*)$ creates no new 4-cycles, so the length of $C$ is at least $(g-1)/2 + (g-3)/2 + 2 = g$. If $C$ goes through one edge in (a) and one

edge in (b) or (c), then the cycle $C$ has length at least $(g-4)+2+(g-3)/2 > g$ since $g \geq 11$. It is obvious that if the cycle $C$ goes through more than two new edges, its length is at least $g$. Hence $G'$ is 4-regular and has girth $g$, but $|V(G')| = |N^*| + |N| = 2|V(G_1)| - 2 < |V(G)|$, a contradiction to the statement that $G$ is a cage.

*Case 1. All the neighbors of $u$ are at distance $(g-3)/2$ to $S$.*

This is a special case of Claim 1, it is clear that in this case, $G$ is not a $(4,g)$-cage.

*Case 2. There are at most three neighbors of $u$ at distance $(g-3)/2$ to $S$.*

The basic idea for this case is somewhat similar to that of Case 1, we sketch it here.

For this case, there exists a vertex $v \in N(u)$ such that $d(v,S) = d(u,S) = (g-1)/2$. Let $N(u) = \{u_1, u_2, u_3, v\}$, $N(v) = \{v_1, v_2, v_3, u\}$, $W_i = N(u_i) - u = \{u_{i1}, u_{i2}, u_{i3}\}$, and $T_i = N(v_i) - v = \{v_{i1}, v_{i2}, v_{i3}\}$, $i = 1, 2, 3$. If there is at most one vertex $x \in W_i$ such that $d(x,S) = (g-5)/2$ for $i = 1, 2, 3$, then from Claim 1, we know that $G$ is not a $(4,g)$-cage.

Assume that there exist two sets $W_i$ and $T_j$, say $W_3$ and $T_3$, such that $|N_{(g-5)/2}(S) \cap T_3| \geq 2$ and $|N_{(g-5)/2}(S) \cap W_3| \geq 2$. By counting the number of paths of length at most $(g-3)/2$ from $N_2(u)$ to $S$, we *can* show that there are exactly two paths of length $(g-5)/2$ and no paths of length $(g-3)/2$ from $W_3$ to $S$. Moreover, the ends of the two paths of length $(g-5)/2$ are distinct in set $N(u_3)$. Thus there are only two paths of length $(g-7)/2$ and no paths of length $(g-5)/2$ from $N_2(u_3)$ to $S$. Now regarding $u_3$ as $u$, we can construct a graph $G'$ in a *fashion similar* to that in Claim 1 and obtain a contradiction. □

**Lemma 6** *If $d_{G_1}(s_1) = d_{G_1}(s_2) = 3$, $d_{G_2}(s_1) = d_{G_2}(s_2) = 1$ and $d_{G_1}(s_3) = d_{G_1}(s_4) = d_{G_2}(s_3) = d_{G_2}(s_4) = 2$, where $s_i \in S$, then $G$ is not a $(4,g)$-cage.*

With the preparation of two lemmas above, we can prove our main result now.

**Theorem 3** *Every $(4,g)$-cage with odd girth $g \geq 11$ is superconnected.*

**Proof.** Suppose $G$ is not superconnected, then we choose a non-trivial cutset $S$ of $G$ such that $S$ minimizes the order of the smaller component of $G-S$ among all non-trivial cutsets. Since $4|V(G_1)| - E(S,G_1) = \sum_{v \in V(G_1)} d_{G_1}(v) \equiv 0 \pmod 2$, we have $E(S,G_1) \equiv 0 \pmod 2$. Similarly, $E(S,G_2) \equiv 0 \pmod 2$. Since every $(4,g)$-cage is edge-superconnected, we need only to discuss three cases for the cutsets $S$ shown in Figure 3. (a) and (b) are impossible by Lemmas 5 and 6. For (c), we can simply delete edge $s_1s_2$ from $G[S]$ and obtain a contradiction as in Lemma 5. □

**Corollary 1** *Every $(4,g)$-cage with odd girth $g \geq 11$ is tightly superconnected.*

**Acknowledgments** The authors are indebted to the anonymous referees for their constructive suggestions.

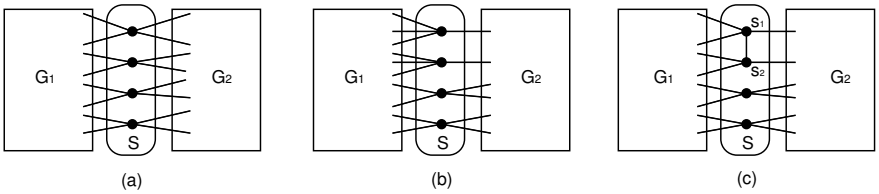



**Fig. 3.** The three extremal cutsets of a $(4, g)$-cage $G$

## References


1. B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.
2. M. D. Daven and C. A. Rodger, $(k, g)$-cages are 3-connected, *Discrete Math.*, 199(1999), 207-215.
3. H. L. Fu, K. C. Huang and C. A. Rodger, Connectivity of cages, *J. Graph Theory*, 24(1997), 187-191.
4. T. Jiang and D. Mubayi, Connectivity and separating sets of cages, *J. Graph Theory*, 29(1998), 35-44.
5. Y. Lin, M. Miller, C. Balbuena and X. Marcote, All $(k, g)$-cages are edge-superconnected, *Networks*, 47(2006), 102-110.
6. Y. Lin, M. Miller and C. Balbuena, Improved lower bound for the vertex connectivity of $(\delta, g)$-cages, *Discrete Math.*, 299(2005), 162-171.
7. Y. Lin, M. Miller and C. A. Rodger, All $(k, g)$-cages are $k$-edge-connected, *J. Graph Theory*, 48(2005), 219-227.
8. X. Marcote and C. Balbuena, Edge-superconnectivity of cages, *Networks*, 43(2004), 54-59.
9. X. Marcote, C. Balbuena, I. Pelayo and J. Fábrega, $(\delta, g)$-cages with $g \geq 10$ are 4-connected, *Discrete Math.*, 301(2005), 124-136.
10. J. Tang, C. Balbuena, Y. Lin and M. Miller, An open problem: $(4, g)$-cage with odd $g \geq 5$ are tightly superconnected, *Proceedings of the Thirteenth Australasian Symposium on Theory of Computing*, 65(2007), 141-144.
11. P. Wang, B. Xu and J.F. Wang, A note on the edge-connectivity of cages, *Electron. J. Combin.*, 10(2003), N4.
12. P. K. Wong, Cages – A survey, *J. Graph Theory*, 6(1982), 1-22.
13. B. Xu, P. Wang and J. F. Wang, On the connectivity of $(4, g)$-cage, *Ars Combin.*, 64(2002), 181-192.

# On the nonexistence of odd degree graphs of diameter 2 and defect 2*

Mirka Miller[1], Minh Hoang Nguyen[2] and Guillermo Pineda-Villavicencio[3]

[1] School of Information Technology and Mathematical Sciences
University of Ballarat, P.O.Box 663, Vic 3353, Australia
and
Department of Mathematics
University of West Bohemia, Pilsen, Czech Republic
m.miller@allarat.edu.au
[2] GNOC Sydney - Ericsson Australia
112-118 Talavera Road, North Ryde, NSW 2113, Australia
and
School of Information Technology and Mathematical Sciences
University of Ballarat, P.O.Box 663, Vic 3353, Australia
minh.n.nguyen@ericsson.com
[3] School of Information Technology and Mathematical Sciences
University of Ballarat, P.O.Box 663, Vic 3353, Australia
and
Department of Computer Science
University of Oriente, Santiago de Cuba, Cuba
gpinedavillavicencio@students.ballarat.edu.au

**Abstract.** In 1960, Hoffman and Singleton investigated the existence of Moore graphs for diameter 2 and found that Moore graphs only exist for maximum degree $d = 2, 3, 7$ and possibly 57. In 1980, Erdős *et al.* asked the following more general question: *Given nonnegative numbers $d$ and $\Delta$, is there a $(d, 2, \Delta)$-graph, that is, a graph of diameter 2, maximum degree $d$ and order $d^2 + 1 - \Delta$?* Erdős *et al.* solved the case for $\Delta = 1$: $C_4$ is the only possible graph.
In this paper, we consider the next case ($\Delta$=2). We prove the nonexistence of such graphs for infinitely many values of odd $d$ and we conjecture that they do not exist for any odd $d$ greater than 5.

**Keywords:** Moore graphs; diameter 2; degree/diameter problem

## 1 Introduction

The *degree/diameter problem* is to determine, for each $d$ and $k$, the largest order $n_{d,k}$ of a graph of maximum degree $d$ and diameter at most $k$. It is easy to show that $n_{d,k} \leq M_{d,k}$, where $M_{d,k}$ is the *Moore bound* given by

* Research partially supported by the Australian Research Council grant ARC DP0450294.

$$n_{d,k} \leq M_{d,k} = 1 + d + d(d-1) + \cdots + d(d-1)^{k-1}.$$

In this paper, we concentrate on the case when the diameter is equal to 2. Since a graph of diameter 2 and maximum degree $d$ has at most $d^2+1$ vertices, it was asked in [3]: Given non-negative numbers $d$ and $\Delta$ (*defect*), is there a graph of diameter 2 and maximum degree $d$ with $d^2+1-\Delta$ vertices? In 1960, it was proved by Hoffman and Singleton[5] that if $\Delta = 0$ then there are unique graphs corresponding to $d = 2, 3, 7$ and possibly $d = 57$. The case $\Delta = 1$ was solved by Erdös *et al.* [3]. In this paper, for the case $\Delta = 2$, we show the nonexistence of such graphs for infinitely many values of odd $d$.

We refer to a graph of maximum degree $d$, diameter $k \geq 2$ and order $M_{d,k} - \Delta$ ($\Delta \geq 1$) as a $(d, k, \Delta)$-graph. Let $G$ be a $(d, k, \Delta)$-graph.

**Definition 1.** Let $u$ be a vertex in $G$. A vertex $v$ in $G$ is called a *repeat* of $u$ with *multiplicity* $m_v(u)$ ($1 \leq m_v(u) \leq \Delta$) if there are exactly $m_v(u)+1$ different paths of lengths at most $k$ from $u$ to $v$.

It is immediate that

**Observation 1.** *Vertex $u$ is a repeat of $v$ with multiplicity $m_u(v)$ if and only if $v$ is a repeat of $u$ with the same multiplicity.*

A repeat with multiplicity 1 will be called a *single* repeat, a repeat with multiplicity 2 will be called a *double* repeat, a repeat with multiplicity $\Delta$ will be called a *maximal* repeat.

We denote by $R_s(u)$ the set of all repeats of a vertex $u$ in $G$. Taking into account the multiplicities of repeats, we denote by $R_m(u)$ the multiset of all repeats of a vertex $u$ in $G$, containing each repeat $v$ of $u$ exactly $m_v(u)$ times.

Let $u$ be a vertex in $G$, we denote by $N(u)$ the set of all neighbours of $u$. If $A$ is a multiset of vertices of $G$, then $N(A)$ denotes the multiset of all neighbours of the vertices of $A$. We use $R_m(A)$ to denote the multiset of all repeats of all vertices in $A$.

**Proposition 1.** *If $G$ is regular then for all $u \in V(G)$,*

$$|R_m(u)| = \sum_{v \in R_s(u)} m_v(u) = \Delta.$$

**Definition 2.** A subset $S$ of $V(G)$ is called a *closed repeat set* if $R_m(S) = S$. A closed repeat set is *minimal* if none of its proper subsets is a closed repeat set.

**Definition 3.** A *repeat subgraph* $H_S$ of a closed repeat set $S$ of $G$ is a multigraph whose vertex set $V(H_S) = S$ and the number of parallel edges between a vertex $u$ and any of its repeats, say $v \in R_m(u)$, equals the multiplicity $m_v(u)$.

We observe that

**Observation 2.** *If $\Delta < 1 + (d-1) + \cdots + (d-1)^{k-1}$ then $G$ is regular.*

It is also true that

**Observation 3.** *If $G$ is regular then the repeat graph $H_G$ of $G$ is $\Delta$-regular.*

For the purpose of this paper, we shall consider each pair of parallel edges in $H_G$ as a cycle of length 2.

**Observation 4.** *$H_G$ is the union of cycles of lengths $\geq 2$, each cycle a minimal closed repeat set of $G$.*

## 2 Structural properties of $(d, 2, 2)$-graphs

Let $G$ be a $(d, 2, 2)$-graph for $d \geq 3$. From Observation 2, we deduce that

**Observation 5.** *Every $(d, 2, 2)$-graph for $d \geq 3$ is regular.*

Let us consider repeat configurations in $(d, 2, 2)$-graphs. Let $u$ be a vertex of a $(d, 2, 2)$-graph. Then there are two possiblities:

- $u$ has two single repeats ($r_i(u)$, $i = 1, 2$).
- $u$ has one double (maximal) repeat ($r(u) = r_1(u) = r_2(u)$) with multiplicity 2.

With respect to repeats in $G$, there are five possible repeat configurations, as depicted in Fig. 2. We denote by $n_0$, $n_1$, $n_{2a}$, $n_{2b}$, $n_{2c}$ the number of vertices of the corresponding repeat types.

We are now interested in pointing out the following theorem, which was proved in [8].

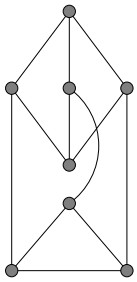
**Theorem 1.** [8] *For odd $d$,*

- *$d = 3$ and $G$ is the graph in Fig. 1$(i)$ or 1$(ii)$*
- *$d = 5$ and $G$ is the graph in Fig. 1$(iii)$*
- *$d \geq 7$ and $n_{2b} = d^2 - 1$.*

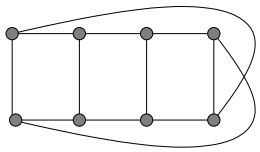
## 3 Nonexistence of infinitely many $(d, 2, 2)$-graphs for odd $d$

Let us now get back to the graph $H_G$. For $d \geq 7$, from Theorem 1, we see that each vertex $u \in G$ has exactly two different repeats, that is, each component of $H_G$ is a cycle of length at least 4. From now on, each cycle in $H_G$ will be called a *repeat cycle*.

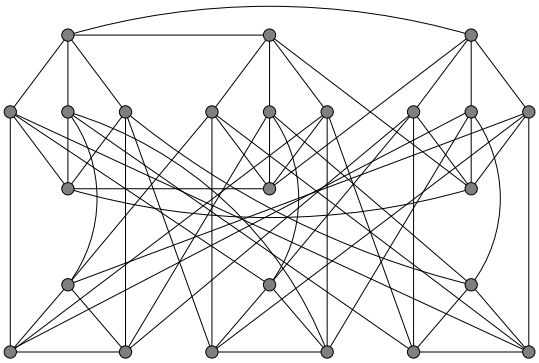
Let $A$ be the adjacency matrix of $G$ and let $B$ be the adjacency matrix of $H_G$, called *the defect matrix*, in which the main diagonals consist entirely of 0's and the row and column sums are equal to 2. With a suitable labeling of $H_G$, $B$ becomes a direct sum of symmetric $a^{th}$-order circulants of the form,



i. $d = 3$: $n_0 = 3, n_1 = 2, n_{2c} = 3$

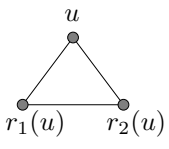


ii. $d = 3$: $n_{2b} = 8$

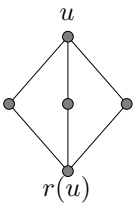


iii. $d = 5$: $n_0 = 9, n_1 = 6, n_{2c} = 9$

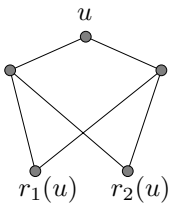
**Fig. 1.** All known $(d, 2, 2)$-graphs for odd $d$.



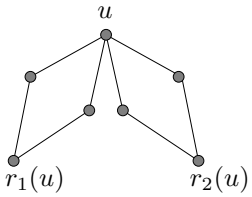

i. Type 0



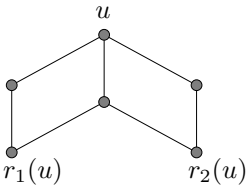

ii. Type 1




iii. Type 2*c*




iv. Type 2*a*




v. Type 2*b*

**Fig. 2.** Repeat configurations in a $(d, 2, 2)$-graph.

$$D_a = \begin{bmatrix} 0\,1\,0\ldots\,0\,1 \\ 1\,0\,1\ldots\,0\,0 \\ 0\,1\,0\ldots\,0\,0 \\ \ldots\ldots\ldots \\ 1\,0\,0\ldots\,1\,0 \end{bmatrix} a \geq 4$$

This matrix has been studied in [2] and [6] in the context of regular graphs of girth 5.

Let $n$ be the order of $G$, let $b$ be the number of cycles in $H_G$ and let $a_i$, $i = 1, \ldots, b$, be the lengths of these cycles. We need to consider the following equation

$$A^2 + A - (d-1)I = J + B \tag{1}$$

where $J$ is a matrix all of whose entries are 1 and $I$ is the identity matrix of order $n$.

The special case when there is just one repeat cycle, that is, $b = 1$ and $H_G = C_n$, was studied by Fajtlowicz in [4]. In that paper, Fajtlowicz proved the following,

**Theorem 2.** [4] *If $B$ is the adjacency matrix of the $n$-cycle then $d = 3$.*

As $A$, $B$ and $J$ are symmetric matrices, they are diagonalizable. Since $J$ commutes with $A$ and $B$, $B$ commutes with $A$ and hence, all the three matrices are simultaneously diagonalizable, that is, there is an orthogonal matrix $P$ for which $P^{-1}AP$, $P^{-1}BP$ and $P^{-1}JP$ are diagonal and the columns of $P$ are corresponding eigenvectors for each of these matrices. Note also that these eigenvectors form an orthogonal basis of $\mathbb{R}^n$.

Furthermore, it is well known that the eigenvalues and its respective multiplicities of a matrix representing a $m$-cycle are

$$\begin{bmatrix} 2 & 2\cos\frac{2\pi}{m} \times 1 & 2c\cos\frac{2\pi}{m} \times 2 \ldots & 2\cos\frac{2\pi}{m} \times (\frac{m}{2} - 1) & -2 \\ 1 & 2 & 2 \ldots & 2 & 1 \end{bmatrix} (m \text{ even})$$

$$\begin{bmatrix} 2 & 2\cos\frac{2\pi}{m} \times 1 & 2\cos\frac{2\pi}{m} \times 2 \ldots & 2\cos\frac{2\pi}{m} \times (\frac{m-1}{2}) \\ 1 & 2 & 2 \ldots & 2 \end{bmatrix} (m \text{ odd})$$

The first row displays the eigenvalues and the second row their respective multiplicities.

Therefore, the eigenvalues of $J + B$ are $n + 2$, 2 and $2\cos\frac{2\pi c_i}{a_i}$ with $c_i = 1 \ldots a_i - 1$ and $i = 1 \ldots b$, of multiplicities 1, $b - 1$ and 1, respectively.

Thus, the spectrum of $A$ in the general case $b \geq 1$ is:

(i) The eigenvalue $d$ with multiplicity 1,
(ii) $b - 1$ roots of the equation

$$\alpha^2 + \alpha - (d-1) = 2, \tag{2}$$

(iii) one root of each of the equations

$$\alpha^2 + \alpha - (d-1) = 2\cos\frac{2\pi c_i}{a_i} \tag{3}$$

where $i = 1 \cdots$ and $c_i = 1 \ldots a_i - 1$.

We denote by $m(\alpha)$ the multiplicity of an eigenvalue $\alpha$ of $A$. The solutions of (2) are $\beta_1 = \frac{-1+\sqrt{4d+5}}{2}$, $\beta_2 = \frac{-1-\sqrt{4d+5}}{2}$ and $m(\beta_1) + m(\beta_2) = b - 1$. The general solution of (3) is

$$\beta = \frac{-1 \pm \sqrt{4d + 8\cos(\frac{2\pi c_i}{a_i}) - 3}}{2}$$

For each even $a_i$, $i = 1, \ldots, b$, and when $c_i = \frac{a_i}{2}$, there is exactly one eigenvalue $\beta$ of $A$ with multiplicity 1 satisfying (3). In other words, corresponding to the special case when $\cos(\frac{2\pi c_i}{a_i}) = -1$, there are eigenvalues $\beta_3 = \frac{-1+\sqrt{4d-11}}{2}$, $\beta_4 = \frac{-1-\sqrt{4d-11}}{2}$. Let $m_e = m(\beta_3) + m(\beta_4)$. Then $m_e$ is exactly the number of even cycles in $H_G$.

**Observation 6.** *For odd $d$, $n = d^2 - 1$ is even. Therefore, if $d$ is odd, then $m_e \equiv b \pmod 2$.*

Using a similar method to that described in [6] we will next show that

**Theorem 3.** *$G$ does not exist for any odd $d \geq 7$ such that $d \neq l^2 + l + 3$ and $d \neq l^2 + l - 1$ for each nonnegative integer $l$.*

*Proof.* If all the four eigenvalues $\beta_1, \ldots, \beta_4$ are irrational, then $m(\beta_1) = m(\beta_2) = \frac{b-1}{2}$ and $m(\beta_3) = m(\beta_4) = \frac{m_e}{2}$. But, by Observation 6, these equations cannot occur at the same time. We can then see that corresponding with those odd values of $d$ such that $d \neq l^2+l+3$ and $d \neq l^2+l-1$ for each nonnegative integer $l$, the four eigenvalues $\beta_1, \ldots, \beta_4$ are irrational, thus the proof follows. □ □

By counting the total number ($N_5$) of 5-cycles in $G$, we are able to derive some further necessary conditions for the existence of $G$.

**Theorem 4.** *$G$ does not exist for any odd $d \geq 7$ such that $N_5$ is not an integer, where*

$$N_5 = \frac{(d^2-1)}{5}\left[\frac{1}{2}(d-3)(d^2+d+4) + d + 2\right].$$

*Proof.* From Theorem 1, we know that $n_{2b} = d^2 - 1$. Let $u_0$ be a vertex of type $2b$. Let $v, w$ be the two repeats of $u_0$ and $N(u_0) = \{u_1, \ldots, u_d\}$. We also denote by $A, B, C, D, E, F$ the sets $N(u_1) \setminus \{u_0, v\}$, $\{v\}$, $N(u_2) \setminus \{u_0, v, w\}$, $\{w\}$, $N(u_3) \setminus \{u_0, w\}$, $\bigcup_{i=4}^{d}(N(u_i) \setminus \{u_0\})$, respectively, as shown in Fig. 3.

Let $e(BE)$ be the number of edges in $G$ between the two sets $B$ and $E$. To reach $v$ from $u_3$ in two steps we must have $e(BE) \geq 1$. But since $u_3$ is of type $2b$, $v$ cannot be a repeat of $u_3$. Thus, $e(BE) = 1$. Similarly, we have $e(AD) = 1$. As a result, $e(BF) = e(DF) = d - 3$.

By an analogous argument, we get $e(AE) = d-2$, which means that $e(AC) = e(CE) = d - 3$. Then it is not difficult to show that $e(CF) = (d-3)^2$, $e(AF) = e(EF) = (d-2)(d-3)$ and $e(FF) = \frac{1}{2}(d-2)(d-3)^2$.

Let $c_5$ be the number of different cycles of length 5 on which $u_0$ lies. Then $c_5 = e(FF) + e(FA) + 2e(FB) + e(FC) + 2e(FD) + e(FE) + e(EC) + 2e(EB) + e(EA) + 2e(AD) + e(AC) = \frac{1}{2}(d-3)(d^2+d+4) + d + 2$.

For odd $d \geq 7$, every vertex of $G$ is of type $2b$ and so the number of different cycles of length 5 in $G$ is $N_5 = \frac{1}{5}(d^2-1)c_5$. The theorem then follows. □ □

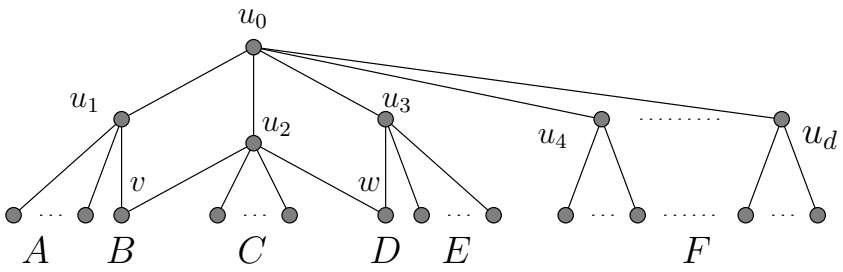



**Fig. 3.** Illustration for proof of Theorem 4.

The results of Theorems 3 and 4 improve the upper bound for the order of $(d, 2, 2)$-graphs so that $n_{d,2} \leq d^2 - 3$ for infinitely many odd degrees $d$. For $d \geq 7$, the first 50 values of $d$ for which $G$ might still exist are shown in Table 1.

**Table 1.** The first 50 values of $d$ for which a $(d, 2, 2)$-graph might still exist for odd $d$.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 11 | 19 | 23 | 29 | 33 | 41 | 59 | 71 | 89 |
| 93 | 109 | 113 | 131 | 159 | 181 | 209 | 213 | 239 | 243 |
| 271 | 309 | 341 | 379 | 383 | 419 | 423 | 461 | 509 | 551 |
| 599 | 603 | 649 | 653 | 701 | 759 | 811 | 869 | 873 | 929 |
| 933 | 991 | 1059 | 1121 | 1189 | 1193 | 1259 | 1263 | 1331 | 1409 |

Finally, we conjecture the following

**Conjecture 1.** *For odd $d \geq 7$, $(d, 2, 2)$-graphs do not exist.*

## References

1. Bannai, E., Ito, T.: On finite Moore graphs. J. Fac. Sci. Tokyo Univ. **20** (1973) 191–208
2. Brown, W.G.: On the non-existence of a type of regular graphs of girth 5. Canad. J. Math. **19** (1967) 644-648
3. Erdös, P., Fajtlowicz, S., Hoffman, A. J.: Maximum degree in graphs of diameter 2. Networks **10** (1980) 87–90

4. Fajtlowicz, S.: Graphs of diameter 2 with cyclic defect. Colloq. Math. **51** (1987) 103–106.
5. Hoffman, A.J., Singleton, R.R.: On Moore Graphs with diameters 2 and 3. IBM J. Res. Dev. **64** (1960) 15-21
6. Kovács, P.: The Non-existence of Certain Regular Graphs of Girth 5. J. Combin. Theory Ser. B **30** (1981) 282–28
7. Miller, M., Širáň, J.: Moore graphs and beyond: A survey of the degree/diameter problem. Electronic Journal of Combinatorics **DS14** (2005) 1–61
8. Nguyen, M. H., Miller, M.: Structural properties of graphs of diameter 2 with defect 2. preprint

# Computing Incongruent Restricted Disjoint Covering Systems

Gerry Myerson[1], Jacky Poon[1] and Jamie Simpson[2]

[1] Department of Mathematics, Macquarie University, Sydney, NSW, gerry@ics.mq.edu.au, jackypoon@optusnet.com.au
[2] Department of Mathematics and Statistics, Curtin University of Technology, Perth, WA, simpson@maths.curtin.edu.au

**Abstract.** An *incongruent restricted disjoint covering system* of length $n$ is a sequence of positive integers $s_1, \ldots, s_n$ with the property that $s_i = m$ for some $m$ if and only if $s_{i+km} = m$ for all $k$ for which $i + km$ is in $[1, n]$, and further such that any integer appearing in the sequence appears at least twice. An example of length 11 is

$$6, 9, 3, 4, 5, 3, 6, 4, 3, 5, 9.$$

We describe two algorithms for finding all such systems of a given length.

## 1 Introduction

An *incongruent restricted disjoint covering system* (henceforth IRDCS) of length $n$ is a sequence of positive integers $s_1, \ldots, s_n$ with the property that $s_i = m$ for some $m$ if and only if $s_{i+km} = m$ for all $k$ for which $i + km$ is in $[1, n]$, and further such that any integer appearing in the sequence appears at least twice. An example of length 11 is

$$6, 9, 3, 4, 5, 3, 6, 4, 3, 5, 9.$$

We introduced the idea of an IRDCS in [5] and [3] as a new twist on the venerable theme of covering systems of congruences. We begin with some background on this, partly by way of motivation and partly to explain the impressive title of our paper.

We write $S(m, a)$ for the congruence class $\{x : x \equiv a \pmod{m}\}$. A *covering system of congruences* is a set of congruence classes with the property that every integer belongs to at least one class. If no integer belongs to more than one class then the system is *disjoint* (or *exact*), if the moduli of the classes are distinct then the system is *incongruent*. An example of a disjoint covering system is

$$\{S(2, 0), S(2, 1)\},$$

and an example of an incongruent system is

$$\{S(2, 0), S(3, 0), S(4, 1), S(6, 1), S(12, 11)\}.$$

It is not possible for a system to be both disjoint and incongruent (see [6]). These systems were introduced by Erdős in [1] and have spawned a large literature (see the survey [6] and sections F13 and F14 in [2]).

We define a *restricted disjoint covering system on* $[1, n]$ as a set of congruence classes such that each integer in the interval $[1, n]$ belongs to exactly one class, and each class contains at least two members of the interval. The condition that the classes contain at least two members is included to avoid trivialities. As with standard covering systems we describe such a system as *incongruent* if the moduli are distinct. Here is an example of an IRDCS on $[1, 11]$.

**Example 1.**

$$S(6, 1), S(9, 2), S(3, 0), S(4, 0), S(5, 0).$$

Rather than exhibiting an IRDCS in this way we can do so by writing down a sequence of $n$ integers the $i$th of which equals the modulus of the unique congruence class to which $i$ belongs. Thus Example 1 becomes

$$6, 9, 3, 4, 5, 3, 6, 4, 3, 5, 9$$

which is the sequence we started with. It is easy to see that our original definition and the definition just given in terms of congruence classes are equivalent.

We have found all IRDCS of lengths up to 40 and found that their number increases quickly with length: there are 1,805,096 with length 40. In [3] we reported on some observations from this data, proved some results about the structure of IRDCS and presented some open problems. Some of these problems are repeated at the end of this paper and some properties of the data are given in Table 1. The main purpose of the present paper is to discuss the algorithms we used to find these IRDCS. We used two algorithms, one based on Knuth's recursive Dancing Links algorithm and the other a more intuitive back-tracking algorithm. We will describe the second in some detail and the first more sketchily. We finish with a comparison of the algorithms, a general discussion and the open problems.

## 2 Backtracking

We present an algorithm that finds all IRDCS on input $n$. The systems are built in an array $x[1..n]$, initially having a 1 in each position, being built up one modulus at a time. If we find a modulus is causing a clash with an already used modulus, that is, if we are trying to use two moduli at the same position, we try a larger modulus if one is available, otherwise backtrack to the previous modulus used and increase that.

Two other arrays are used. The *modusage* array shows which moduli have already been used - true if used, false if not. The modulus currently being considered is $m$. When a new modulus is being chosen, we increase from the previous value till we find one which hasn't yet been used. The other array is *primary*. A *primary* array element is true if that position in the $x$ array is empty or if it was

the first position which used some modulus. After a modulus had been placed in a certain position we fill other appropriate positions with this modulus, but these later (or secondary) occurrences of the modulus would get 'false' in the corresponding position of *primary*. The variable *position* shows the position at which we've just inserted a primary modulus or where we're just about to do so. The variable *position* starts at $\lfloor n/2 \rfloor + 1$, then changes by having $increment[i]$ added to it.

$Increment[i]$ goes $1, -2, 3, -4, \ldots, \pm(n+1)$ for odd $n$, signs reversed for even $n$. The next increment to be used (if we're not backtracking) is $increment[ctr]$, ($ctr$ for counter). If we're backtracking we have to subtract an increment and this requires decreasing $ctr$ by 1 first. The point of this is that we filling the middle of the interval $[1, n]$ first, then moving towards the ends. We found this was faster than starting at one end of the interval.

If we get a clash while entering secondary moduli the variable *clash* takes the value true.

**Initialisation**
Input $n$
Set all entries of vectors $x[1..n]$ to 1, $primary[1..n]$ to $true$, $modusage[2..n]$ to $false$ and for $i = 1..n+1$ $increment[i]$ to $(-1)^{n+1} * i$. Set $position = \lfloor n/2 \rfloor + 1$, $ctr = 1$, $finished := false$ and $clash := false$.
**Begin main loop**
while not $finished$ do
**Choose next modulus**
Set $m =$ next unused modulus after $x[position]$
if this modulus feasible at this position then
**Feasible modulus found - enter while checking for clashes**
$i := position - m * \lfloor (position - 1)/m \rfloor$
while $i \leq n$ and not clash
if $i \neq position$ then
if $x[i] = 1$ then
$x[i] := x[position]$
$primary[i] := false$
else
$clash := true$
$i = i - m$
end if
end if
$i := i + m$
**If clash has occurred clear last modulus except at current position**
if $clash$ then
$modusage[m] := false$
for $i \equiv position \pmod{m}$ and $primary[i] = false$ do
$x[i] = 1$
$primary[i] = true$

**Check to see if finished current system**
    else
        while $x[position] > 1$ and $ctr < n$ do:
            if $ctr < n$ then
                $position := position + increment[ctr]$
            end if
            $ctr := ctr + 1$
        end while
        if $ctr > n$ then
            output system
            $ctr := ctr + 1$
            $modusage[m] := false$
            set *position* to that of primary position for $m$
            reinitialise other positions where this $m$ used
        end if
    end if
else
**Backtrack or finish**
    if $position = \lfloor n/2 \rfloor + 1$ then
        $finished := true$
        Output "No more solutions"
    else
        set $m$ equal to last modulus entered
        reinitialise all positions where this $m$ used
        change position to last modulus' primary position
    end if
end if
end main loop

## 3 Recursion

The second algorithm we used was Knuth's Dancing Links implementation of his Algorithm X [4] for solving the NP-complete Exact Cover problem. Exact Cover may be given as follows.

**Instance** An $m \times n$ matrix $A$ with each entry either 0 or 1.

**Question** Find a subset of the rows of $A$ such that for each column exactly one row in the set has a 1 in that column.

To find a length $n$ IRDCS we consider all congruence classes that might appear in the IRDCS. These are $\{S(m, a) : 2 \leq m \leq n-1, 1 \leq a \leq \min\{m, n-m-1\}$ For each such congruence class construct a row of a matrix $A$ which has

a 1 in column $j$ if $j \in S(m, a)$ and 0 otherwise. Thus the first few rows of the matrix for $n = 11$ would be:

$$\begin{bmatrix} 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0 \\ 1\,0\,0\,1\,0\,0\,1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1\,0\,0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1\,0\,0\,1\,0\,0 \end{bmatrix}.$$

Clearly finding an IRDCS of length $n$ is equivalent to finding an exact cover for this matrix.

Algorithm X solves the problem recursively by choosing a row $i$ from $A$ then for each $j$ such that $a_{i,j} = 1$ delete column $j$ from $A$, and for each $k \neq i$ such that $a_{k,j} = 1$ delete row $k$. Then apply the algorithm to this reduced matrix.

The algorithm is expensive in space and requires the computer to spend most of its time searching for 1s. For our problem $A$ will have $n^2/4 + O(n)$ rows. The Dancing Links algorithm implements the algorithm using circular doubly-linked lists of the 1s in the matrix. There is a list of 1s for each row and each column. Each 1 in the matrix has a link to the next 1 above, below, to the left, and to the right of itself.

Table 1 compares the speed of our two algorithms, showing that the second is substantially quicker. Both algorithms were run on a PC with a 2.0 GHz processor. We had not expected Knuth's algorithm to be faster than back-tracking since it is designed to solve a very general problem and does not exploit the special structure of an IRDCS. Perhaps another faster algorithm can be found. Knuth's algorithm solves an NP-complete problem. A decision problem relating to our situation is: "given a set of positive integers, does there exist an IRDCS with these integers as moduli?" We do not know the complexity of this question.

| Length | Number of IRDCS | Backtracking | Dancing Links |
|---|---|---|---|
| 35 | 68,176 | 3 seconds | 1 seconds |
| 36 | 85,762 | 4 | 2 |
| 37 | 304,892 | 10 | 3 |
| 38 | 855,072 | 24 | 7 |
| 39 | 1,229,050 | 41 | 11 |
| 40 | 1,805,096 | 68 | 18 |

**Table 1.** Times taken to find all IRDCS of various lengths using the two algorithms described above.

## 4 Open Problems

We end with some open problems.

(1) Is there a more efficient method of finding IRDCS than those described here?

(2) Do there exist IRDCS with all moduli odd?

(3) Can the smallest modulus of an IRDCS be arbitrarily large?

(4) In [3] we showed that the number of distinct moduli appearing in an IRDCS of length $n$ is at most $(n-1)/2$. This bound is attained in the example with $n = 11$ given above, but computational results suggest that a stronger result should be possible for large $n$. What is it?

(5) The following two IRDCS both have length 43 and their sets of moduli are disjoint.

$$\{S(24,1), S(2,2), S(4,3), S(36,5), S(12,9), S(16,13), S(20,17)\}$$

$$\{S(25,1), S(33,2), S(7,3), S(8,4), S(9,5), S(21,6), S(18,7), S(13,8), S(10,9), S(11,11), S(27,13), S(15,15), S(26,16), S(19,18)\}.$$

Generally, if each of $\{S(m_1,a_1), ..., S(m_s,a_s)\}$ and $\{S(n_1,b_1), ..., S(n_t,b_t)\}$ is an IRDCS for $[1,n]$ and their sets of moduli are disjoint, as in the case above, then

$$\{S(3m_i, 3a_i+1) : i = 1, \ldots, s\} \cup \{S(3n_i, 3b_i+2) : i = 1, \ldots, t\} \cup \{S(3,0)\}$$

is an IRDCS for $[1,3n]$ in which every modulus is divisible by 3.

This suggests the question, does an IRDCS exist with every modulus divisible by $k$ for any value of $k$? This example shows it's possible for $k = 3$. We can easily produce examples for $k = 2$ by doubling each modulus then inserting a 2 in every second position, so that, for instance, the $n = 11$ example produces

$$2, 12, 2, 18, 2, 6, 2, 8, 2, 10, 2, 6, 2, 12, 2, 8, 2, 6, 2, 10, 2, 18, 2.$$

(6) Our definition of an IRDCS requires that each congruence is satisfied at least twice. Do analogous systems exist in which each congruence is satisfied at least $k$ times for values of $k$ exceeding 2?

## References

1. P. Erdős, On integers of the form $2^k + p$ and some related problems, Summa Brasil. Math. 2(1950), 192-210.
2. R.K. Guy, Unsolved Problems in Number Theory, 3rd edition, Springer, 2004.
3. Gerry Myerson, Jacky Poon and Jamie Simpson, Incongruent Restricted Disjoint Covering Systems, submitted to Discrete Mathematics.
4. Donald Knuth, Dancing Links, http://www-cs-faculty.stanford.edu/~knuth/preprints.html, P159, retrieved 7/7/7.
5. G. Myerson, ed., Western Number Theory Problems. Western Number Theory Conference, 18 and 20 Dec., 2006, Website in preparation.
6. Š. Porubský and J. Schönheim, Covering Systems of Paul Erdős: past, present and future in Halász, Gábor (ed.), Paul Erdős and his Mathematics I, Bolyai Soc. Math. Stud. 11, 581-627 (2002).

# Existence of Regular Supermagic Graphs

Andrea Semaničová[1], Jaroslav Ivančo[2] and Petr Kovář[3]
andrea.semanicova@tuke.sk, jaroslav.ivanco@upjs.sk, petr.kovar@vsb.cz

[1] *Department of Appl. Mathematics, Technical University, Letná 9, 04200 Košice, Slovakia*
[2] *Institute of Mathematics, P.J.Šafárik University, Jesenná 5, 04154 Košice, Slovakia*
[3] *Department of Appl. Mathematics, VŠB – Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic*

**Abstract.** A graph is called supermagic if it admits a labeling of the edges by pairwise different consecutive integers such that the sum of the labels of the edges incident with a vertex is independent of the particular vertex.
In this talk we deal with the existence of $r$-regular supermagic graph of order $n$. Some constructions of supermagic labeling of regular graphs are described.

*Keywords:* regular graph, supermagic graph, $(a, 1)$-antimagic graph, circulant graph

## 1 Introduction

We consider finite undirected graphs without loops and multiple edges. Let $G$ be a graph, we denote by $V(G)$ and $E(G)$ the vertex set and edge set of $G$ respectively.

Let $G$ be a graph and $f$ be a mapping from $E(G)$ into positive integers. The *index-mapping* of $f$ is the mapping $f^\star$ from $V(G)$ into positive integers defined by

$$f^\star(v) = \sum_{e \in E(G)} \eta(v, e) f(e) \quad \text{for every } v \in V(G),$$

where $\eta(v, e)$ is equal to 1 when $e$ is an edge incident with the vertex $v$, and 0 otherwise. An injective mapping $f$ from $E(G)$ into positive integers is called a *magic labeling* of $G$ for an *index* $\lambda$ if its index-mapping $f^\star$ satisfies

$$f^\star(v) = \lambda \quad \text{for all } v \in V(G).$$

A magic labeling $f$ of $G$ is called *supermagic* if the set $\{f(e) : e \in E(G)\}$ consists of consecutive positive integers. A graph $G$ is called *supermagic* (*magic*) if and only if there exists a supermagic (magic) labeling of $G$.

The concept of magic graphs was introduced by Sedláček [12]. The regular magic graphs are characterized in [3]. Two different characterizations of all magic graphs are given in [10] and [9]. Supermagic graphs were introduced by

B.M. Stewart [13]. It is easy to see that the classical concept of a magic square of $n^2$ boxes corresponds to the fact that the complete bipartite graph $K_{n,n}$ is supermagic for every positive integer $n \neq 2$ (see also [13]). Stewart [14] characterized supermagic labeling of complete graphs. In [7], supermagic regular complete multipartite graphs and supermagic cubes are characterized. In [8] characterizations are given for magic line graphs of general graphs and supermagic line graphs of regular bipartite graphs. In [11] and [1] supermagic labellings of the Möbius ladders and two special classes of 4-regular graphs are constructed. Constructions of supermagic labellings of various classes of regular graphs are described in [6] and [7]. More comprehensive information on magic and supermagic graphs can be found in [5].

Trenkler proved the following condition

**Proposition 1** [15] *A connected magic graph with $n$ vertices and $\varepsilon$ edges exists if and only if $n = 2$ and $\varepsilon = 1$ or $n \geq 5$ and $\frac{5n}{4} < \varepsilon \leq \frac{n(n-1)}{2}$.*

However, for supermagic graphs we do not have similar results. In [4], it is proved that if $d$ is the greatest common divisor of integers $n$ and $\varepsilon$, and if $\frac{n}{d}$ and $\varepsilon$ are both even, then there exists no supermagic graph of order $n$ and size $\varepsilon$. Moreover in [4] the following bound was established for the number of edges in supermagic graphs of order $n$:

$$\lceil (3-\sqrt{3})n \rceil \leq |E(G)| \leq \begin{cases} 8 & \text{for} \quad n = 5, \\ \frac{n(n-1)}{2} & \text{for} \quad 6 \leq n \not\equiv 0 \pmod 4, \\ \frac{n(n-1)}{2} - 1 & \text{for} \quad 8 \leq n \equiv 0 \pmod 4. \end{cases}$$

The existence of supermagic graphs is a very complicated problem. In this talk we will focus on the existence of regular supermagic graph of a given order.

## 2 Regular supermagic graphs

Ivančo proved the following necessary conditions for supermagic regular graphs

**Proposition 2** [7] *Let $G$ be an $r$-regular supermagic graph of order $n$. Then the following statements hold:*

*(i) if $r \equiv 1 \pmod 2$, then $n \equiv 2 \pmod 4$;*
*(ii) if $r \equiv 2 \pmod 4$ and $n \equiv 0 \pmod 2$, then $G$ contains no component of an odd order;*
*(iii) if $n > 2$, then $r > 2$.*

It is easy to see, that $K_2$ is the only one 1-regular supermagic graph and that there exists no 2-regular supermagic graph. For 3-regular supermagic graphs, let us consider the Möbius ladder. The Möbius ladder $M_n$, where $6 \leq n \equiv 0 \pmod 2$, is the 3-regular graph consisting of the cycle $C_n$ of length $n$, in which all pairs of the opposite vertices are connected. Sedláček [11] proved that if $6 \leq n \equiv 2 \pmod 4$ then the Möbius ladder is supermagic.

**Proposition 3** [14] *A complete graph of order $n$ is supermagic if and only if $n = 2$ or $5 < n \not\equiv 0 \pmod 4$.*

From this result it is clear that the $(n-1)$-regular supermagic graph of order $n$ exists if and only if $n = 2$ or $5 < n \not\equiv 0 \pmod 4$.

Before we present the main result, we first introduce some definitions.

Let $n, m$ and $a_1, \ldots, a_m$ be positive integers, $1 \leq a_i \leq \lfloor \frac{n}{2} \rfloor$ and $a_i \neq a_j$ for all $1 \leq i, j \leq m$. An undirected graph with the set of vertices $V = \{v_1, \ldots, v_n\}$ and the set of edges $E = \{v_i v_{i+a_j} : 1 \leq i \leq n, 1 \leq j \leq m\}$, the indices being taken modulo $n$, is called a *circulant graph* and it is denoted by $C_n(a_1, \ldots, a_m)$.

A graph $G$ is called $(a, 1)$-*antimagic* if it is possible to label its edges with the integers from the set $1, 2, \ldots, |E(G)|$ such that

$$\{f^\star(v) : v \in V(G)\} = \{a, a+1, \ldots, a+|V(G)|-1\}.$$

The $(a, 1)$-antimagic labeling is a special type of $(a, d)$-antimagic labeling defined by Bodendiek and Walter [2].

A $k$-regular spanning subgraph of a graph is called the $k$-regular factor of a graph. In [7] it is proved

**Proposition 4** [7] *Let $G$ be a graph decomposable into pairwise edge-disjoint supermagic regular factors. Then $G$ is supermagic.*

In the next sections, we will deal with the constructions of supermagic $r$-regular graphs of order $n$ for $r \geq 4$. We will consider two cases, when $r$ is odd and $r$ is even.

## 3 Main results

According to Proposition 2, if $r = 2k+1$ then $n \equiv 2 \pmod 4$ for a graph $G$.

For $r = 4k+1$, we proved:

**Theorem 1** *Let $n, k$ be positive integers, $4k+2 \leq n \equiv 2 \pmod 4$. Then there exists a $(4k+1)$-regular supermagic graph of order $n$.*

*Proof.* Let $n, k$ be positive integers, $4k+2 \leq n \equiv 2 \pmod 4$.

If $4k+2 = n$ then the $(4k+1)$-regular graph is isomorphic to the complete graph $K_{4k+2}$ and according to Proposition 3 is supermagic.

If $4k+2 < n \equiv 2 \pmod 4$ then the graph

$$G := C_n(2, 4, \ldots, 4A-2, 4A; 1, 3, \ldots, 4B-3, 4B-1, \frac{n}{2})$$

is supermagic graph of order $n$. To prove this, we will decompose this graph into two edge-disjoint supermagic regular factors $G_1$ and $G_2$ where

$$G_1 := C_n(2, 4, \ldots, 4A-2, 4A, \frac{n}{2}),$$
$$G_2 := C_n(1, 3, \ldots, 4B-3, 4B-1),$$

where $A + B = k$, $8A + 2 \leq n$ and $8B + 2 \leq n$. It is not difficult to prove that both graphs $G_1$ and $G_2$ are supermagic. According to Proposition 4, the graph $G$ is supermagic.

Analogously to construct the $(4k + 3)$-regular supermagic graph of order $n$ we shall decompose the graph into three factors:

$$G_1 := C_n(1, \tfrac{n}{2});$$
$$G_2 := C_n(3, 5, 7, 9, \ldots, 4A - 1, 4A + 1);$$
$$G_3 := C_n(2, 4, 6, 8; 10, 12, 14, 16; \ldots; 8B - 6, 8B - 4, 8B - 2, 8B),$$

where $A, B$ are arbitrary positive integers such that

$$4k + 3 = 3 + 4A + 8B,\ 4A \leq \tfrac{n}{2} - 3,\ 4B \leq \tfrac{n}{2} - 1.$$

It is not difficult to show that $G_1$, $G_2$ and $G_3$ are supermagic graphs. According to Proposition 4 the graph with the factorization into $G_1$, $G_2$ and $G_3$ is a $(4k+3)$-regular supermagic graph of order $n$. Thus we have

**Theorem 2** *Let $n, k$ be positive integers, $4k + 4 < n \equiv 2 \pmod 4$. Then there exists a $(4k + 3)$-regular supermagic graph of order $n$.*

Using the above mentioned ideas, we construct $8k$-regular supermagic graphs of an arbitrary order and $4k$-regular supermagic graphs of even order.

**Theorem 3** *Let $n, k$ be positive integers. Then the following statements hold:*

*(i) if $8k + 1 \leq n$, then there exists a $8k$-regular supermagic graph of order $n$.*
*(ii) if $4k + 1 \leq n \equiv 0 \pmod 2$, then there exists a $4k$-regular supermagic graph of order $n$.*

Moreover we are able to prove the following two lemmas

**Lemma 1** *A 4-regular supermagic graph of order $n$ exists if and only if $n \geq 6$.*

**Lemma 2** *A 6-regular supermagic graph of order $n$ exists if $n \geq 7$.*

Analogously as in the proof of Theorem 2, we prove that up to a finite number of cases, there exists a $2r$-regular supermagic graph of order $n$ by decomposing the original graph into 4-, 6- and $8k$-regular supermagic factors.

## 4 Conclusion

In this talk we introduce some constructions of the $r$-regular supermagic graphs of order $n$ for all $r$ and $n$ except certain values. However, there are some difficulties of applying this technique for dense graphs. This is a topic for further investigation.

**Acknowledgement.** Support of Slovak VEGA Grant 1/4005/07 is acknowledged. Research for this article was partially supported by the institutional project MSM6198910027.

## References


1. M. Bača, I. Holländer, Ko-Wei Lih, Two classes of super-magic graphs, *J. Combin. Math. Combin. Comput.* **23** (1997) 113–120.
2. R. Bodendiek and G. Walther, Aritmethisch antimagische graphen, in: K. Wagner, R. Bodendiek, eds., Graphentheorie III (BI-Wiss. Verl., Mannheim, 1993).
3. M. Doob, Characterizations of regular magic graphs, *J. Combin. Theory*, Ser. B **25** (1978) 94–104.
4. S. Drajnová, J. Ivančo, A. Semaničová, Numbers of edges in supermagic graphs, *J. Graph Theory* **52** (2006) 15–26.
5. J.A. Gallian, A dynamic survey of graph labeling, *Electronic J. Combinatorics* # **DS6**(2007).
6. N. Hartsfield, G. Ringel, Pearls in Graph Theory, Academic Press, Inc., San Diego, 1990.
7. J. Ivančo, On supermagic regular graphs, *Mathematica Bohemica* **125** (2000) 99–114.
8. J. Ivančo, Z. Lastivková, A. Semaničová, On magic and supermagic line graphs, *Mathematica Bohemica* **129** (2004) 33–42.
9. R.H. Jeurissen, Magic graphs, a characterization, *Europ. J. Combin.* **9** (1988) 363–368.
10. S. Jezný, M. Trenkler, Characterization of magic graphs, *Czechoslovak Math. J.* **33** (1983) 435–438.
11. J. Sedláček, On magic graphs, *Math. Slovaca* **26** (1976) 329–335.
12. J. Sedláček, Problem 27. Theory of Graphs and Its Applications, Proc. Symp. Smolenice, Praha (1963) 163–164.
13. B.M. Stewart, Magic graphs, *Canad. J. Math.* **18** (1966) 1031–1059.
14. B.M. Stewart, Supermagic complete graphs, *Canad. J. Math.* **19** (1967) 427–438.
15. M. Trenkler, Numbers of vertices and edges of magic graphs, *Ars Combinatoria* **53** (2000) 93–96.

# Some Parameterized Problems Related to Seidel's Switching

Ondřej Suchý

Department of Applied Mathematics, Faculty of Mathematics and Physics,
Charles University, Prague, Czech Republic

**Abstract.** Seidel's switching of vertex set is an operation, which deletes edges leaving this set from the graph and adds those edges between the set and the rest of the graph, that weren't there originally. Other edges remain untouched by this operation. The usual question in parameterized complexity is whether the exponential part of the algorithms for hard problems can be bounded by some function of only selected parameter, which we assume to be small. We study the complexity of a question, if the given graph can be turned into a graph with some property P using Seidel's switching, from the parameterized view. We show fixed-parameter tractability of switching to a regular graph, to a graph with bounded degree of vertices, or with bounded number of edges, a graph without a forbidden subgraph and a bipartite graph.

## 1 Introduction

Parameterized complexity became one of the standard tools for exact solving of hard problems these days. The basic notions and ideas of parameterized complexity, were introduced by Downey and Fellows [1].

We say that a *parameterized problem* $P \subseteq \Sigma^* \times \mathbb{N}$, where $\Sigma$ is some fixed alphabet is *fixed-parameter tractable* (FPT), if there is an algorithm that decides whether the input $(I, k)$ belongs to $P$ in time $f(k)|I|^c$, where $c$ is a fixed constant and $f(k)$ is a function independent of the size of the input $|I|$. If $(I, k) \in \Sigma^* \times \mathbb{N}$ is an *instance* of the problem $P$, then $I$ is called the *main part* and $k$ is called the *parameter*.

We use the standard $\mathcal{O}^*()$ notion for the asymptotic running time of an exact algorithm, which suppresses the polynomial part of the function.

Seidel's switching is possibly the most elegant graph transformation since we can achieve global modification of a graph by switching just one vertex. It was introduced by Dutch mathematician J.J.Seidel [2].

For a simple undirected graph $G = (V, E)$ and $A \subseteq V$ subset of its vertices we define *Seidel's switching of a set* $A$ in a graph $G$ to be the graph $S(G, A) = (V, E')$, where

$$E' = E \triangle \{\{u, v\} | u \in A, v \in V \setminus A\}.$$

Specially if $A = \{v\}$, we talk about Seidel's switching of a vertex $v$.

It is quite easy to see (cf. Lemma 1), that the following relation is an equivalence. Graphs $G$ and $H$ are *switching equivalent* (denoted by $G \sim H$), iff there is an $A \subseteq V_G$ such that $S(G, A)$ is isomorphic to $H$. An equivalence class of this relation, e.g. the set $[G] = \{S(G, A) : A \subseteq V_G\}$ is called *switching class* of the graph $G$.

Since all the switchings in this paper are Seidel's we sometimes omit this adjective.

There were many results concerning structural properties as well as complexity questions related to Seidel's switching. Probably the most studied problem is, whether the given (or every in the structural approach, respectively) graph can be switched to have some property $P$. This problem is usually denoted by $S(P)$. It is highly interesting that the complexity of the problem $S(P)$ is not related to the complexity of the original problem $P$. In particular there are problems known such that the problem $P$ is NP-complete and $S(P)$ is polynomial and vice-versa. For a survey of results in this field see e.g. [3] or [4].

For the problems we concentrate on, the most important is the work of Kratochvíl [5]. He proved that the problem of switching to a $k$-regular graph is NP-complete when $k$ is a part of the input. On the other hand, he proved that for $k$ fixed and for the polynomial time recognizable class of $k$-degenerate graphs $P$ (e.g. the class of all $k$-regular graphs), the problem $S(P)$ is polynomial time solvable. Another important result is due to Jelínková [6], who proved that it is NP-complete to decide whether the graph can be switched to a graph having at most $k$ edges, when $k$ is a part of the input.

In the next section we provide some basic results about the switching. Then we examine the problem whether the given graph can be switched to a graph with all the degrees at most $k$. As with all the other problems here considered we show fixed-parameter tractability. Since the classical complexity is not known yet, the main reason to consider this problem is to provide a tool for the problems that follow.

We highlight that the parameterized dual of the problem (Switching to a graph with all the degrees at least $k$) is only of limited interest since it is trivially fixed parameter tractable due to the Corollary 1. The complexity of that problem parameterized above the guaranteed values remains open.

We continue in the next section by proving, that the parameterized version of the NP-complete problem of switching to a $k$-regular graph is in FPT. The same is proven for another NP-complete problem of switching to graph with at most $k$ edges in the next section. We also show the tractability result for the problem of switching to an $H$-free graph. The last section contains some open problems.

## 2 Basic properties of Seidel's switching

Now we give a few basic results that we will use through the paper.

**Lemma 1.** *Let $G$ be a graph and $A, B \subset V$ two subsets of its vertices. Then*

- $S(G, A)[A] = G[A]$,
- $S(G, A) = S(G, V \setminus A)$,
- $S(G, \emptyset) = S(G, V) = G$,
- $S(S(G, A), B) = S(S(G, B), A) = S(G, A \triangle B)$,
- $S(\overline{S(G, A)}, A) = \overline{S(G, A)}$.

**Lemma 2.** *If $G$ is a graph, $v$ is its vertex and $X \subseteq V \setminus \{v\}$, then there is a unique graph $H \in [G]$ such that $N_H(v) = X$.*

**Lemma 3.** *Let $G = (V, E)$ be a graph on $n$ vertices. Then there is a graph $H \in [G]$ such that $\forall v \in V : deg_H(v) \geq \lfloor n/2 \rfloor$.*

**Corollary 1.** *It is trivially fixed-parameter tractable to decide whether a graph on the input can be switched to some graph that has all the degrees at least $k$.*

*Proof.* If $k < n/2$ then answer YES. Otherwise try all possibilities, this takes time at most $n^2 \cdot 2^{n-1} \leq k^2 \cdot 2^{2k}$.

## 3 Switching to a graph with all the vertices of degree at most $k$

The classical complexity of this problem is unknown. Indeed, ideas of the algorithm we present here are the crucial ones in the proofs of the tractability of other problems. Moreover, it is the best known algorithm for this problem. The exact specification of the problem is:

| Switching to a graph with all the vertices of degree at most $k$ | SwitchSmallDegs |
|---|---|
| **Input:** Graph $G = (V, E)$ | |
| **Parameter:** A positive integer $k$ — the desired maximal degree | |
| **Task:** Is there a subset of vertices $A \subseteq V$ such that the degree of each vertex in the switching of the set $A$ in a graph $G$ is at most $k$, i.e. $\forall v \in V : deg_{S(G,A)}(v) \leq k$? | |

Before we present an algorithm to solve the problem, observe that we can assume that the input graph contains an isolated vertex $v_0$. If it does not, we can switch it in linear time (in the number of the edges of the original graph) according to Lemma 2 to obtain an isolated vertex. The answer remains unchanged due to Lemma 1.

Since $A$ is a solution iff $V \setminus A$ is, we may assume without loss of generality, that $v_0 \notin A$. These two things together give us that $N_{S(G,A)}(v_0) = A$. Thus necessarily $|A| \leq k$.

During the run of the algorithm we denote by $l$ the number of the vertices we can still switch. At the beginning we set $l := k$.

Now we introduce two simple rules that solve the "big" instances:

**Lemma 4 (Rule 1).** *We must switch every vertex $v$ of degree greater than $k+l$ in order to obtain a solution.*

*Proof.* By contradiction. Suppose that $A$ determines a switching solving the problem and that $v \notin A$. Vertex $v$ has at least $k + l + 1$ neighbors in $G$. By switching $v$ could loose at most $|A|$ of them and thus at most $l$. So it still has at least $k + 1$ neighbors. Switching doesn't solve the problem, a contradiction.

**Lemma 5 (Rule 2).** *No vertex $v$ of degree less than $n - k - l$ can be switched to obtain a solution.*

This lemma is proved similarly as the previous one, so we omit the proof.

The following statement is an easy corollary of the previous two lemmas:

**Corollary 2 (Boundary).** *If $n \geq 2k + 2l + 1$, then at least one of the rules 1 and 2 applies on each vertex.*

That means we have already obtained a kernel of size $4k$ for the problem and thus the problem is in FPT. As we want to give a better bound on the running time of the algorithm, we concentrate on the instance with $0 \leq p \leq k - 1$ and $n = 2k + 2p + 2$ or $n = 2k + 2p + 1$. In this case, after switching $k - p$ vertices it is possible to decide which vertices should be switched and which should not. The algorithm proceeds as follows:

1. Set $l := k$
2. For every vertex $v$ (except $v_0$)
   - Check which of the rules 1 and 2 applies on $v$
   - If both rules apply answer NO and quit.
   - If the rule 1 applies switch $v$ and decrease $l$
   - If $l < 0$ answer NO and quit.
3. If $n > 4k$ then check if all the vertices have degree at most $k$ in this switching and answer.
4. If $n \leq 4k$ then try all the possibilities $A$ to choose up to $l - p$ vertices that can be switched and that were not switched already. For each choice of $A$ switch according to $A$, apply the rules once again and check whether we obtained the desired solution.

*Remark 1.* The algorithm can be also used to enumerate all the graphs with the desired property.

The correctness of the algorithm follows immediately from Lemmas 4, 5 and Corollary 2.

To count the running time of the algorithm we must bound the number $\sum_{i=0}^{k-p} \binom{n-1}{i}$ of the candidate sets in step 4. It is easy to observe that this is always at most $\binom{2k+2p+1}{k-p} \cdot k$, so we have to search the biggest number among

$$\binom{2k+1}{k}, \binom{2k+3}{k-1}, ..., \binom{2k+2p+1}{k-p}, ..., \binom{4k-3}{2}, \binom{4k-1}{1}.$$

By comparison of two neighboring terms we obtain a cubic equation with parameter $k$, with only one real root, which is asymptotically $p = \lceil 0.223k \rceil$. This gives $\binom{2\lceil 1.223k \rceil + 1}{\lfloor 0.777k \rfloor} \cdot k$ to be the biggest term.

We summarize our results in the following theorem:

**Theorem 1.** *The problem* SwitchSmallDegs *is linear fixed-parameter tractable and it can be solved in time* $\mathcal{O}^*\left(4.62^k\right)$.

*Proof.* A graph $G$ with an isolated vertex can be obtained in time $\mathcal{O}(m)$. Then we just test some candidate sets each of the size at most $k$. It takes $\mathcal{O}(k \cdot n)$ time to switch the graph according to the candidate set and $\mathcal{O}(n)$ time to check if it is a solution. If $n > 4k$ then we have at most one candidate. If $n \leq 4k$ then we can have up to $\binom{2\lceil 1.223k\rceil+1}{\lfloor 0.777k\rfloor} \cdot k$ candidates, as we have counted before. This grows approximately as $4.614^k$

## 4 Switching to a $k$-regular graph

| Switching to a $k$-regular graph | SwitchReg |
|---|---|
| **Input:** Graph $G = (V, E)$ | |
| **Parameter:** A positive integer $k$ — the desired degree of the vertices | |
| **Task:** Is there a subset of vertices $A \subseteq V$ such that the graph $S(G, A)$ is $k$-regular? | |

Kratochvíl [5] proved that the problem of SwitchReg is NP-complete, but it can be solved in a polynomial time for any fixed $k$. We show fixed-parameter tractability of this problem.

In an arbitrary $k$-regular graph each vertex has degree at most $k$, this means that we can use our previous algorithm to decide the problem. We just enumerate all the degree $\leq k$ graphs and check if some of them is $k$-regular. But we can slightly improve the algorithm.

We always have to switch exactly $k$ vertices in this case, so if there is a candidate set of size $k-p$ that leads to the solution then there is also one using just first $n-p-1$ vertices different from $v_0$. Using this idea we can bound the number of candidate tests by $\binom{n-p-1}{k-p} \leq \binom{2k+p+1}{k-p}$. To find the bound for this is somewhat easier than in the previous case, the answer being $\binom{\lceil 2.171k\rceil+1}{\lfloor 0.829k\rfloor}$ asymptotically (for $p = \lceil 0.171k\rceil$).

Our results are summarized by the following theorem:

**Theorem 2.** *The problem* SwitchReg *is linear fixed-parameter tractable and can be solved in time* $\mathcal{O}^*\left(4.263^k\right)$

## 5 Switching to a graph with at most $k$ edges

| Switching to a graph with at most $k$ edges | SwitchMinEdges |
|---|---|
| **Input:** Graph $G = (V, E)$ | |
| **Parameter:** A positive integer $k$ — the desired number of edges | |
| **Task:** Is there a subset of vertices $A \subseteq V$ such that the graph $S(G, A)$ has at most $k$ edges? | |

The problem SwitchMinEdges is NP-complete[6]. But since the desired graph has at most $k$ edges, each vertex has degree at most $k$. So the problem is

fixed-parameter tractable by similar arguments to the ones used for SwitchSmallDegs problem. We present much more efficient algorithm for this problem.

Following the way our previous algorithm works, we may suppose that our graph has an isolated vertex $v_0$ that is not switched in the solution sought. We denote by $l$ the upper bound for the number of vertices we can still switch and by $B$ the set of vertices already switched. At the beginning $B := \emptyset$ and during the entire algorithm we set $l := k - |B|$. We introduce two rules. They are more powerful and the proof is different.

**Rule 1**: Switch every vertex of degree greater than $l$.

**Lemma 6 (Correctness of Rule 1).** *Vertex $v \in V \setminus B \setminus \{v_0\}$ of degree $deg_{S(G,B)}(v) > l$ must be switched in order to obtain a solution.*

*Proof.* Suppose that $v \in V \setminus B \setminus \{v_0\}$ is a vertex of degree $d = deg_{S(G,B)}(v) > l$ and $A \subset V \setminus B \setminus \{v_0\}$ is a set, such that its switching in $S(G, B)$ gives a graph $H$. We show that if $v \notin A$, then the graph $H$ has too many edges. Denote $D = A \cap N_{S(G,B)}(v)$. Then $\{\{v_0, x\}|x \in D \cup B\}$ and $\{\{v, x\}|x \in N_{S(G,B)}(v) \setminus D\}$ are two disjoint sets of edges of the graph $H$ such that the size of their union is $|B| + |D| + |N_{S(G,B)}(v) \setminus D| = |B| + |N_{S(G,B)}(v)| = d + k - l > k$. Thus the graph $H$ has more than $k$ edges and the switching of the set $A$ does not lead to a solution.

**Lemma 7 (Rule 2).** *Vertex $v \in V \setminus B \setminus \{v_0\}$ of degree $deg_{S(G,B)}(v) < n - l - 1$ cannot be switched in order to obtain a solution.*

*Proof.* Let again $d = deg_{S(G,B)}(v) < n - l - 1$, $A \subset V \setminus B \setminus \{v_0\}$ and $H = S(G, A \cup B)$. Suppose $v \in A$. Then $\{\{v_0, x\}|x \in A \cup B \setminus \{v\}\}$ and $\{\{v, x\}|x \in V \setminus N_{S(G,B)}(v) \setminus A\}$ are two disjoint sets of edges of the graph $H$. The size of them together is at least $(|B| + |A| - 1) + (n - d - |A|) = k - l - 1 + n - d > k - l + n - (n - l - 1) - 1 = k$. Thus the graph $H$ has more than $k$ edges and the switching of the set $A$ does not lead to a solution.

**Lemma 8 (Boundary).** *If $n > 2l + 1$, then on each vertex either Rule 1 or Rule 2 applies.*

The rest of the algorithm remains almost unchanged, only the kernel bound is $2k$. Time complexity can be determined in a similar way as in previous cases giving us the following theorem:

**Theorem 3.** SwitchMinEdges *is fixed-parameter tractable and there is an algorithm running in time $\mathcal{O}^*\left(\binom{2\lceil 0{,}611k\rceil+1}{\lfloor 0{,}389k\rfloor}\right) \approx 2.148^k$.*

## 6 Switching to an $H$-free graph

There are many polynomial time algorithms known that decide whether the input graph can be switched to some containing no subgraph isomorphic to a fixed graph $H$. Most of them uses a reduction to 2-SAT, or to the system of linear

equations over GF[2] or a characterization by forbidden subgraphs. But these methods cannot be applied when the number of switches should be optimized.

We show that these problems are fixed-parameter tractable with the same parameter (i.e. the number of switches allowed). Before we state the general theorem, we show the method on the special case of the problem

| Switching to a triangle free graph — SwitchTriangle-Free |
|---|
| **Input:** Graph $G = (V, E)$ |
| **Parameter:** A positive integer $k$ — the number of switches allowed |
| **Task:** Is there a subset of vertices $A \subseteq V$ of size at most $k$ such that the graph $S(G, A)$ is triangle-free? |

Hayward et al. [7] and Hage et al. [3] independently presented two different algorithms recognizing the graphs that can be switched to triangle-free graphs. Both of them work in time $\mathcal{O}\left(n^3\right)$ by means of reduction to 2-SAT. In contrary, it is known that Weighted 2-SAT is NP-complete problem [1].

**Theorem 4.** *It is possible to decide the problem* SwitchTriangle-Free *in time* $\mathcal{O}\left(3^k \cdot n^3\right)$.

*Proof.* We use the bounded search tree technique. Our algorithm is recursive and each call gets on the input a graph $G$, a number $k$ and a set $A$ of already switched vertices.

At the beginning of each call, we search the graph $S(G, A)$ for a triangle. If there is none, we answer YES and the set $A$ determines the solution. Otherwise if $|A| = k$ or the graph $G[A] = S(G, A)[A]$ contains a triangle then answer NO. If that is not the case then pick an (arbitrary) triangle with the most number of vertices in $A$. For each of its vertices not in $A$ call the algorithm on triple $(G, k, A \cup \{v\})$ recursively. Answer YES if any of the calls answers so, taking the appropriate solution set and NO otherwise.

The result is obtained by the call on $(G, k, \emptyset)$.

Correctness and the time complexity of the algorithm are easy to check.

We generalize this result to an arbitrary set of forbidden induced subgraphs. Suppose that we are given a set $\mathcal{S} = \{H_1, H_2, ..., H_p\}$. We consider this problem:

| Switching to a graph with no induced subgraph from the set $\mathcal{S}$ Switch $\mathcal{S}$-free |
|---|
| **Input:** Graph $G = (V, E)$ |
| **Parameter:** A positive integer $k$ — the number of switches allowed |
| **Task:** Is there a subset of vertices $A \subseteq V$ of size at most $k$ such that the graph $S(G, A)$ contains no induced subgraph from the set $\mathcal{S}$ ? |

Denote by $l(\mathcal{S}) = \max\{|V_H||H \in \mathcal{S}\}$ the size of the biggest graph in the set $\mathcal{S}$.

**Theorem 5.** *For each finite set* $\mathcal{S}$ *it can be decided in time* $\mathcal{O}\left(l(S)^k \cdot n^{l(S)}\right)$ *whether the input graph* $G$ *can be switched to a graph containing no induced subgraph from the set* $\mathcal{S}$*. Hence the corresponding problem of* Switch $\mathcal{S}$-free *is fixed-parameter tractable for each finite set* $\mathcal{S}$ *of graphs.*

The proof is a slight modification of the ideas of the triangle-free case and we omit it here.

## 7 Switching to a bipartite graph

The last problem we concern is:

| Switching to a bipartite graph | SwitchBipartite |
|---|---|
| **Input:** Graph $G = (V, E)$ | |
| **Parameter:** A positive integer $k$ — the number of switches allowed | |
| **Task:** Is there a subset of vertices $A \subseteq V$ of size at most $k$ such that the graph $S(G, A)$ is bipartite? | |

This problem can be formulated as switching to a graph without odd cycles. As the set of forbidden graphs is infinite, our algorithm for Switch $\mathcal{S}$-free does not immediately apply. Hage et al. [3] showed an algorithm for the case of unbounded number of switches that works in time $\mathcal{O}\left(n^3\right)$ by a reduction to 2-SAT. This indicated that the bounded problem could be significantly harder. We show fixed-parameter tractability of the problem.

We start with some key observations: First of all, each graph that is not bipartite contains not only an odd cycle, but also an induced odd cycle. Hence it is sufficient to get rid of induced odd cycles. Moreover an odd cycle $C_n$ of length $n \geq 7$ cannot be switched to a bipartite graph [8]. Hence the graph that contains an odd cycle of length at least 7 as an induced subgraph, cannot be switched to a bipartite graph. So we deal almost only with triangles and 5-cycles.

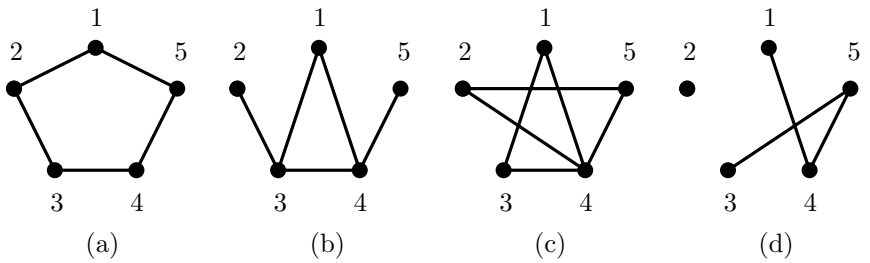



**Fig. 1.** A five-cycle (a), with one vertex (b), two adjacent (c) and two non-adjacent vertices (d) switched.

Moreover if we have a 5-cycle $(\{1, 2, 3, 4, 5\}, \{\{12\}, \{23\}, \{34\}, \{45\}, \{51\}\})$ then we must switch at least two vertices to obtain a bipartite graph (see Fig. 1). The sets $\{13\}, \{24\}, \{35\}, \{41\}, \{52\}$ and their complements determine the only suitable switches. Since all the three-element sets contain some of the two-element sets as a subset, we can switch always one of the five two-element sets when dealing with five-cycle, with possibly switching some more in some of the next steps. This significantly reduces the size of the search tree.

Now we introduce our recursive algorithm, leaving some details to the next paragraph:

Each call gets on the input a graph $G$, a number $k$ and a set $A$ of already switched vertices.

At the beginning of each call, we search the graph $S(G, A)$ for the shortest odd cycle (the shortest is always induced). If there is none, we answer YES and the set $A$ determines the solution. Otherwise if $|A| = k$ or the entire cycle is contained in the graph $G[A] = S(G, A)[A]$ then answer NO.

Otherwise continue according to what we found:

- Triangle: For each of the vertices of triangle not in $A$ call the algorithm on triple $(G, k, A \cup \{v\})$ recursively. Answer YES if any of the calls answers so, taking the appropriate solution set and NO otherwise.
- Five-cycle $\{a_1, \ldots, a_5\}$: Check whether $|A|$ is at most $k-2$. If it is not, answer NO. Otherwise for each of the sets $\{x, y\}$ from $\{a_1, a_3\}$, $\{a_2, a_4\}$, $\{a_3, a_5\}$, $\{a_4, a_1\}$ and $\{a_5, a_2\}$ that has empty intersection with $A$ call the algorithm on $(G, k, A \cup \{x, y\})$. Answer YES if any of the calls answers so, taking the appropriate solution set and NO otherwise.
- An induced odd cycle of length seven or more: Answer NO.

The result is obtained by the call on $(G, k, \emptyset)$.

The search for the shortest odd cycle can be done be running breath-first-search (BFS) from each vertex $u$ and looking for an edge between two vertices of same distance from $u$. When running from one vertex $u$ it is sufficient to use each edge at most twice. As there is at most $n^2$ edges, the search can be done in time $\mathcal{O}\left(n^3\right)$.

The only remaining thing to count the running time of the algorithm is the maximum number of the recursive calls. It is sufficient to count the number $f(k)$ of leaves of the search tree of height $k$. It holds that $f(0) = 1$ and we have two recurrences for the function $f(k)$:

$$f(k) = 3 \cdot f(k-1) \text{ for the triangle case}$$
$$f(k) = 5 \cdot f(k-2) \text{ for the five-cycle case}$$

The first recurrence gives the worse result of $f(k) = 3^k$.

We summarize the result:

**Theorem 6.** SwitchBipartite *can be solved in time* $\mathcal{O}\left(3^k \cdot n^3\right)$.

## 8 Open problems

The classical complexity of several problems in this area is still unknown. We mention at least one of them that is closely related to the problems we have solved.

**Switching to a $k$-degenerated graph** Kratochvíl [5] presented an $\mathcal{O}\left(n^{k+5}\right)$ time algorithm for this problem. If $k$ is a part of the input, no polynomial time algorithm is known. Fixed-parameter tractability would be a good alternative.

## Acknowledgments

The author would like to thank his advisor Jan Kratochvíl for his useful advices and tips and also Petr Hliněný for valuable discussion and suggestions.

## References

1. Downey, R.G., Fellows, M.R.: Parametrized Complexity. Monographs in Computer Science. Springer (1999)
2. Seidel, J.: Graphs and two-graphs. In: Proc. 5th Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Winnipeg, Canada, Utilitas Mathematica Publishing Inc. (1974)
3. Hage, J., Harju, T., Welzl, E.: Euler graphs, triangle-free graphs and bipartite graphs in switching classes. In: Proceedings of ICGT 2002. Volume 2505 of LNCS., Springer (2002) 148–160
4. Ondráčková, E.: Computational complexity in graph theory. Master's thesis, Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University in Prague (2006)
5. Kratochvíl, J.: Complexity of Hypergraph Coloring and Seidel's Switching. In Bodlaender, H.L., ed.: WG. Volume 2880 of Lecture Notes in Computer Science., Springer (2003) 297–308
6. Jelínková, E.: Three NP-Complete Optimization Problems in Seidel's Switching. In: Proceedings of IWOCA 2007
7. Hayward, R.B., Hougardy, S., Reed, B.A.: Polynomial time recognition of $P_4$-structure. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. (2002) 382–389
8. Hage, J., Harju, T.: Towards a characterization of bipartite switching classes by means of forbidden subgraphs. Technical Report UU-CS-2006-028, Institute of Information and Computing Sciences, Utrecht University (2006)

# Computing the Moments of Costs over the Solution Space of the TSP in Polynomial Time

Paul J. Sutcliffe, Andrew Solomon, and Jenny Edwards

Faculty of Information Technology
University of Technology, Sydney,
Sydney, Australia.
psutclif,andrews,jenny@it.uts.edu.au

**Abstract.** We give polynomial time algorithms to compute the third and fourth moments about the mean of tour costs over the solution space of the general symmetric Travelling Salesman Problem (TSP). These algorithms complement previous work on the population variance and provide a tractable method to compute the skewness and kurtosis of the probability distribution of tour costs. The methodology is generalisable to higher moments. Experimental evidence is given that suggests the skewness asymptotically approaches a limit point as the instance size is increased in several problem types.

## 1 Introduction

### 1.1 The TSP

The travelling salesman problem (TSP) is a classic problem in combinatorial optimization. Extensive references include [1–3]. Linear programming reductions are surveyed in [4] while the properties of frequently used local search heuristics are considered in [5]. It is natural to define the symmetric TSP in terms of a complete *undirected* graph $\Gamma = (V, E)$ with the vertices $V$ representing cities, and the edges $E$ representing the connections between cities. We label the set of $n$ vertices as $\{1, 2, \ldots, n\}$, and an $n$-cycle permutation of these is a *tour* or *solution*, $\pi$. The set of all tours, the *solution space*, is denoted $\Theta$. The distance between cities (or *cost* of an edge), is a function cost : $E \to \Re$ which we extend to the function $\Omega : \Theta \to \Re$, defined as the cost of a tour $\Omega(\pi) = \sum_{i=1}^{n} \text{cost}(\{\pi(i), \pi((i \mod n) + 1)\})$.

The TSP is to find some $n$-cycle permutation $\pi$ of $V$ for which $\Omega(\pi)$ is smallest. Such a permutation $\pi^*$ is called a *global minimum tour*. If there are $n$ cities then the number of tours is $|\Theta| = (n-1)!/2$.

### 1.2 Survey of Statistical Results

Previous theoretical work on the probability distribution of the TSP is surveyed in [6, 7], these largely concern the case of the Euclidean TSP with *city coordinates* as $n$ random variables in bounded subsets of $\Re^d$. Beardwood et. al. [8] prove that

$\Omega(\pi^*)$ approaches a constant as $n \to \infty$. Steele proves the variance of costs over the solution space is bounded [6]. Rhee and Talagrand prove that the tails of the cost distribution approach that of a Gaussian as the number of cities increases [9]. In the more general case Krauth and Mézard [10] extend the result of Beardwood et. al. to problems with uniform random *edge costs*. More recently Wästlund [11] extends it to the TSP on bipartite graphs with uniform random *edge costs*.

Basel et. al. [12] show by *random sampling* a remarkable linear correlation between the square root of a problem size and an *estimate* of the number of standard deviations between the mean tour cost and the known optimal tour cost in a real world set of approximately Euclidean problems. Sutcliffe et. al. [13] give a constructive proof that the population variance of tour costs over the solution space of an instance of size $n$ cities can be computed in $O(n^2)$, see Theorem 1 below. Applying this, they confirm the linear relationship found by Basel et. al. and show a similar, although non-linear, relationship in the case of a set of non-Euclidean real world problems.

### 1.3 Moments in Terms of the TSP

In terms of a TSP with solution space $\Theta$, cost function $\Omega$ and mean tour cost $\mu$, the $k$th *moment about the mean* or *central moment* [14] can be written

$$\mathrm{mm_k}(\Theta) = \frac{\sum\limits_{\pi \in \Theta} ((\Omega(\pi) - \mu)^k)}{|\Theta|}. \tag{1}$$

It is reported in [15](and a simple proof follows from Lemma 1) that the mean tour cost over the solution space of a problem of size $n$ cities with edge set $E$ is $\mu = \frac{2}{n-1} \sum\limits_{e \in E} \mathrm{cost}(e)$. The second moment or *population variance* is given by Theorem 1 below. Comparison of the second and third moments provides the well known statistic, the *skewness,* $\alpha_3(X) = \frac{\mathrm{mm}_3(X)}{\mathrm{mm}_2(X)^{3/2}}$, which reflects the degree of symmetry of a probability distribution [14].

**Theorem 1.** *The population variance of tour costs over the solution space of a TSP of size $n$ cities and with edge set, $E$ and vertex set $V$ is*

$$\mathrm{var} = \frac{2\beta_1}{(n-1)} - \frac{4\beta_1 + 2\beta_2}{(n-1)(n-2)} \tag{2}$$

*with the values $\beta_1, \beta_2$ being defined as*

$$\begin{aligned} \beta_1 &= \sum_{e \in E} c_0(e)^2 \\ \beta_2 &= \sum_{e=\{x,y\} \in E} [c_0(e)(S_x + S_y - 2c_0(e))] \end{aligned} \tag{3}$$

*where $c_0(e) = \mathrm{cost}(e) - \mu/n$, $I_x$ is the set of edges incident to a vertex $x$ with $S_x = \sum_{e \in I_x} c_0(e)$ and similarly for $S_y$.*

## 2 The Third and Fourth Moment of Costs over the Solution Space

We begin with a technical lemma providing the number of tours containing various configurations of edges. Table 1 enumerates the eleven cases to be used.

**Lemma 1.** *Given a TSP with graph $\Gamma$, let $P$ be a set of $m$, non-cyclic, non-singleton paths over $\Gamma$ sharing no vertices. Let $k$ be the number of vertices not appearing in any path of $P$. Then there are $2^{m-1}(k+m-1)!$ tours containing all the paths in $P$.*

*Proof.* Label the paths of $P$, $p_j$ with $j \in [1 \dots m]$. We recall that a tour is a cyclic permutation of vertices. Therefore, without loss of generality, fix $p_1$ in position and orientation and write a tour as $(p_1, i_1, i_2 \dots, i_q, p_2, i_{i_q+1} \dots, p_m \dots, i_k)$. There are $(k+m-1)!$ orderings of the free paths and vertices. Each path is at least 2 vertices long and so each of the $m-1$ free paths has 2 orientations, implying the result. □

**Table 1.** The eleven ways that, up to four unlabelled edges, can be arranged into paths in tours of size $n$. The $-$ character represent an edge, so $--$ means a path with 2 edges and three vertices. The $\rightsquigarrow$ symbol, the set (possibly empty) of free vertices between unconnected paths. The number of paths is given by $m$, while $k$ is the number of free vertices.

| case | pattern | m | k | num. tours | cities $n$ |
|---|---|---|---|---|---|
| 1 | $- \rightsquigarrow$ | 1 | $(n-2)$ | $(n-2)!$ | $n > 2$ |
| 2 | $-- \rightsquigarrow$ | 1 | $(n-3)$ | $(n-3)!$ | $n > 2$ |
| 3 | $- \rightsquigarrow - \rightsquigarrow$ | 2 | $(n-4)$ | $2(n-3)!$ | $n > 3$ |
| 4 | $--- \rightsquigarrow$ | 1 | $(n-4)$ | $(n-4)!$ | $n > 3$ |
| 5 | $-- \rightsquigarrow - \rightsquigarrow$ | 2 | $(n-5)$ | $2(n-4)!$ | $n > 4$ |
| 6 | $- \rightsquigarrow - \rightsquigarrow - \rightsquigarrow$ | 3 | $(n-6)$ | $4(n-4)!$ | $n > 5$ |
| 7 | $- --- \rightsquigarrow$ | 1 | $(n-5)$ | $(n-5)!$ | $n > 4$ |
| 8 | $--- \rightsquigarrow - \rightsquigarrow$ | 2 | $(n-6)$ | $2(n-5)!$ | $n > 5$ |
| 9 | $-- \rightsquigarrow -- \rightsquigarrow$ | 2 | $(n-6)$ | $2(n-5)!$ | $n > 6$ |
| 10 | $-- \rightsquigarrow - \rightsquigarrow - \rightsquigarrow$ | 3 | $(n-7)$ | $4(n-5)!$ | $n > 6$ |
| 11 | $- \rightsquigarrow - \rightsquigarrow - \rightsquigarrow - \rightsquigarrow$ | 4 | $(n-8)$ | $8(n-5)!$ | $n > 7$ |

### 2.1 Computing the Third Moment

In order to prove our central theorem we provide some notational machinery. Let $\Theta$ be the solution space of a TSP with edge set $E$ and cost function $\Omega$. We index each $\pi$ in $\Theta$ with an integer $m \in [1 \dots |\Theta|]$, similarly we label the edges of $E$ as

$e_i$ with $i \in [1 \ldots |E|]$. We define the function $[1 \ldots |\Theta|] \times [1 \ldots |E|] : t \to \{0, 1\}$ as

$$t_{mi} = \begin{cases} 1 & \text{if edge } e_i \text{ is in tour } m \\ 0 & \text{otherwise.} \end{cases}$$

Under this arrangement if $m$ is the index of a tour $\pi$ then the cost of $\pi$ is

$$\Omega(\pi) = t_{m1}\text{cost}(e_1) + t_{m2}\text{cost}(e_2) \ldots t_{m|E|}\text{cost}(e_{|E|}) \ ,$$

and specializing (1) to $k = 3$, the third moment about the mean $\mu$ is

$$\text{mm}_3(\Theta) = \frac{\sum_{m=1}^{|\Theta|} ((t_{m1}\text{cost}(e_1) + t_{m2}\text{cost}(e_2) \ldots t_{m|E|}\text{cost}(e_{|E|}) - \mu)^3)}{|\Theta|} \ . \quad (4)$$

Now $|\Theta|$ is, of course, factorial on $n$ and so this formulation is impractical for all but the smallest problems. In Theorem 2 we give a polynomial time solution to the problem.

Returning to notational matters, let $A_p$ be the set of edges adjacent to edge $e_p$. Let $N_{p,q,\ldots}$ be the set of edges neither adjacent to nor equal to edges $e_p, e_q, \ldots$, so $N_{p,q,\ldots} = E - (A_p \bigcup \{e_p\} \bigcup A_q \bigcup \{e_q\} \ldots)$.

**Theorem 2.** *The third moment about the mean of tour costs over the solution space of a TSP with $n > 3$ cities, mean tour cost $\mu$, and with edge set $E$ is*

$$\text{mm}_3 = \frac{2\gamma_1}{(n-1)} + \frac{2(\gamma_2 + 2\gamma_3)}{(n-1)(n-2)} + \frac{2(\gamma_4 + 2\gamma_5 + 4\gamma_6)}{(n-1)(n-2)(n-3)}$$

*with the values $\gamma_1, \gamma_2, \gamma_3\gamma_4, \gamma_5, \gamma_6$ given by*

$$\begin{aligned}
\gamma_1 &= \sum_{e \in E} c_0(e)^3 \\
\gamma_2 &= 3 \sum_{e_p \in E} c_0(e_p)^2 \sum_{e_q \in A_p} c_0(e_q) \\
\gamma_3 &= 3 \sum_{e_p \in E} c_0(e_p)^2 \sum_{e_q \in N_p} c_0(e_q) \\
\gamma_4 &= 3 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{e_r \in A_q - (A_p \bigcup \{e_p\})} c_0(e_r) \\
\gamma_5 &= 3 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{e_r \in N_{p,q}} c_0(e_r) \\
\gamma_6 &= \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in N_p} c_0(e_q) \sum_{e_r \in N_{p,q}} c_0(e_r)
\end{aligned}$$

*where $c_0(e) = \text{cost(e)} - \mu/n$.*

*Proof.* Consider (4). Each tour has only $n$ edges, so for a given $m$ there are just $n$ $t_{mi}$ which are equal to 1, the remainder being equal to 0. So let $c_0(e_i) = \text{cost}(e_i) - \mu/n$. Then (4) is written

$$\mathrm{mm}_3(\Theta) = \frac{\sum_{m=1}^{|\Theta|} ((t_{m1}c_0(e_1) + t_{m2}c_0(e_2)\ldots t_{m|E|}c_0(e_{|E|}))^3)}{|\Theta|} \ ,$$

$$= \frac{\sum_{m=1}^{|\Theta|}\sum_{k=1}^{|E|}\sum_{j=1}^{|E|}\sum_{i=1}^{|E|} t_{mi}t_{mj}t_{mk}c_0(e_i)c_0(e_j)c_0(e_k)}{|\Theta|} \ .$$

The product $t_{mi}t_{mj}t_{mk} = 1$, if and only if, tour $m$ contains the edges $e_i, e_j, e_k$ and there are six way in which this can occur,

**case 1** All of $e_i, e_j, e_k$ are equal. By Lemma 1 and Case 1 of Table 1 there are $(n-2)!$ tours containing the edge.

**case 2** Two of $e_i, e_j, e_k$ are equal and the third is adjacent. By Lemma 1 and Case 2 of Table 1 there are $(n-3)!$ tours containing the three edges so configured.

**case 3** Two of $e_i, e_j, e_k$ are equal and the third is non-adjacent to them. By Lemma 1 and Case 3 of Table 1 there are $2(n-3)!$ tours containing the 2 edges so configured.

**case 4** The three edges $e_i, e_j, e_k$ form a path. By Lemma 1 and Case 4 of Table 1 there are $(n-4)!$ tours containing the edges so configured.

**case 5** Two of $e_i, e_j, e_k$ are adjacent and the third is non adjacent to either. By Lemma 1 and Case 5 of Table 1 there are $2(n-4)!$ tours containing the three edges so configured.

**case 6** All $e_i, e_j, e_k$ are all non adjacent to each other. By Lemma 1 and Case 6 of Table 1 there are $4(n-4)!$ tours containing the three edges so configured.

For each of there six cases we write the sum of edge cost products as $\gamma_1$ to $\gamma_6$ in (2). Upon collecting like terms we have:

$$\begin{aligned}\mathrm{mm}_3(\Theta) =& ((n-2)!\gamma_1 + (n-3)!\gamma_2 + 2(n-3)!\gamma_3 \\ &+ (n-4)!\gamma_4 + 2(n-4)!\gamma_5 + 4(n-4)!\gamma_6)/|\Theta|\end{aligned}$$

$$= \frac{2((n-2)!\gamma_1 + (n-3)!(\gamma_2 + 2\gamma_3) + (n-4)!(\gamma_4 + 2\gamma_5 + 4\gamma_6))}{(n-1)!}$$

$$= \frac{2\gamma_1}{(n-1)} + \frac{2(\gamma_2 + 2\gamma_3)}{(n-1)(n-2)} + \frac{2(\gamma_4 + 2\gamma_5 + 4\gamma_6)}{(n-1)(n-2)(n-3)} \ .$$

as required. □

### 2.2 Reducing the Computational Complexity of Third Moment

The set $A_p$ is $O(n)$ in size, while the sets $E, N_p, N_{p,q}$ are all $O(n^2)$ in size. This implies that a naive application of Theorem 2 above would have complexity $O(n^6)$, being that of the sum $\gamma_6$. Here we show that this can be reduced to $O(n^4)$. Let $I_x$ be the set of edges incident the vertex $x$ and let $S_x = \sum_{e \in I_x} c_0(e)$, be the sum of edge costs incident to $x$. Now $|I_x| = n-1$, so the time complexity of pre-computing all the $n$ values $S_x$ is $O(n^2)$ and the space complexity of saving them is $O(n)$.

**Lemma 2.** *$\gamma_2$ can be found in $O(n^2)$*

*Proof.* Recall that $\gamma_2 = 3 \sum_{e_p \in E} c_0(e_p)^2 \sum_{e_q \in A_p} c_0(e_q)$. Consider the right most sum on $A_p$. We show this can be found in constant time. Writing each edge $e_p$, as $e_p = \{p_1, p_2\}$ and noting that $A_p = (I_{p1} \bigcup I_{p2}) - \{e_p\}$ gives,

$$\begin{aligned}\gamma_2 &= 3 \sum_{e_p \in E} c_0(e_p)^2 (S_{p1} + S_{p2} - 2c_0(e_p)) \\ &= 6\gamma_1 + 3 \sum_{e_p \in E} c_0(e_p)^2 (S_{p1} + S_{p2}) \ .\end{aligned}$$

This along with $|E| \in O(n^2)$ implies the result. □

**Lemma 3.** $\gamma_3 = -\gamma_2 - 3\gamma_1$

*Proof.* Recall that $\gamma_3 = 3 \sum_{e_p \in E} c_0(e_p)^2 \sum_{e_q \in N_p} c_0(e_q)$. Consider the right most sum, $N_p = E - (A_p \bigcup \{e_p\})$. So $\sum_{e \in N_p} c_0(e) = \sum_{e \in E} c_0(e) - \sum_{e \in A_p} c_0(e) - c_0(e_p)$, but $\sum_{e \in E} c_0(e) = 0$ thus

$$\begin{aligned}\gamma_3 &= 3 \sum_{e_p \in E} c_0(e_p)^2 \left[ - \sum_{e_q \in A_p} c_0(e_q) - c_0(e_p) \right] \\ &= -3 \sum_{e_p \in E} c_0(e_p)^2 \sum_{e_q \in A_p} c_0(e_q) - 3 \sum_{e_p \in E} c_0(e_p)^2 c_0(e_p) \\ &= -\gamma_2 - 3\gamma_1 \ .\end{aligned}$$

As required. □

**Lemma 4.** *$\gamma_4$ can be found in $O(n^3)$*

*Proof.* Recall that $\gamma_4 = 3 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{e_r \in A_q - (A_p \bigcup \{e_p\})} c_0(e_r)$. We show that the right most sum can be found in constant time given an $e_p$ and $e_q$. Let $e_p = \{s, p\}$, let $e_q = \{s, q\}$ be adjacent to it, sharing vertex $s$, and let $e_{pq} = \{p, q\}$ be adjacent to both. In addition let $I_q$ be the sets of edges incident to vertex $q$ and let $S_q$ be the pre-computed edge sum. Then $A_q - (A_p \bigcup \{e_p\}) = I_q - (\{e_q\} \bigcup \{e_{pq}\})$ and

$$\gamma_4 = 3 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q)(S_q - c_0(e_q) - c_0(e_{pq})) \ .$$

This along with $|E| \in O(n^2)$ and $|A_p| \in O(n)$ implies the result. □

**Lemma 5.** *$\gamma_5$ can be found in $O(n^3)$*

*Proof.* Recall that $\gamma_5 = 3 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{e_r \in N_{p,q}} c_0(e_r)$ We show that the right most sum can be found in constant time.

Let $e_p = \{s, p\}$, let $e_q = \{s, q\}$ be adjacent to it, sharing vertex $s$, and let $e_{pq} = \{p, q\}$ be adjacent to both. In addition let $I_s, I_p, I_q$ be the sets of edges incident to the vertices $s, p, q$ respectively and let $S_s, S_p, S_q$ be the pre-computed edge sums. Now $N_{p,q} = E - (I_s \bigcup I_p \bigcup I_q)$, but $\sum_{e \in E} c_0(e) = 0$ and the edges $e_{pq}, e_q, e_p$ are each elements of two of $I_s, I_p, I_q$ so, $\sum_{e_r \in N_{p,q}} c_0(e_r) = c_0(e_p) + c_0(e_q) + c_0(e_{pq}) - S_s - S_p - S_q$ and

$$\begin{aligned}\gamma_5 &= 3 \sum_{e_p \in E} \left[ c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \left[ c_0(e_p) + c_0(e_q) + c_0(e_{pq}) - S_s - S_p - S_q \right] \right] \\ &= 6\gamma_2 + 3 \sum_{e_p \in E} \left[ c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \left[ c_0(e_{pq}) - S_s - S_p - S_q \right] \right] \quad .\end{aligned}$$

This along with $|E| \in O(n^2)$ and $|A_p| \in O(n)$ implies the result □

**Lemma 6.** *$\gamma_6$ can be found in $O(n^4)$*

*Proof.* Recall that $\gamma_6 = \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in N_p} c_0(e_q) \sum_{e_r \in N_{p,q}} c_0(e_r)$. We show that the right most sum can be found in constant time and the that the middle sum can be rewritten over $A_p$. Let $e_p = \{p_1, p_2\}$ and let $e_q = \{q_1, q_2\}$ be non adjacent. In addition let $I_{p1}, I_{p2}, I_{q1}, I_{q2}$ be the sets of edges incident to these vertices and let $S_{p1}, S_{p2}, S_{q1}, S_{q2}$ be the pre-computed edge sums. Now $N_{p,q} = E - (I_{p1} \bigcup I_{p2} \bigcup I_{q1} \bigcup I_{q2})$, but $\sum_{e \in E} c_0(e) = 0$ and the edges $e_p, e_q, \{p_1, q_1\}$, $\{p_1, q_2\}, \{p_2, q_1\}, \{p_2, q_2\}$ are each elements of two of $I_{p1}, I_{p2}, I_{q1}, I_{q2}$. Therefore write $S_{Np,q} = \sum_{e \in N_{p,q}} c_0(e) = -S_{p1} - S_{p2} - S_{q1} - S_{q2} + c_0(e_p) + c_0(e_q) + c_0(\{p_1, q_1\}) + c_0(\{p_1, q_2\}) + c_0(\{p_2, q_1\}) + c_0(\{p_2, q_2\})$ and

$$\gamma_6 = \sum_{e_p \in E} \left[ c_0(e_p) \sum_{e_q \in N_p} c_0(e_q) S_{Np,q} \right],$$

as required. □

**Theorem 3.** *The complexity of computing the third moment about the mean of tour costs over the solution space of a TSP with n cities is $O(n^4)$.*

*Proof.* This follows directly from Theorem 2, the comments at the beginning of Sect. 2.2, and Lemmas 2 to 6. □

### 2.3 Computing the Fourth Moment about the Mean of Tour Costs

**Theorem 4.** *The fourth moment about the mean of tour costs over the solution space of a TSP with mean tour cost $\mu$, size $n > 5$ cities and with edge set $E$ is*

$$\begin{aligned}\mathrm{mm}_4 = \frac{2\delta_1}{(n-1)} &+ \frac{2(\delta_{2a}+\delta_{2b}+2\delta_{3a}+2\delta_{3b})}{(n-1)(n-2)} \\ &+ \frac{2(\delta_{4a}+\delta_{4b}+2\delta_{5a}+2\delta_{5b}+4\delta_6)}{(n-1)(n-2)(n-3)} \\ &+ \frac{2(\delta_7+2\delta_8+2\delta_9+4\delta_{10}+8\delta_{11})}{(n-1)(n-2)(n-3)(n-4)} \quad ,\end{aligned}$$

*with the values* $\delta_1, \delta_{2a}, \delta_{2b}, \delta_{3a}, \delta_{3b}, \delta_{4a}, \delta_{4b}, \delta_{5a}, \delta_{5b}, \delta_6, \delta_7, \delta_8, \delta_9, \delta_{10}\delta_{11}$ *given by*

$$\begin{aligned}
\delta_1 &= \sum_{e\in E} c_0(e)^4 \\
\delta_{2a} &= 3 \sum_{e_p\in E} c_0(e_p)^2 \sum_{e_q\in A_p} c_0(e_q)^2 \\
\delta_{2b} &= 4 \sum_{e_p\in E} c_0(e_p)^3 \sum_{e_q\in A_p} c_0(e_q) \\
\delta_{3a} &= 3 \sum_{e_p\in E} c_0(e_p)^2 \sum_{e_q\in N_p} c_0(e_q)^2 \\
\delta_{3b} &= 4 \sum_{e_p\in E} c_0(e_p)^3 \sum_{e_q\in N_p} c_0(e_q) \\
\delta_{4a} &= 12 \sum_{e_p\in E} c_0(e_p)^2 \sum_{e_q\in A_p} c_0(e_q) \sum_{\substack{e_r\in A_q,\\ e_r\notin A_p,\\ e_r\neq e_p}} c_0(e_r) \\
\delta_{4b} &= 6 \sum_{e_p\in E} c_0(e_p) \sum_{e_q\in A_p} c_0(e_q)^2 \sum_{\substack{e_r\in A_q,\\ e_r\notin A_p,\\ e_r\neq e_p}} c_0(e_r) \\
\delta_{5a} &= 12 \sum_{e_p\in E} c_0(e_p)^2 \sum_{e_q\in A_p} c_0(e_q) \sum_{e_r\in N_{p,q}} c_0(e_r) \\
\delta_{5b} &= 6 \sum_{e_p\in E} c_0(e_p) \sum_{e_q\in A_p} c_0(e_q) \sum_{e_r\in N_{p,q}} c_0(e_r)^2 \\
\delta_6 &= 6 \sum_{e_p\in E} c_0(e_p)^2 \sum_{e_q\in N_p} c_0(e_q) \sum_{e_r\in N_{p,q}} c_0(e_r)
\end{aligned}$$

$$\delta_7 = 12 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{\substack{e_r \in A_q, \\ e_r \notin A_p, \\ e_r \neq e_p}} c_0(e_r) \sum_{\substack{e_s \in A_r, \\ e_s \notin A_q, \\ e_s \notin A_p}} c_0(e_s)$$

$$\delta_8 = 12 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{\substack{e_r \in A_q, \\ e_r \notin A_p, \\ e_r \neq e_p}} c_0(e_r) \sum_{e_s \in N_{p,q,r}} c_0(e_s)$$

$$\delta_9 = 3 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{e_r \in Np,q} c_0(e_r) \sum_{\substack{e_s \in A_r, \\ e_s \in N_{p,q}}} c_o(e_s)$$

$$\delta_{10} = 6 \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in A_p} c_0(e_q) \sum_{e_r \in Np,q} c_0(e_r) \sum_{e_s \in N_{p,q,r}} c_0(e_s)$$

$$\delta_{11} = \sum_{e_p \in E} c_0(e_p) \sum_{e_q \in N_p} c_0(e_q) \sum_{e_r \in Np,q} c_0(e_r) \sum_{e_s \in N_{p,q,r}} c_0(e_s)$$

*where* $c_0(e) = \text{cost(e)} - \mu/n$.

*Proof.* Specializing (1) to $k = 4$, and proceeding as we did for the third moment we have

$$\text{mm}_4(\Theta) = \frac{\sum_{m=1}^{|\Theta|} \sum_{l=1}^{|E|} \sum_{k=1}^{|E|} \sum_{j=1}^{|E|} \sum_{i=1}^{|E|} t_{mi} t_{mj} t_{mk} t_{ml} \text{c}_0(e_i) \text{c}_0(e_j) \text{c}_0(e_k) \text{c}_0(e_l)}{|\Theta|} \quad .$$

The product $t_{mi}t_{mj}t_{mk}t_{ml} = 1$ if and only if tour $m$ contains the edges $e_i, e_j, e_k, e_l$ and there are eleven ways in which this can occur.

**case 1** All of $e_i, e_j, e_k, e_l$ are equal. By Lemma 1 there are $(n-2)!$ tours containing the edge. The value $\delta_1$ is the sum of terms in this case.

**case 2** From $e_i, e_j, e_k, e_l$ there are 2 distinct edges and they form a path. By Lemma 1 there are $(n-3)!$ tours containing the edges. The values $\delta_{2a}, \delta_{2b}$ are the sums of terms in this case.

**case 3** From $e_i, e_j, e_k, e_l$ there are 2 distinct edges and they are non adjacent. By Lemma 1 there are $2(n-3)!$ tours containing the edges. The values $\delta_{3a}, \delta_{3b}$ are the sums of terms in this case.

**case 4** From $e_i, e_j, e_k, e_l$ there are 3 distinct edges and they form a path. By Lemma 1 there are $(n-4)!$ tours containing the edges. The values $\delta_{4a}, \delta_{4b}$ are the sums of terms in this case.

**case 5** From $e_i, e_j, e_k, e_l$ there are 3 distinct edges two of which form a path, the third is non adjacent. By Lemma 1 there are $2(n-4)!$ tours containing the edges. The values $\delta_{5a}, \delta_{5b}$ are the sums of terms in this case.

**case 6** From $e_i, e_j, e_k, e_l$ there are 3 distinct. All are non adjacent. By Lemma 1 there are $4(n-4)!$ tours containing the edges. The value $\delta_6$ is the sum of terms in this case.

**case 7** Each of $e_i, e_j, e_k, e_l$ are distinct and form a path. By Lemma 1 there are $(n-5)!$ tours containing the edges. The value $\delta_7$ is the sum of terms in this case.

**case 8** Each of $e_i, e_j, e_k, e_l$ are distinct, 3 form a path, the other is non adjacent. By Lemma 1 there are $2(n-5)!$ tours containing the edges. The value $\delta_8$ is the sum of terms in this case.

**case 9** Each of $e_i, e_j, e_k, e_l$ are distinct and form 2 non adjacent paths of 2 edges. By Lemma 1 there are $2(n-5)!$ tours containing the edges. The value $\delta_9$ is the sum of terms in this case.

**case 10** Each of $e_i, e_j, e_k, e_l$ are distinct. Two are adjacent. The remaining are non adjacent. By Lemma 1 there are $4(n-5)!$ tours containing the edges. The value $\delta_{10}$ is the sum of terms in this case.

**case 11** Each of $e_i, e_j, e_k, e_l$ are distinct. All are non adjacent. By Lemma 1 there are $8(n-5)!$ tours containing the edges. The value $\delta_{11}$ is the sum of terms in this case.

For each of these cases we write the sum of edge cost products as $\delta_1$ to $\delta_{11}$ in (4). Upon collecting like terms we have

$$\begin{aligned}\mathrm{mm}_4(\Theta) = & \tfrac{(n-2)!\delta_1}{|\Theta|} \\ & + \tfrac{(n-3)!\delta_{2a}+(n-3)!\delta_{2b}+2(n-3)!\delta_{3a}+2(n-3)!\delta_{3b}}{|\Theta|} \\ & + \tfrac{(n-4)!\delta_{4a}+(n-4)!\delta_{4b}+2(n-4)!\delta_{5a}+2(n-4)!\delta_{5b}+4(n-4)!\delta_6}{|\Theta|} \\ & + \tfrac{(n-5)!\delta_7+2(n-5)!\delta_8+2(n-5)!\delta_9+4(n-5)!\delta_{10}+8(n-5)!\delta_{11}}{|\Theta|} \ .\end{aligned}$$

Recall $|\Theta| = (n-1)!/2$ so upon cancelation we have the result. □

## 3 Empirical Examination of the Relationship between Skewness and Problem Size

We examine four problem sets, two real world and two randomly generated. The four types are summarized in Table 2.

**Table 2.** Problem types

| ProblemType | Size in Cities | Cases | Problem Description |
|---|---|---|---|
| Random Euclidean | 10-1000 | 21 | 2 Euclidean Metric of TSPLIB [16]. |
| VLSI | 131-984 | 10 | 2 Euclidean Metric of TSPLIB. |
| Random no embed. | 10-1000 | 21 | Random integer edge costs from $U(0, 999)$ |
| RH Data | 68-662 | 39 | Non Euclidean. Genomics problems. |

Of the real world sets the first set originated in the production of very large scale integrated circuits (VLSI) and uses the 2 dimensional Euclidean metric of [16]. The second set, of 39 instances, approximately obey the triangular inequality, but are non-Euclidean. They originate in the genomics community and arise from physical mapping of canine DNA by the radiation-hybrid (RH) method. The specific data set used was obtained from the RHDF9000 dog radiation hybrid panel[17].

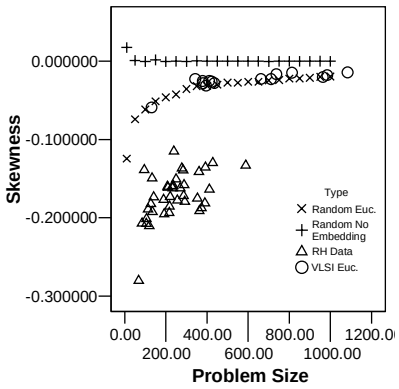



**Fig. 1.** The skewness versus the problem size in four problem types.

### 3.1 Results

The skewness of each instance was found using Theorems 1 and 2 in conjunction with Lemmas 2 to 6. Figure 1 shows its relationship to problem size. The relationship suggests, that in the case of the non RH data sets the skewness asymptotically approaches 0 with size. The RH data set is somewhat suggestive of convergence but to a lower limit point.

## 4 Conclusions and Future Work

In this paper we have given constructive proofs that the third central moment of tour costs over the solution space of any instance of a TSP of size $n$ cities can be computed in $O(n^4)$ and the that fourth central moment can be computed in $O(n^8)$. Experience with the third moment would suggest this computational complexity may be reduced to $O(n^6)$.

The method can be generalised to higher moments (at increased cost) and to variations of the problem such as the asymmetrical TSP.

Previous theoretical work on the probability distribution of the TSP was largely confined to the Euclidean case and did not extend to providing the moments. Future work will investigate the role of the third and fourth moments in refining current methods to estimate the optimal solution cost and to understanding the solution space of the problem.

Experimental evidence is given suggesting that the skewness asymptotically approaches 0 as the problem size is increased, in randomly generated non-embeddable and both random and real world 2 dimensional Euclidean instances. This implies that in these problem types, the distribution of tour costs become more symmetric as the problem size increases. This may make it possible to find bounds on the value of the odd moments of the cost distribution in certain classes of problem.

**Acknowledgements**

The authors would like to thank Andre Rohe, Simon de Givry, Thomas Schiex, Christophe Hitte and Jill Maddox for their assistance in obtaining the real world data sets.

## References

1. Gutin, G., Punnen, A.P.: Traveling Salesman Problem and Its Variations. Kluwer Academic Publishers (2002)
2. Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., Kann, V.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1999)
3. Reinelt, G.: The traveling salesman: Computational solutions for TSP applications. Springer Verlag (1994) LNCS 840.
4. Orman, A.J., Williams, H.P.: A survey of different integer programming formulations of the travelling salesman problem. Working Paper LSEOR 04.67, Department of Operational Research, London School of Economics and Political Science, London (2004)
5. Colletti, B., Barnes, J.: Local search structure in the symmetric travelling salesperson problem under a general class of rearrangement neighborhoods. Applied Mathematics Letters. **14**(1) (2001) 105–108
6. Steele, J.M.: Probability Theory and Combinatorial Optimization. SIAM (1997)
7. Yukich, J.E.: Probability Theory of Classical Euclidean Optimization Problems. Volume 1675 of Lecture Notes in Mathematics. Springer (1998)
8. Beardwood, J., Halton, J., Hammersley, J.: The shortest path through many points. Proc. Cambridge Philos. Soc. **55** (1959) 299–327
9. Rhee, W.T., Talgrand, M.: A sharp deviation inequality for the stochastic traveling salesman problem. Annals of Probability **17** (1989) 1–8
10. Krauth, W., Mézard, M.: The cavity method and the travelling-salesman problem. Europhys. Lett. **8**(3) (1988)
11. Wästlund, J.: The limit in the mean field bipartite travelling salesman problem. unpublished 2006, wwww.mai.liu.se/jowas
12. John Basel, I., Willemain, T.R.: Random tours in the traveling salesman problem analysis and application. Comput. Optim. Appl. **20**(2) (2001) 211–217
13. Sutcliffe, P., Solomon, A., Edwards, J.: Finding the population variance of costs over the solution space of the tsp in polynomial time. In Psarris, K., Jones, A.D., eds.: Math 07, Proceedings of the Eleventh WSEAS International Conference on Applied Mathematics, WSEAS (March 22-24, 2007) 23–28
14. Freund, J.E.: Mathematical Statistics. First edn. Prentice Hall (1972)
15. Punnen, A., Margot, F., Kabadi, S.: Tsp heuristics: Domination analysis and complexity. Technical report, Dept. of Mathematics, Univ. of Kentucky (2001)
16. Reinelt, G.: TSPLIB — a traveling salesman problem library. ORSA Journal on Computing **3**(4) (1991) 376–384
17. Faraut, T., de Givry, S., Chabrier, P., Derrien, T., Galibert, F., Hitte, C., Schiex, T.: A comparative genome approach to marker ordering. In: Proc. of ECCB-06. (2007) 7p.

# Vertex Coloring of Chordal+$k_1$e−$k_2$e Graphs

Yasuhiko Takenaga and Yusuke Miura*

The University of Electro-Communications, Chofu, Tokyo, Japan
takenaga@cs.uec.ac.jp

**Abstract.** $\mathcal{F}+k$e ($\mathcal{F}-k$e resp.) graph is the classes of graphs which can be obtained by adding (deleting resp.) at most $k$ edges to a graph in graph class $\mathcal{F}$. Complexity of the problems on these graphs can be measured using $k$ which is a parameter to represent the closeness to graph class $\mathcal{F}$. In this paper, we consider chordal+$k_1$e−$k_2$e graphs which are the classes of graphs obtained by adding and deleting edges at the same time from chordal graphs. We show an algorithm for vertex coloring of chordal+$k_1$e−$k_2$e graphs and prove that it is fixed parameter tractable.

## 1 Introduction

Many graph problems are NP-complete for general graphs. It is natural to consider that if the graph problem is tractable for a graph class $\mathcal{F}$, it is also tractable for a class of graphs which are close to graphs in $\mathcal{F}$. The classes of graphs obtained by adding or deleting edges or vertices to an $\mathcal{F}$-graph are very natural classes which are close to $\mathcal{F}$-graphs. $\mathcal{F}+k$e ($\mathcal{F}-k$e resp.) graphs are the classes of graphs obtained by adding (deleting resp.) $k$ edges to $\mathcal{F}$-graphs. Similarly, $\mathcal{F}+k$v ($\mathcal{F}-k$v resp.) graphs are the classes of graphs obtained by adding (deleting resp.) $k$ vertices to $\mathcal{F}$-graphs. Recently, complexity of several problems on such parameterized graph classes has been interested in from the viewpoint of parameterized complexity [2, 5, 7]. In general, problems become difficult as $k$ increases. A problem with parameter $k$ is called to be fixed parameter tractable if it can be solved in $f(k)n^c$ time, where $f$ is an arbitrary function depending only on $k$ and $n$ is the size of instance[3].

Though such parameterized graph classes has been studied recently, only the addition or deletion of edges (or vertices) to $\mathcal{F}$-graphs has been considered. It has not been considered yet to add and delete edges at the same time. In this paper, we first consider $\mathcal{F}+k_1$e−$k_2$e graphs, which is the class of graphs obtained by adding at most $k_1$ edges and deleting at most $k_2$ edges from $\mathcal{F}$-graphs. We call the added edges to be the plus modulators, and the removed edges to be the minus modulators. The modulators are the union of plus and minus modulators.

In this paper, we consider vertex coloring problem of chordal+$k_1$e−$k_2$e graphs. For fixed $k_1$ and $k_2$, chordal+$k_1$e−$k_2$e graphs can be recognized in polynomial time by checking exhaustively all the possibilities of the plus and minus modulators. However, it is not known if it is fixed parameter tractable or not. Thus, we assume that chordal+$k_1$e−$k_2$e graphs are given with their modulators.

* Presently with SIC CO.,LTD.

In this paper, we show that vertex coloring problem of chordal+$k_1$e−$k_2$e graphs is fixed parameter tractable for parameters $k_1$ and $k_2$, that is, solved in $f(k_1, k_2)n^c$ time. When only the addition or deletion of edges are made to chordal graphs, it is already known that vertex coloring is fixed parameter tractable. For chordal+$k$e graphs, Marx showed a fixed parameter tractable algorithm[7]. For chordal−$k$e graphs, it is known that vertex coloring of $\mathcal{F}$-$k$e graphs is fixed parameter tractable if $\mathcal{F}$ is closed under edge contraction[2]. Our algorithm for coloring chordal+$k_1$e−$k_2$e graphs is based on the idea of the algorithm of [7] and operations to deal with minus modulator are added.

This paper is organized as follows. Section 2 gives the definition of tree decomposition. In Section 3, we overview the vertex coloring algorithm for chordal+$k$e graphs. In Section 4, we show that vertex coloring of chordal+$k_1$e−$k_2$e graphs is fixed parameter tractable.

## 2 Tree Decomposition

A graph which does not contain an induced cycle of length four or larger is called a chordal graph. In this section, we give the characterization of chordal graphs in relation with the tree decomposition[4].

**Theorem 1.** *The following two statements are equivalent.*

- *$G(V, E)$ is a chordal graph.*
- *There exist a tree $T(U, F)$ and a subtree $T_v \subseteq T$ for each $v \in V$ such that $(u, v) \in E$ iff $T_u \cap T_v \neq \emptyset$.*

The tree $T$ (with its subtrees $T_v$) is called the tree decomposition of $G$. A tree decomposition of a chordal graph can be found in linear time [1]. Note that we use the term 'node' for a decomposition tree and 'vertex' for a graph to be colored.

For a node $x \in U$, let $V_x$ be the set of vertices of $G$ whose corresponding subsets include $x$ or the descendants of $x$. Let the subgraph of $G$ induced by $V_x$ be $G_x = G[V_x]$. For a node $x \in U$, let $K_x$ be the set of vertices $v$ such that $x$ is a node in $T_v$. We consider $T$ as a rooted tree with some root.

A tree decomposition that satisfies the following conditions are called a nice tree decomposition [6]. A nice tree decomposition can be obtained from a tree decomposition in polynomial time.

- Each node $x \in U$ has at most two children.
- If $x \in U$ has two children $y, z$, $K_x = K_y = K_z$ holds. In this case, $x$ is called a join node.
- If $x \in U$ has only one child $y$, either $K_x = K_y \cup \{v\}$ or $K_x = K_y \backslash \{v\}$ holds for some $v \in V$. In the former case, $x$ is called an introduce node, and in the latter case, $x$ is called a forget node.
- If $x \in U$ has no child, $K_x$ consists of exactly one vertex. In this case, $x$ is called a leaf node.

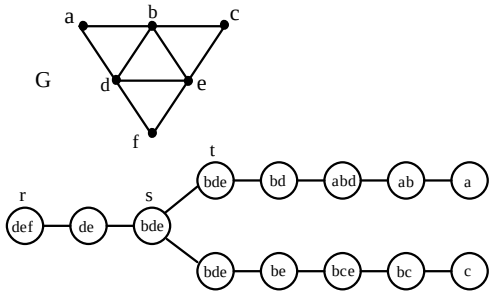



**Fig. 1.** A chordal graph and its nice tree decomposition.

An example of a chordal graph and its nice tree decomposition are shown in Fig.1. Node $r$ is the root of the decomposition tree. The vertices in $K_x$ are shown in each node $x$. $K_s = K_t = \{b, d, e\}$ as shown in the figure. $V_s = \{a, b, c, d, e\}$ and $V_t = \{a, b, d, e\}$.

## 3 Vertex Coloring of Chordal+$k$e Graphs

Our algorithm for coloring chordal+$k_1$e$-k_2$e graphs is based on the algorithm for coloring chordal+$k$e graphs proposed by Marx[7]. So, we briefly explain the algorithm in this section. Though the notations for our algorithm defined in the next section is similar to the ones used in the algorithm, many of them are modified to deal with minus modulators. Thus, to avoid confusion, we give only a few definitions in this section and describe only the outline of the algorithm.

Let $H$ be a chordal+$k$e graph, and $G$ be a chordal graph obtained by removing modulators from $H$. Let $H_x$ be the subgraph of $H$ induced by $V_x$. The algorithm is based on the nice tree decomposition of chordal graphs. In the algorithm, all the possible $C$-colorings of the subgraph $H_x$ are listed for each node $x$ of the decomposition tree. A coloring of $H_x$ is represented by a set $S$ of pairs of vertices with the same color. The set system $\mathcal{S}_x$ is the collection of sets of pairs which have the corresponding $C$-coloring for $H_x$

The listing of $C$-colorings starts from the leaves of the decomposition tree and proceeds to their parents. In the algorithm, when we deal with node $x$, $C - |K_x|$ dummy vertices $u_1, \cdots, u_{C-|K_x|}$ are added to $K_x$ so as to make a $C$-clique consisting of $K_x$ and the dummy vertices. Let $H_x^*$ be the graph that consists of $H_x$ and the dummy vertices.

On each node of the decomposition tree, the $C$-colorings for $H_x^*$ is computed from the $C$-colorings of the subgraphs corresponding to the children of $x$. The operation to list the colorings depends on the kind of the node; leaf node, introduce node, forget node or join node. When vertex $v$ is introduced, a dummy vertex is replaced by $v$ in the sets of pairs. When vertex $v$ is forgotten, $v$ is replaced by a dummy vertex. Then for each set $S$ and each $i$, set $S[u_i]$ that is

obtained by exchanging $u_i$ and $u_{C-|K_x|}$ is added to the set system. On a join node, by merging the sets for its children, all the possible colorings obtained by joining the colorings of two subgraphs are listed. In addition, when a modulator first appears in $H_x^*$ on an introduce node or a join node, the colorings that have the same color in the endpoints of the modulator must be omitted. For the root $r$ of the decomposition tree, $H_r$ equals $H$. Therefore, $H$ is $C$-colorable iff there exists a $C$-coloring when the listing finishes at the root.

The number of sets in the set system may become too large. However, instead of having all the possible $C$-colorings, it is sufficient to have its representative sets[8]. By using representative sets, the number of sets is reduced in small computational time. Hence the algorithm is fixed parameter tractable.

## 4 Vertex Coloring of Chordal+$k_1$e−$k_2$e Graphs

In this section, we propose an algorithm for coloring chordal+$k_1$e−$k_2$e graphs.

We call an endpoint of a plus modulator be a plus special vertex, and that of a minus modulator be a minus special vertex. Let the set of plus special vertices and minus special vertices be $W^+$ and $W^-$, respectively. Here, $|W^+| \leq 2k_1$ and $|W^-| \leq 2k_2$ hold. Let $W_x^+$ ($W_x^-$ resp.) denote the set of plus (minus resp.) special vertices in $H_x^*$.

In the algorithm for coloring chordal+$k_1$e−$k_2$e graphs, for each minus modulator, we have to consider two cases; the case when the endpoints have the same color and the case when they have different colors. When there exist minus modulators in $K_x$ whose endpoints have the same color, the number of colors used in $K_x$ is less than $|K_x|$. Let $|K_x|'$ denote the number of colors used in $K_x$. Let $H_x^*(|K_x|')$ be the graph obtained by adding a clique that consists of $|C| - |K_x|'$ dummy vertices $u_1, u_2, \ldots, u_{|C|-|K_x|'}$ to $H_x$ and connecting each dummy vertex with each vertex of $K_x$. Let $K_x^*(|K_x|') = K_x \cup \{u_1, u_2, \ldots, u_{|C|-|K_x|'}\}$. In the following, we simply write them as $H_x^*$ and $K_x^*$ unless they are confusing.

Let $y$ be a child of $x$ in the nice tree decomposition. A minus special vertex $v$ is called latter-in if $v$ is introduced in node $x$ when $v^- \in K_y$ for some minus modulator $(v, v^-)$. A minus special vertex $v$ is called former-out if $v$ is forgotten in node $x$ when $v^- \in K_y$ for some minus modulator $(v, v^-)$. When a latter-in minus special vertex is introduced, one or more minus modulators appear in $H_x$. On the other hand, when a former-out minus special vertex is forgotten, one or more minus modulators disappear from $H_y$.

### 4.1 Set Systems

In our algorithm, a coloring of a graph is represented by a set system. Here, we use the same notations as in [7], however, some modifications are made considering that there are two kinds of special vertices.

When some minus modulators form a clique, more than two vertices in $K_x$ can have the same color. In such a coloring, the vertices with the same color are represented by the pairs constructed according to the order the vertices

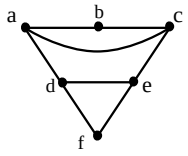



**Fig. 2.** A chordal+1e-2e graph.

are forgotten. When vertices $v_1, \cdots, v_s$ have the same color and the vertices are forgotten in this order, the set representing the coloring includes the pairs $(v_s, v_{s-1}), \cdots, (v_2, v_1)$. If some of the vertices in $v_1, \cdots, v_s$ are in $K_r$ for the root $r$ of the tree decomposition, the order of them can be decided arbitrarily. Among the vertices with the same color as $v$, let $l(v)$ denote the minus special vertex forgotten at last, and let $f(v)$ be the minus special vertex forgotten first.

**Definition 1.** *To each $C$-coloring $\psi$ of $H_x^*$, we associate a set $S_x(\psi) \subseteq (K_x^* \times W^+) \cup ((W^- \cap K_x) \times (W^- \cap K_x))$ that is obtained by the following rules.*
*i) For $v_1, \cdots, v_s \in K_x$ that have the same color in $\psi$, pairs to represent them are in $S_x(\psi)$.*
*ii) For $w \in W^+$ which does not appear in the right of any pair made by rule i), $(f(v), w) \in S_x(\psi)$ iff $\psi(v) = \psi(w)$ for $v \in K_x^*$.*
*The set system $\mathcal{S}_x$ contains a set $S$ iff there is a coloring $\psi$ of $H_x^*$ such that $S = S_x(\psi)$.*

Fig.2 is an example of a chordal+1e−2e graph which is obtained by adding and removing edges to chordal graph $G$ of Fig.1. Here, $(a, c)$ is the plus modulator and $(b, d), (b, e)$ are the minus modulators. In this example, when $|C| = 3$, $\mathcal{S}_t$ consists of four sets of pairs $\{(d, b), (e, a)\}, \{(e, a)\}, \{(d, b), (u_1, a)\}$ and $\{(e, b), (u_1, a)\}$. In each of them, two colors are used in $K_t$. Hence, each set corresponds to a coloring of $H_t^*(2)$, which consists of vertices $a, b, d, e$ and $u_1$.

**Definition 2.** *A set $S \subseteq (K_x^* \times W^+) \cup ((W^- \cap K_x) \times (W^- \cap K_x))$ is called regular if there exists at most one pair $(v, w)$ for each $w \in W^+$.*

**Definition 3.** *A blocker of $S$, denoted as $B(S)$, is the set of pairs $(v, w) \in K_x^* \times W^+$ satisfying that $(v, w') \in S$ for some plus modulator $(w, w')$. We say that sets $S_1$ and $S_2$ form a non-blocking pair if $B(S_1) \cap S_2 = S_1 \cap B(S_2) = \emptyset$.*

### 4.2 Algorithm

In this section, we describe the operations executed in each node $x$ of the tree decomposition. The operations are different according to the property of the node.

**Leaf node** If a vertex $v$ of $H_x$ is a plus special vertex, $\mathcal{S}_x$ contains only the set $\{(v, v)\}$. Otherwise, $\mathcal{S}_x$ contains only the empty set.

**Introduce node** Let $y$ be the child of $x$, and let vertex $v$ be introduced in $x$.
*Case 1. $v$ is neither a latter-in minus special vertex nor a plus special vertex.*
If $S \in \mathcal{S}_y$ includes a minus modulator in $K_y^*$, remove $S$ from $\mathcal{S}_y$. After that, obtain $\mathcal{S}_x$ by replacing each $u_{|C|-|K_y|'}$ by $v$ in each set in $\mathcal{S}_y$.
*Case 2. $v$ is a plus special vertex and not a latter-in minus special vertex.*
When $v' \in W_x^+$ for some plus modulator $(v, v')$, remove the sets including $(u_{|C|-|K_y|'}, v')$ from $\mathcal{S}_y$. After that, execute the same operations as in Case 1. Finally, if $v' \notin W_x^+$, add $(v, v)$ to each set.
*Case 3. $v$ is a latter-in minus special vertex and not a plus special vertex.*
For each set in $\mathcal{S}_y$, execute both 1) and 2) in the following. 1) corresponds to the case when $v$ has the same color as some of its pairs in $K_y$, and 2) corresponds to the case when $v$ has the color different from any of its pairs in $K_y$.
1) Let $(v, v_1), \cdots, (v, v_t)$ be all the minus modulators from $v$. Divide $v_1, \ldots, v_t$ into the sets of vertices with the same color. For each of them, if there does not exist a vertex in $K_x \backslash \{v_1, \ldots, v_t\}$ with the same color, make a set in $K_x$ by adding $v$ to the vertices with the same color. In this case, pairs are made so as to follow the rule described in Section 4.1. If $v$ is forgotten first among the vertices with the same color, replace the pair $(v_s, w)$ $(w \notin K_x)$ by $(v, w)$ when $v$ has the same color as $v_s$.
2) Make a set in $K_x$ by replacing $u_{|C|-|K_x|'}$ by $v$.
*Case 4. $v$ is both a positive special vertex and a latter-in minus special vertex.*
This case is almost similar to Case 3. However, there are two differences. In 1), when $v' \in W_x^+$ for some plus modulator $(v, v')$, $v$ cannot have the same color as $v_i$ for the set including $(v_i, v')$. If $v = l(v)$, add $(v, v)$ to the set. In 2), add $(v, v)$ to the set.
**Forget node** Let $y$ be the child of $x$, and let vertex $v$ be forgotten in $x$.
*Case 1. $v$ is not a former-out minus special vertex.*
Replace each $v$ in the left of a pair by $u_{|C|-|K_x|'}$ in each set. Then, add $S[u_i]$ $(1 \leq i \leq |C| - |K_x|')$ to $\mathcal{S}_x$ for each $S \in \mathcal{S}_x$.
*Case 2. $v$ is a former-out minus special vertex.*
For a set in $S$ that does not include a minus modulator $(v^-, v)$, do the same operations as Case 1. For a set in $S$ that includes a minus modulator $(v^-, v)$, execute the following operations. If $v$ is not a plus special vertex, remove the pair $(v^-, v)$ from $S$. If $v$ is a plus special vertex, replace $(v, v')$ by $(v^-, v')$.
**Join node** Let $y$ and $z$ be the children of $x$. $S_x = \{S_y \cup S_z \mid S_y \in \mathcal{S}_y$ and $S_z \in \mathcal{S}_z$ form a non-blocking pair, $S_y$ and $S_z$ include the same minus modulators in $K_y$ and $K_z$.$\}$

In the following, we prove the correctness of the algorithm.

**Lemma 1.** *Any set created by the above algorithm is regular.*

*Proof.* In the above algorithm, a new pair is added to a set only on the operations of an introduce node and a join node.
**introduce node** We consider the case when a plus special vertex $v$ is also a latter-in minus special vertex. It is obvious in other cases. Pair $(v, v)$ is added if either $v$ is not colored with any of its pairs or $v$ is forgotten at last among the

vertices with the same color in $K_x^*$. No other pair of the form $(*, v)$ is added in both cases.

**Join node** Assume that $S_y \in \mathcal{S}_y$ and $S_z \in \mathcal{S}_z$ are joined. Consider $v \in K_x$. $S_y$ and $S_z$ have the same coloring on the vertices in $K_x$. Therefore, if both sets have the pair of the form $(*, v)$, they must be the same. For $v \notin K_x$, either $S_y$ or $S_z$ does not include a pair of the form $(*, v)$. Therefore, the lemma holds for $S_y \cup \mathcal{S}_z$. □

**Lemma 2.** *For each set $S \in \mathcal{S}_x$, there exists a coloring of $H_x^*$ that does not contradict with the pairs in $S$.*

*Proof.* We inductively prove the lemma on the decomposition tree. A leaf node is the base case, and for non-leaf nodes, induction step should be shown for three kinds of nodes.

**Leaf node** For a leaf node $x$, the set created at $x$ is obviously a proper coloring of $H_x^*$.

**Introduce node** Assume that a set $Y \in \mathcal{S}_y$ corresponds to a coloring of $H_y^*$.

*Case 1. $v$ is neither a latter-in minus special vertex nor a plus special vertex.*
There exists a coloring corresponding to the set produced from $Y$ because $H_x^*$ is the same as $H_y^*$ except that $u_{|C|-|K_x|'}$is replaced by $v$.

*Case 2. $v$ is a plus special vertex and not a latter-in minus special vertex.*
This case is similar to Case 1.

*Case 3. $v$ is a latter-in minus special vertex and not a plus special vertex.*
1) When $v$ has the same color as some of its pairs in $K_y$, it is checked that the vertices form a clique of minus modulators. Thus the vertices have the same color. When $(f(v), w) \in Y$, $v$ and $w$ have the same color in $H_x^*$. Thus, the set produced from $Y$ corresponds to a proper coloring of $H_x^*$.
2) When $v$ has a color different from any of its pairs in $K_y$, it is obvious because $H_x^*$ is equivalent to $H_y^*$ except that $u_{|C|-|K_x|'}$ is replaced by $v$.

*Case 4. $v$ is both a positive special vertex and a latter-in minus special vertex.*
This case is similar to Case 3.

**Forget node** Assume that a set $Y \in \mathcal{S}_y$ corresponds to a coloring of $H_y^*$.

*Case 1. $v$ is not a former-out minus special vertex.*
As $H_x^*$ is equivalent to $H_y^*$ except that $v$ is replaced by $u_{|C|-|K_x|'}$, the set obtained by the replacement has a corresponding proper coloring. In addition, $S[u_i]$ also has a corresponding coloring in which only the colors of $u_i$ and $u_{|C|-|K_x|'}$ are exchanged from the corresponding coloring of $S$.

*Case 2. $v$ is a former-out minus special vertex.*
When $(v, v^-) \in Y$ for a minus modulator $(v, v^-)$, the corresponding coloring does not change by replacing $(v, *)$ by $(l(v), *)$ because $v$ and $v^-$ have the same color. When $(v, v^-) \notin Y$, it is similar to Case 1.

**Join node** Let $S_y$ and $S_z$ include the same minus modulators. In addition to the regularity of the sets, from the algorithm, if there exists a pair $(v, v')$ for a plus special vertex $v' \in H_x^* \backslash K_x^*$ in $S_y$ and $S_z$, then $v = f(v)$ holds. Therefore, if $S_y$ and $S_z$ form a non-blocking pair, there is no conflict on the colorings. Thus the set obtained by merging them has a corresponding coloring. □

The next lemma is almost obvious from the algorithm.

**Lemma 3.** *For each coloring of $H_x^*$, there exists a set $S \in \mathcal{S}_x$ that does not contradict with $H_x^*$.*

From Lemma 2 and 3, there exists a coloring of $G$ iff the set system is nonempty at the root node of the decomposition tree.

### 4.3 Representative System

**Definition 4.** *[8] A set system $\mathcal{S}' \subseteq \mathcal{S}$ is q-representative for $\mathcal{S}$ if the following holds: for every set $B$ of size at most $q$, there is a set $A \in \mathcal{S}$ with $A \cap B = \phi$ iff there is a set $A' \in \mathcal{S}'$ with $A' \cap B = \phi$.*

**Lemma 4.** *[8] Given a set system $\mathcal{S}$ containing $n$ sets of size at most $p$, a q-representative subsystem $\mathcal{S}' \in \mathcal{S}$ of size at most $\sum_{i=0}^{n} p^i$ can be found in $O(pq \cdot \sum_{i=0}^{p} p^i \sum n)$ time.*

The above definition of a representative set can be applied to our set system representing vertex colorings in the following manner.

**Definition 5.** *[7] A subsystem $\mathcal{S}_x^* \subseteq \mathcal{S}_x$ is representative for $\mathcal{S}_x$ if the following holds: for each regular set $U \subseteq K_x \times W$ that does not contain vertices from $W_x \backslash K_x^*$, if $\mathcal{S}_x$ contains a set $S$ disjoint from $B(U)$, then $\mathcal{S}_x^*$ also contains a set $S'$ disjoint from $B(U)$.*

In the algorithm described in Section4.2, we can replace a set system $\mathcal{S}_x$ by its representative system $\mathcal{S}_x^*$ for any node $x$. The following lemmas show that the algorithm works correctly even when we use the representative systems.

**Lemma 5.** *If $\mathcal{S}_y^*$ is representative for $\mathcal{S}_y$, $\mathcal{S}_x^*$ generated from $\mathcal{S}_y^*$ by introducing a vertex is representative for $\mathcal{S}_x$.*

*Sketch of proof.* We show that, if a set $S \in \mathcal{S}_x$ which corresponds to a coloring $\psi$ of $H_x^*$ is disjoint from $B(U)$, there exists a set $S' \in \mathcal{S}_x^*$ which is disjoint from $B(U)$.

In the case when $v$ is not a latter-in minus special vertex, The proof is similar to the proof for chordal+$k$e graphs[7]. The only difference is that there may be more than one vertices with the same color. Even when $v$ is a latter-in minus special vertex, if $v$ has a color different from any vertex of $K_x$ in $\psi$, it is equivalent to the case when $v$ is not a latter-in minus special vertex.

In the following, we consider the case when $v$ is a latter-in minus special vertex and $v$ has the same color as vertices $v_1, \ldots, v_t$ in $K_x$. Let a coloring $\psi'$ of $H_y^*$ be obtained by only removing $v$ from $H_x^*$. Note that the number of dummy vertices in $H_x^*$ and $H_y^*$ are the same. Set $R$ corresponding to $\psi'$ is obtained from $S$ by removing $v$ or replacing $v$ by a vertex with the same color. Thus, as $S$ is disjoint from $B(U)$, $R$ is also disjoint from $B(U)$. As $\mathcal{S}_y^*$ is representative, there exists $R' \in \mathcal{S}_y^*$ which is disjoint from $B(U)$. It is easy to check that the sets created from $R'$ by our algorithm is disjoint from $B(U)$. □

**Lemma 6.** *If $\mathcal{S}_y^*$ is representative for $\mathcal{S}_y$, $\mathcal{S}_x^*$ generated from $\mathcal{S}_y^*$ by forgetting $v$ is representative for $\mathcal{S}_x$.*

*Sketch of proof.* The outline of the proof is the same as that of Lemma5. As the other cases are similar to the case of chordal+$k$e graphs, we consider the case when $v$ is a former-out minus special vertex and $S$ includes a minus modulator $(v, v^-)$.

By adding $v$ to the set of vertices with the same color as $v^-$ in $H_x^*$, a coloring $\psi'$ of $H_y^*$ is obtained. Set $R$ corresponding to $\psi'$ is equivalent to the set obtained by adding $v$ to the set of vertices with the same color as $v^-$, except the replacement of $v$ by vertices with the same color. Therefore, $R$ is disjoint to $B(U)$.

As $\mathcal{S}_y^*$ is representative, there exists $R' \in \mathcal{S}_y^*$ which is disjoint to $B(U)$. It can be easily seen that the set obtained from $R'$ by our algorithm is disjoint to $B(U)$. □

**Lemma 7.** *If $\mathcal{S}_y^*$ and $\mathcal{S}_z^*$ are representative for $\mathcal{S}_y$ and $\mathcal{S}_z$ respectively, $\mathcal{S}_x^*$ obtained by joining them is representative for $\mathcal{S}_x$.*

Lemma7 can be proved similarly to the case of chordal+$k$e graphs.

In a set of pairs, the number of pairs which include a plus special vertex is bouded by the number of plus special vertices. The number of pairs which does not include a plus special vertex is bounded by the number of minus modulators. Thus, the size of each set is bounded by a function of $k_1$ and $k_2$. Whenever we construct $\mathcal{S}_x^*$, we can apply Lemma4 for sets which has the same minus modulators in $K_x$. The number of different sets of minus modulators in $K_x$ is bounded by $2^{k_2}$. Therefore, throughout the algorithm, the size of $\mathcal{S}_x^*$ is always bounded by a function of $k_1$ and $k_2$. The time to compute $\mathcal{S}_x^*$ is also bounded by a function of $k_1$ and $k_2$.

From the above discussions, the following theorem holds.

**Theorem 2.** *Vertex coloring problem of chordal+$k_1$e−$k_2$e graphs is fixed parameter tractable for parameters $k_1$ and $k_2$.*

## Acknowledgements

This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan.

## References

1. J. R. S. Blair and B. W. Payton, An Introduction to Chordal Graphs and Clique Trees, In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, Graph Theory and Sparse Matrix Computations, pp.1-30. Springer Verlag, IMA Volumes in Mathematics and its Applications, Vol.56, 1993.
2. L. Cai, Parameterized Complexity of Vertex Colouring, Discrete Applied Math. 127, pp.415-429, 2003.

3. R. G. Downey and M. R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer-Verlag, New York, 1999.
4. M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs 2nd Ed., Annals of Discrete Mathematics 57, Elsevier, Amsterdam, 2004.
5. J. Guo, F. Hüffner and R. Niedermeier, A Structural View on Parameterizing Problems: Distance from Triviality, IWPEC 2004, LNCS 3162, 162-173, 2004.
6. T. Kloks, Treewidth, LNCS 842, Springer-Verlag, Berlin, 1994.
7. D. Marx, Parameterized Coloring Problems on Chordal Graphs, Theoret. Comput. Sci., 351, 3, pp.407-424, 2006.
8. B. Monien, How to Find Long Paths Efficiently, Analysis and Design of Algorithms for Combinatorial Problems (Udine, 1982), vol.109, North-Holland MAth. Stud., pp.239-254, North-Holland, Amsterdam, 1985.
9. Y. Takenaga and K. Higashide, Vertex Coloring of Comparability+ke and -ke Graphs, 32nd International Workshop on Graph-Theoretic Concepts in Computer Science, LNCS 4271, pp.102-112, 2006.

# Fault-free Hamiltonian Cycles in Alternating Group Graphs with Conditional Edge Faults

Ping-Ying Tsai[1,*], Jung-Sheng Fu[2], and Gen-Heuy Chen[1]

[1] Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, ROC
[2] Department of Electronics Engineering,
National United University, Miaoli, Taiwan, ROC
[*]bytsai0808@gmail.com

**Abstract.** The alternating group graph, which belongs to the class of Cayley graphs, is one of the most versatile interconnection networks for parallel and distributed computing. In this paper, adopting the conditional fault model in which each vertex is assumed to be incident with two or more fault-free edges, we show that an $n$-dimensional alternating group graph can tolerate up to $4n - 13$ edge faults, where $n \geq 4$, while retaining a fault-free Hamiltonian cycle exists. The result is optimal with respect to the number of edge faults tolerated. Previously, for the same problem, at most $2n - 6$ edge faults can be tolerated if the random fault model is adopted.

**Keywords:** alternating group graph, embedding, Hamiltonian cycle, fault tolerance, conditional fault model, Cayley graph.

## 1 Introduction

Network topology is a crucial factor for the interconnection networks since it determines the performance of the networks. Many interconnection network topologies have been proposed in the literature for the purpose of connecting hundreds or thousands of processing elements. It can be represented by a graph where nodes represent processors and edges represent links between processors. The alternating group graphs [11], like the well-known star graphs [1] and hypercubes [14], belongs to the class of Cayley graphs [2, 12]. Furthermore, it has been shown in [7] that a class of generalized star graphs called the arrangement graphs also contains alternating group graphs as members. Indeed, a proof given in [7] showed that the $n$-dimensional alternating group graph $AG_n$ is isomorphic to the $(n, n-2)$-arrangement graph $A_{n,n-2}$.

Arrangement graphs have been shown to be vertex and edge symmetric, strongly hierarchical, maximally fault tolerant, and strongly resilient [8], and thus of alternating group graphs. Besides, alternating group graphs have sublogarithmic degree and diameter [11]. They are all desirable when we are building an interconnection topology for a parallel and distributed system. An efficient

communication algorithm for shortest-path routing is available for alternating group graphs [11].

The study of graph embeddings arises naturally in a number of computational problems: finding storage schemes for logical data structures, layout of circuits in VLSI, portability of algorithms across various parallel architectures, just to mention a few [12]. Among of them, the ring is one of the most fundamental networks for parallel and distributed computation, and it is suitable to develop simple and efficient algorithms. Numerous algorithms that were designed on rings for solving various algebraic problems and graph problems can be found in [3, 13]. A ring can be also used as a control/data flow structure for distributed computation in a network. These applications motivate the embedding of cycles in networks. It was shown that the arbitrary cycles can be embedded in an alternating group graph [11]. Besides, the alternating group graph also can embed some other fundamental networks, such as grids [11], trees [11], and arbitrary paths [6].

Since node faults and/or link faults may occur to networks, it is significant to consider faulty networks. Many fundamental problems such as diameter, routing, broadcasting, gossiping, embedding, *etc.*, have been studied on various faulty networks. Among them, two fault models were adopted; one is the random fault model, and the other is the conditional fault model. The *random fault model* assumed that the faults might occur everywhere without any restriction, whereas the *conditional fault model* assumed that the distribution of faults must satisfy some properties, e.g., two or more fault-free links incident to each node. Apparently, it is more difficult to solve problems under the conditional fault model than the random fault model. Previously related work about embedding in faulty networks under the conditional fault model can be found in [4, 5, 9, 15, 16].

In this paper, under the conditional fault model and with the assumption of at least two fault-free links incident to each node, we show that an $n$-dimensional alternating group graph can tolerate up to $4n-13$ link faults, where $n \geq 4$, while retaining a fault-free Hamiltonian cycle exists. The result is optimal with respect to the number of link faults tolerated. For the same problem, at most $2n-6$ link faults can be tolerated if the random fault model is adopted [10]. With our results, all parallel algorithms developed on rings can be executed as well on an $n$-dimensional alternating group graph with up to $4n-13$ link faults.

The rest of the paper is organized as follows. In Section 2, the structure of the alternating group graph is reviewed. Necessary definitions, notations, and some properties of the alternating group graph are also introduced in order to prove the main result. Then the main result and its proof are shown in Section 3. Finally, this paper concludes with some remarks in Section 4.

## 2 Preliminaries

It is convenient to represent a network with a graph $G$, where each vertex (edge) of $G$ uniquely represents a node (link) of the network. For the graph definition and notation, we follow [17]. We use $V(G)$ and $E(G)$ to denote the

vertex set and edge set of $G$, respectively. Given a vertex $u$ in $G$, we define $N(u) = \{v|(u,v) \in E(G)\}$ to be the *neighborhood* of $u$, which is the set of vertices that are adjacent to $u$ in $G$. The *degree* of $u$, denoted by $\deg(u)$, is the size of $N(u)$, i.e., $\deg(u) = |N(u)|$. We use $\delta(G)$ to denote $\min\{\deg(u)|u \in V(G)\}$. Let $V'$ be a vertex subset of $G$. We define $N(V') = \bigcup_{u \in V'} N(u) - V'$ to be the *neighborhood* of $V'$. A path $P_{x_0x_t} = \langle x_0, x_1, \ldots, x_t \rangle$, is a sequence of vertices such that every two consecutive vertices are adjacent. In addition, $P_{x_0x_t}$ is a cycle if $x_0 = x_t$. A path $\langle x_0, x_1, \ldots, x_t \rangle$ may contain other subpath, denoted as $\langle x_0, x_1, \ldots, x_i, P_{x_ix_j}, x_j, \ldots, x_t \rangle$, where $P_{x_ix_j} = \langle x_i, x_{i+1}, \ldots, x_{j-1}, x_j \rangle$.

A path (or cycle) in $G$ is called a *Hamiltonian path* (or *Hamiltonian cycle*) if it contains every vertex of $G$ exactly once. A graph is called *Hamiltonian* if it has a Hamiltonian cycle. A graph is called *Hamiltonian connected* if every two vertices of $G$ are connected by a Hamiltonian path. All Hamiltonian connected graphs except $K_1$ and $K_2$ are Hamiltonian.

Let $p = a_1a_2 \cdots a_n$ is a permutation on $\{1, 2, \ldots, n\}$. A pair of symbols $a_i$ and $a_j$ in $p$ are said to be an *inversion* if $a_i < a_j$ whenever $i > j$. A permutation is an *even permutation* if it has an even number of inversions. The *alternating group* $A_n$ is the set consisting of all even permutations on $\{1, 2, \ldots, n\}$, where $|A_n| = \frac{n!}{2}$. The following is a formal definition of alternating group graphs, in terms of graph theory.

**Definition 1.** *An n-dimensional alternating group graph, denoted by $AG_n$, has the vertex set $V(AG_n) = \{a_1a_2 \cdots a_n | a_1a_2 \cdots a_n$ is an even permutation of $1, 2, \ldots, n\}$ and the edge set $E(AG_n) = \{(a_1a_2 \cdots a_n, a_2a_i \cdots a_{i-1}a_1a_{i+1} \cdots a_n)$ or $(a_1a_2 \cdots a_n, a_ia_1 \cdots a_{i-1}a_2a_{i+1} \cdots a_n) | a_1a_2 \cdots a_n \in V(AG_n)$ and $3 \leq i \leq n\}$.*

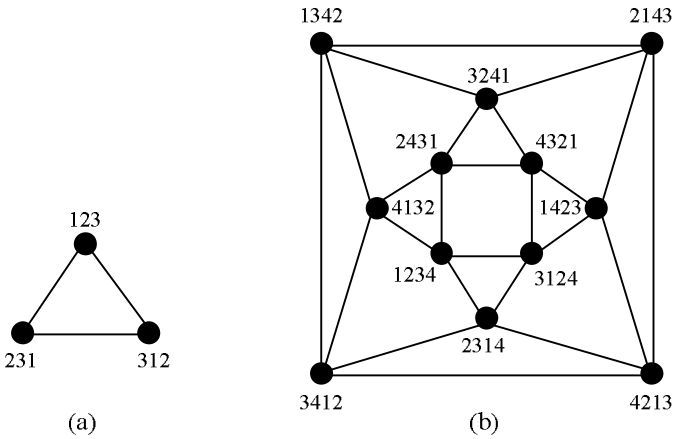



**Fig. 1.** Examples of alternating group graphs. (a) $AG_3$. (b) $AG_4$.

From definition, it is easy to see that the vertex set of $AG_n$ is the alternating group $A_n$. It has $\frac{n!}{2}$ vertices, each of degree $2(n-2)$, and has $\frac{(n-2)n!}{2}$ edges. The alternating group graphs $AG_3$ and $AG_4$ are shown in Fig. 1. Let $u = a_1a_2 \cdots a_n$

be any vertex of the alternating group graph $AG_n$. The edges $(a_1a_2\cdots a_n,\ a_2a_i\cdots a_{i-1}a_1a_{i+1}\cdots a_n)$ and $(a_1a_2\cdots a_n,\ a_ia_1\cdots a_{i-1}a_2a_{i+1}\cdots a_n)$, denoted by $e^{(i)}(u)$, are referred to as *i-dimensional edges* of $u$, where $3 \leq i \leq n$. We use $E^{(i)}(AG_n)$ to denote the set of all $i$-dimensional edges in $AG_n$.

Alternating group graphs are vertex symmetric, edge symmetric, and strongly hierarchical [11]. For $1 \leq k \leq n$, let $AG_n(k)$ denote the subgraph of $AG_n$ induced by those vertices $u$ with $a_n = k$. Clearly, each $AG_n(k)$ is isomorphic to $AG_{n-1}$ for $1 \leq k \leq n$. Due to the strongly hierarchical structure, the alternating group graph can also be defined recursively: $AG_n$ is constructed from $n$ disjoint copies of $(n-1)$-dimensional alternating group graph $AG_{n-1}$'s. We use $\tilde{E}^{(n)}_{p,q}(AG_n)$ to represent the set of those $n$-dimensional edges in $AG_n$ that connect $AG_n(p)$ and $AG_n(q)$, where $1 \leq p \neq q \leq n$.

Throughout this paper, the paired terms network and graph, node and vertex, and link and edge are used interchangeably. Since $AG_n$ is isomorphic to the $(n, n-2)$-arrangement graph $A_{n,n-2}$, the following lemma of $AG_n$ is deduced from a result of arrangement graphs.

**Lemma 1.** ([10]) *For any $F' \subseteq V(AG_n) \cup E(AG_n)$, $AG_n - F'$ is Hamiltonian if $|F'| \leq 2n-6$, and Hamiltonian connected if $|F'| \leq 2n-7$, where $n \geq 4$.*

We also present some properties of $AG_n$ in the following. They are necessary in order to show our main result in the next section. Besides, we use $F$ ($\subseteq E(AG_n)$) to denote the set of edge faults in $AG_n$.

**Lemma 2.** $|\tilde{E}^{(n)}_{i,j}(AG_n)| = (n-2)!$, *where $n \geq 4$.*

*Proof.* Consider $p = p_1p_2\cdots p_n \in V(AG_n(i))$, where $p_n = i$. Suppose that $p$ connect to $AG_n(j)$, hence we have $p_1 = j$ or $p_2 = j$. If $p_1 = j$, then there are $\frac{(n-2)!}{2}$ choices for $p_2, p_3, \ldots, p_{n-1}$. The discussion is similar if $p_2 = j$. So $|\tilde{E}^{(n)}_{i,j}(AG_n)| = (n-2)!$. □

**Lemma 3.** *Suppose that $I = \{k_1, k_2, \ldots, k_m\} \subseteq \{1, 2, \ldots, n\}$, where $n \geq 5$ and $m \geq 2$. Let $AG_n(I)$ denote the subgraph of $AG_n$ induced by $\bigcup_{k\in I} V(AG_n(k))$. If $AG_n(k) - F$ is Hamiltonian connected for every $k \in I$ and $|\tilde{E}^{(n)}_{k_j,k_{j+1}}(AG_n) - F| \geq 3$ for all $1 \leq j < m$, then there is a Hamiltonian path $P_{st}$ in $AG_n(I) - F$, where $s \in V(AG_n(k_1))$ and $t \in V(AG_n(k_m))$.*

*Proof.* Note that if $v = c_1c_2\cdots c_n \in V(AG_n)$, then $v \in N(V(AG_n(c_1))) \cap N(V(AG_n(c_2)))$, thus, the two edges of $e^{(n)}(v)$ incident to different $AG_n(c)$s. Let $u_1 = s$. Since $|\tilde{E}^{(n)}_{k_j,k_{j+1}}(AG_n) - F| > 2$ for all $1 \leq j < m$, we can find an edge $(v_1, u_2) \in E^{(n)}(AG_n) - F$ such that $v_1 \neq u_1$ and $u_2 \in V(AG_n(k_2))$. Similarly, we can find edges $(v_2, u_3), (v_3, u_4), \ldots, (v_{m-2}, u_{m-1}) \in E^{(n)}(AG_n) - F$, where $u_i$ and $v_i$ are two distinct vertices in $AG_n(k_i)$ for all $i \in \{2, 3, \ldots, m-1\}$. Since $|\tilde{E}^{(n)}_{k_{m-1},k_m}(AG_n) - F| \geq 3$, we can find an edge $(v_{m-1}, u_m) \in E^{(n)}(AG_n) - F$ such that $v_{m-1} \neq u_{m-1}$, $u_m \neq t$, and $u_m \in V(AG_n(k_m))$. Let $v_m = t$. In addition, since $AG_n(k_i) - F$ is Hamiltonian connected for all $k_i \in I$, there is a Hamiltonian

path $P_{u_iv_i}$ in $AG_n(k_i) - F$ for all $i \in \{1, 2, \ldots, m\}$. An Hamiltonian path $P_{st}$ in $AG_n(I) - F$ is constructed as follows (see Fig. 2):

$\langle s, P_{sv_1}, v_1, u_2, P_{u_2v_2}, v_2, \ldots, u_{m-1}, P_{u_{m-1}v_{m-1}}, v_{m-1}, u_m, P_{u_mt}, t \rangle$. □

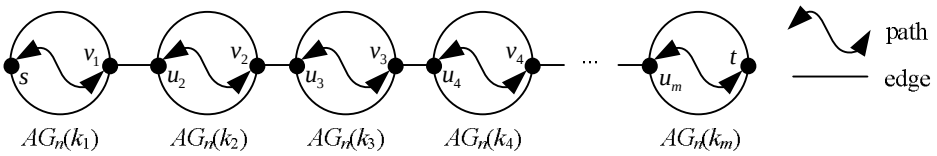



**Fig. 2.** A Hamiltonian path $P_{st}$ in $AG_n(I) - F$.

## 3 Main Result

In this section, we would show that with the assumption of two or more fault-free edges incident to each vertex, an $n$-dimensional alternating group graph can tolerate up to $4n - 13$ edge faults, while retaining a fault-free Hamiltonian cycle exists, where $n \geq 4$.

**Theorem 1.** *$AG_n - F$ is Hamiltonian if $|F| \leq 4n - 13$ and $\delta(AG_n - F) \geq 2$, where $n \geq 4$.*

*Proof.* We can proceed by induction on $n$. When $n = 4$, we can use a computer program to check that the result is true [18]. Assume that the result holds for $AG_n$ for some $n \geq 4$. Consider $AG_{n+1}$ with $|F| \leq 4n-9$ and $\delta(AG_{n+1}-F) \geq 2$. For brevity, assume that $|F| = 4n - 9$. Without loss of generality, assume that $|E^{(n+1)}(AG_{n+1}) \cap F| \geq |E^{(n)}(AG_{n+1}) \cap F| \geq \ldots \geq |E^{(3)}(AG_{n+1}) \cap F|$. If $n \geq 7$, we have $|E^{(n+1)}(AG_{n+1}) \cap F| \geq \left\lceil \frac{(4n-9)}{(n-1)} \right\rceil \geq 4$, $|F - E^{(n+1)}(AG_{n+1})| \leq 4n - 13$, and $|E(AG_{n+1}(r)) \cap F| \leq 4n - 13$ for all $1 \leq r \leq n + 1$. In addition, when $4 \leq n \leq 6$, we have $|E^{(n+1)}(AG_{n+1}) \cap F| \geq 3$, $|F - E^{(n+1)}(AG_{n+1})| \leq 4n - 12$, and $|E(AG_{n+1}(r)) \cap F| \leq 4n - 12$ for all $1 \leq r \leq n + 1$. Without loss of generality, assume that $|E(AG_{n+1}(n + 1)) \cap F| \geq |E(AG_{n+1}(n)) \cap F| \geq \ldots \geq |E(AG_{n+1}(1)) \cap F|$. Note that when $n \geq 5$, by Lemma 2, $|\tilde{E}^{(n+1)}_{j,k}(AG_{n+1})| = ((n+1)-2)! = (n-1)! > 4n-6 = |F|+3$, hence we have $|\tilde{E}^{(n+1)}_{j,k}(AG_{n+1})-F| \geq 3$ for all $j, k \in \{1, 2, \ldots, n + 1\}$ and $j \neq k$. If $n = 4$, we have $|\tilde{E}^{(5)}_{j,k}(AG_5)| = (5 - 2)! = 6 < 7 = 4n - 9$ for all $j, k \in \{1, 2, \ldots, 5\}$ and $j \neq k$. Hence, it is possible that $|\tilde{E}^{(5)}_{j',k'}(AG_5) - F| < 3$ for some $j', k' \in \{1, 2, \ldots, 5\}$ and $j' \neq k'$. We use $i_1, i_2, \ldots, i_{n+1}$ to denote the $n+1$ distinct integers from 1 to $n+1$ (i.e., $\{i_1, i_2, \ldots, i_{n+1}\} = \{1, 2, \ldots, n + 1\}$). Four cases are considered:

*Case 1.* $|E(AG_{n+1}(n + 1)) \cap F| \leq 2n - 7$. In this case, $\delta(AG_{n+1}(r) - F) \geq 2$ for all $1 \leq r \leq n + 1$. Let $i_1 = n + 1$ and $I = \{1, 2, \ldots, n\}$. We can find $u_1, v_1 \in V(AG_{n+1}(i_1))$ such that $(v_1, u_2), (u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1}) - F$, where $u_2 \in$

$V(AG_{n+1}(i_2))$ and $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. By Lemma 1 and Lemma 3, we can find a Hamiltonian path $P_{u_1v_1}$ in $AG_{n+1}(i_1) - F$ and a Hamiltonian path $P_{u_2v_{n+1}}$ in $AG_{n+1}(I) - F$ (when $n = 4$ and if $j'$, $k'$ exist, let $\{j', k'\} \neq \{i_r, i_{r+1}\}$ for $2 \leq r \leq n$). Hence $\langle u_1, P_{u_1v_1}, v_1, u_2, P_{u_2v_{n+1}}, v_{n+1}, u_1 \rangle$ form a Hamiltonian cycle in $AG_{n+1} - F$.

*Case 2.* $2n - 6 \leq |E(AG_{n+1}(n+1)) \cap F| \leq 4n - 13$ and $|E(AG_{n+1}(n)) \cap F| \leq 2n - 7$. Then we have $|E(AG_{n+1}(r)) \cap F| \leq 2n - 7$ for all $r \in \{1, 2, \ldots, n\}$. Let $i_1 = n + 1$. Three cases are further considered:

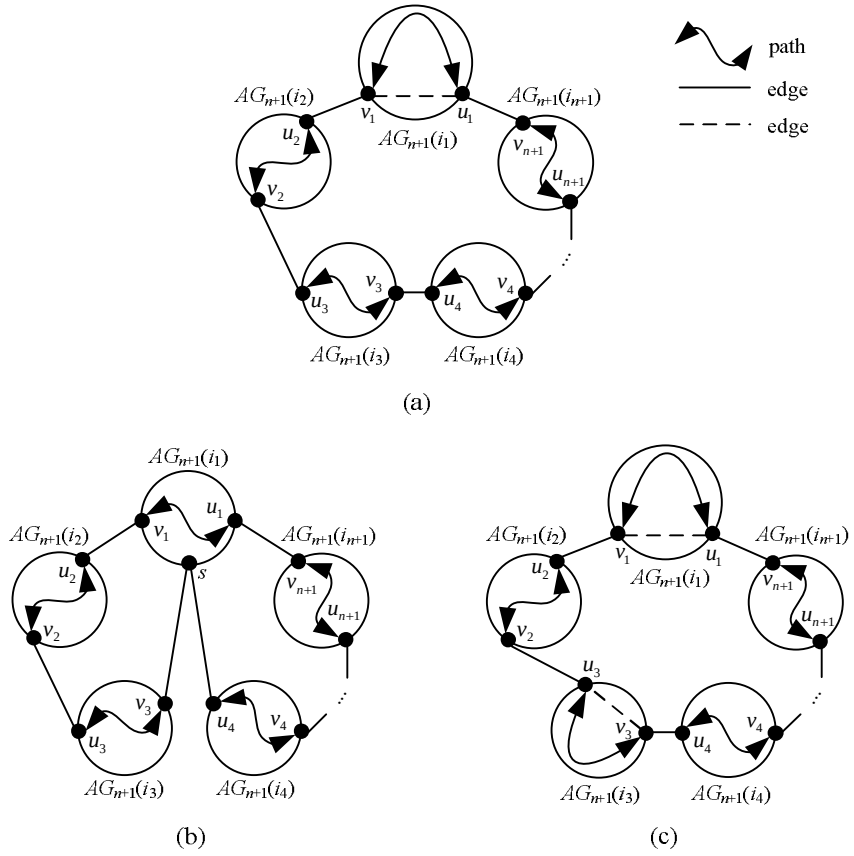



**Fig. 3.** A Hamiltonian cycle in $AG_{n+1} - F$. (a) $|E(AG_{n+1}(n+1)) \cap F| \geq 2n - 6$, $|E(AG_{n+1}(n)) \cap F| \leq 2n - 7$, and $\delta(AG_{n+1}(i_1) - F) \geq 1$. (b) $|E(AG_{n+1}(n+1)) \cap F| \geq 2n - 6$, $|E(AG_{n+1}(n)) \cap F| \leq 2n - 7$, and $\delta(AG_{n+1}(i_1) - F) = 0$. (c) $|E(AG_{n+1}(n+1)) \cap F| = |E(AG_{n+1}(n)) \cap F| = 2n - 6$.

*Case 2.1.* $\delta(AG_{n+1}(i_1) - F) \geq 2$. The induction hypothesis assures that there exists a Hamiltonian cycle $C$ in $AG_{n+1}(i_1) - F$. We can find $(u_1, v_1) \in C$ such that $(v_1, u_2), (u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1}) - F$, where $u_2 \in V(AG_{n+1}(i_2))$ and

$v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. Let $P_{u_1v_1} = C - \{(u_1, v_1)\}$ and $I = \{1, 2, \ldots, n\}$. By Lemma 3, we can find a Hamiltonian path $P_{u_2v_{n+1}}$ in $AG_{n+1}(I) - F$ (when $n = 4$ and if $j'$, $k'$ exist, let $\{j', k'\} \neq \{i_r, i_{r+1}\}$ for $2 \leq r \leq n$). Hence a desired Hamiltonian cycle in $AG_{n+1} - F$ can be obtained as shown in Fig. 3(a).

*Case 2.2.* $\delta(AG_{n+1}(i_1)-F) = 1$. There is exactly one vertex $v_1$ with degree one in $AG_{n+1}(i_1)-F$. Since $\delta(AG_{n+1}-F) \geq 2$, we have $(v_1, u_2) \in E^{(n+1)}(AG_{n+1})-F$, for some $u_2 \in V(AG_{n+1}(i_2))$. Select $(u_1, v_1) \in E(AG_{n+1}(i_1)) \cap F$ such that $(u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1})-F$, where $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. We can always find such $(u_1, v_1)$ since $|\{z|(v_1, z) \in F \text{ and } z \in V(AG_{n+1}(i_1))\}| = 2n - 5$, $|e^{(n+1)}(z)| = 2$ for all $z \in V(AG_{n+1}(i_1)), |E^{(n+1)}(AG_{n+1}) \cap F| \leq (4n - 9) - (2n - 5) = 2n - 4$, and $2(2n - 5) > 2n - 4$ when $n \geq 4$. Moreover, since $|E(AG_{n+1}(i_1)) \cap (F - \{(u_1, v_1)\})| \leq 4n - 14$, the induction hypothesis assures that there is a Hamiltonian cycle $C$ in $AG_{n+1}(i_1) \cap (F - \{(u_1, v_1)\})$. Note that $(u_1, v_1)$ must be contained in $C$. Then the construction of a Hamiltonian cycle in $AG_{n+1} - F$ is similar to Fig. 3(a) (when $n = 4$ and if $j'$, $k'$ exist, let $\{j', k'\} \neq \{i_r, i_{r+1}\}$ for $2 \leq r \leq n$).

*Case 2.3.* $\delta(AG_{n+1}(i_1) - F) = 0$. Note that $4n - 13 = 3 < 4 = 2(n - 2)$ when $n = 4$, hence this case will occur only when $n \geq 5$, thus, $j'$ and $k'$ do not exist. There is exactly one vertex $s$ with degree zero in $AG_{n+1}(i_1) - F$. Since $\delta(AG_{n+1} - F) \geq 2$, we have $e^{(n+1)}(s) \cap F = 0$. Let $(v_3, s), (s, u_4) \in e^{(n+1)}(s)$ where $v_3 \in V(AG_{n+1}(i_3))$ and $u_4 \in V(AG_{n+1}(i_4))$. Additionally, there exist $(v_1, u_2), (u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1}) - F$, where $u_1$, $v_1$ are two distinct vertices in $V(AG_{n+1}(i_1)) - \{s\}$, $u_2 \in V(AG_{n+1}(i_2))$, and $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. In addition, let $F' = \{s\} \cup (E(AG_{n+1}(i_1)) \cap F - \{(s, z)|z \in V(AG_{n+1}(i_1))\})$. Note that $|F'| \leq 2n - 8$. By Lemma 1, we can find a Hamiltonian path $P_{u_1v_1}$ in $AG_{n+1}(i_1) - F'$. Let $I_1 = \{i_2, i_3\}$ and $I_2 = \{1, 2, \ldots, n + 1\} - \{i_1, i_2, i_3\}$. By Lemma 3, we can find a Hamiltonian path $P_{u_2v_3}$ in $AG_{n+1}(I_1) - F$, and a Hamiltonian path $P_{u_4v_{n+1}}$ in $AG_{n+1}(I_2) - F$. Hence a desired Hamiltonian cycle in $AG_{n+1} - F$ can be obtained as shown in Fig. 3(b).

*Case 3.* $|E(AG_{n+1}(n + 1)) \cap F| = 4n - 12$. Note that this case will occur only when $4 \leq n \leq 6$. In addition, $j'$ and $k'$ do not exist and $|E(AG_{n+1}(r)) \cap F| = 0$ for all $r \in \{1, 2, \ldots, n\}$. Let $i_1 = n + 1$. Three cases are further considered:

*Case 3.1.* $\delta(AG_{n+1}(i_1) - F) \geq 2$. We can find $(u_1, v_1) \in E(AG_{n+1}(i_1)) \cap F$ such that $(v_1, u_2), (u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1}) - F$, where $u_2 \in V(AG_{n+1}(i_2))$ and $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. Since $|E(AG_{n+1}(i_1)) \cap (F - \{(u_1, v_1)\})| = 4n - 13$, the induction hypothesis assures that there is a Hamiltonian cycle $C$ in $AG_{n+1}(i_1) \cap (F - \{(u_1, v_1)\})$. Assume that $C$ contains $(u_1, v_1)$ (otherwise, the discussion is easier). Then the construction of a Hamiltonian cycle in $AG_{n+1} - F$ is similar to Fig. 3(a).

*Case 3.2.* $\delta(AG_{n+1}(i_1)-F) = 1$. There is exactly one vertex $v_1$ with degree one in $AG_{n+1}(i_1)-F$. Since $\delta(AG_{n+1}-F) \geq 2$, we have $(v_1, u_2) \in E^{(n+1)}(AG_{n+1})-F$, for some $u_2 \in V(AG_{n+1}(i_2))$. Select $(u_1, v_1) \in E(AG_{n+1}(i_1)) \cap F$ such that $(u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1})-F$, where $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. We can always

find such $(u_1, v_1)$ since $|\{z|(v_1, z) \in F \text{ and } z \in V(AG_{n+1}(i_1))\}| = 2n-5$, $|e^{(n+1)}(z)| = 2$ for all $z \in V(AG_{n+1}(i_1))$, $|E^{(n+1)}(AG_{n+1}) \cap F| = (4n-9) - (4n-12) = 3$, and $2(2n-5) > 3$ when $n \geq 4$. Moreover, since $|E(AG_{n+1}(i_1)) \cap (F - \{(u_1, v_1)\})| = 4n-13$, the induction hypothesis assures that there is a Hamiltonian cycle $C$ in $AG_{n+1}(i_1) \cap (F - \{(u_1, v_1)\})$. Note that $(u_1, v_1)$ must be contained in $C$. Then the construction of a Hamiltonian cycle in $AG_{n+1} - F$ is similar to Case 3.1.

*Case 3.3.* $\delta(AG_{n+1}(i_1) - F) = 0$. There is exactly one vertex $s$ with degree zero in $AG_{n+1}(i_1) - F$. Since $\delta(AG_{n+1} - F) \geq 2$, we have $e^{(n+1)}(s) \cap F = 0$. Let $(v_3, s), (s, u_4) \in e^{(n+1)}(s)$ where $v_3 \in V(AG_{n+1}(i_3))$ and $u_4 \in V(AG_{n+1}(i_4))$. Additionally, there exist $(v_1, u_2), (u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1}) - F$, where $u_1, v_1 \in V(AG_{n+1}(i_1)) - \{s\}$, $u_2 \in V(AG_{n+1}(i_2))$, and $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. In addition, let $F' = \{s\} \cup (E(AG_{n+1}(i_1)) \cap F - \{(s, z)|z \in V(AG_{n+1}(i_1))\})$. Note that $|F'| \leq 2n-7$. By Lemma 1, we can find a Hamiltonian path $P_{u_1v_1}$ in $AG_{n+1}(i_1) - F'$. Let $I_1 = \{i_2, i_3\}$ and $I_2 = \{1, 2, \ldots, n+1\} - \{i_1, i_2, i_3\}$. By Lemma 3, we can find a Hamiltonian path $P_{u_2v_3}$ in $AG_{n+1}(I_1) - F$, and a Hamiltonian path $P_{u_4v_{n+1}}$ in $AG_{n+1}(I_2) - F$. Hence a desired Hamiltonian cycle in $AG_{n+1} - F$ can be obtained as shown in Fig. 3(b).

*Case 4.* $|E(AG_{n+1}(n+1)) \cap F| = |E(AG_{n+1}(n)) \cap F| = 2n-6$. Note that this case will occur only when $4 \leq n \leq 6$. In addition, $j'$ and $k'$ do not exist in this case and $|E(AG_{n+1}(r)) \cap F| = 0$ for all $r \in \{1, 2, \ldots, n-1\}$. In this case, we also have $\delta(AG_{n+1}(r) - F) \geq 2$ for all $1 \leq r \leq n+1$. Let $i_1 = n+1$ and $i_3 = n$. The induction hypothesis assures that there are a Hamiltonian cycle $C_1$ in $AG_{n+1}(i_1) - F$ and a Hamiltonian cycle $C_3$ in $AG_{n+1}(i_3) - F$. We can find $(u_1, v_1) \in C_1$ and $(u_3, v_3) \in C_3$ such that $(v_1, u_2), (v_2, u_3), (v_3, u_4), (u_1, v_{n+1}) \in E^{(n+1)}(AG_{n+1}) - F$, where $u_2, v_2 \in V(AG_{n+1}(i_2))$, $u_4 \in V(AG_{n+1}(i_4))$, and $v_{n+1} \in V(AG_{n+1}(i_{n+1}))$. Let $I = \{1, 2, \ldots, n+1\} - \{i_1, i_2, i_3\}$, $P_{u_1v_1} = C_1 - \{(u_1, v_1)\}$, and $P_{u_3v_3} = C_3 - \{(u_3, v_3)\}$. By Lemma 1 and Lemma 3, we can find a Hamiltonian path $P_{u_2v_2}$ in $AG_{n+1}(i_2) - F$ and a Hamiltonian path $P_{u_4v_{n+1}}$ in $AG_{n+1}(I) - F$. Hence a desired Hamiltonian cycle in $AG_{n+1} - F$ can be obtained as shown in Fig. 3(c). □

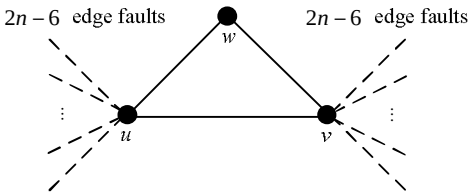



**Fig. 4.** A distribution of $4n-12$ edge faults over $AG_n$.

The result is optimal with respect to the number of edge faults tolerated. Since alternating group graphs can be constructed recursively, it is easy to see

that there exists a cycle of length 3 in $AG_n$, where $n \geq 4$. Fig. 4 shows a distribution of $4n - 12$ edge faults over $AG_n$, where $\langle w, u, v, w \rangle$ is a cycle and $(u, v)$, $(u, w)$ (respectively, $(v, u)$, $(v, w)$) are the only two fault-free edges incident to $u$ (respectively, $v$). It is easy to see that no fault-free Hamiltonian cycle exists in the faulty $AG_n$.

## 4 Concluding Remarks

Since processor faults and/or link faults may occur to multiprocessor systems, it is both practically significant and theoretically interesting to study the fault tolerance of multiprocessor systems. Most of previous works used the random fault model, which assumed that the faults might occur everywhere without any restriction. There was another fault model, i.e., the conditional fault model, which assumed that the fault distribution must satisfy some properties.

In this paper, adopting the conditional fault model and assuming that there were two or more fault-free edges incident to each vertex, we constructed a fault-free Hamiltonian cycle of an $n$-dimensional alternating group graph with up to $4n-13$ edge faults. The main construction methods were demonstrated in Fig. 3. This result is optimal with respect to the number of edge faults tolerated. Since an $n$-dimensional alternating group graph $AG_n$ is isomorphic to an $(n, n-2)$-arrangement graph $A_{n,n-2}$, our results and methods might be useful for people who want to solve Hamiltonian problems on faulty arrangement graphs under conditional fault model and the same assumption.

## Acknowledgments

The authors would like to thank the National Science Council of the Republic of China, Taiwan for financially supporting this research under Contract No. NSC 95-2221-E-239-002-.

## References

1. S. B. Akers, D. Horel, and B. Krishnamurthy. The Star Graph: An Attractive Alternative to the $n$-cube. *Proceedings of the International Conference on Parallel Processing*, pp. 393-400, 1987.
2. S. B. Akers and B. Krishnamurthy. A Group-Theoretic Model for Symmetric Interconnection Networks. *IEEE Transactions on Computers*, vol. 38 (4), pp. 555-566, 1989.
3. S. G. Akl. *Parallel Computation: Models and Methods*. Prentice Hall, NJ, 1997.
4. Y. A. Ashir and I. A. Stewart. Fault-Tolerant Embedding of Hamiltonian Circuits in $k$-ary $n$-cube. *SIAM Journal on Discrete Mathematics*, vol. 15 (3), pp. 317-328, 2002.
5. M. Y. Chan and S. J. Lee. On the Existence of Hamiltonian Circuits in Faulty Hypercubes. *SIAM Journal on Discrete Mathematics*, vol. 4 (4), pp. 511-527, 1991.

6. J. M. Chang, J. S. Yang, Y. L. Wang, and Y. Cheng. Panconnectivity, Fault-Tolerant Hamiltonicity and Hamiltonian-Connectivity in Alternating Group Graphs. *Networks*, vol. 44 (4), pp. 302-310, 2004.
7. W. K. Chiang and R. J. Chen. On the Arrangement Graph. *Information Processing Letters*, vol. 66, pp. 215-219, 1998.
8. K. Day and A. Tripathi. Arrangement Graphs: A Class of Generalized Star Graphs. *Information Processing Letters*, vol. 42, pp. 235-241, 1992.
9. J. S. Fu. Conditional Fault-Tolerant Hamiltonicity of Star Graphs. *Parallel Computing*, vol. 33, pp. 488-496, 2007.
10. H. C. Hsu, T. K. Li, Jimmy J. M. Tan, and L. H. Hsu. Fault Hamiltonicity and Fault Hamiltonian Connectivity of Arrangement Graphs. *IEEE Transactions on Computers*, vol. 53 (1), pp. 39-53, 2004.
11. J. S. Jwo, S. Lakshmivarahan, and S. K. Dhall. A New Class of Interconnection Networks Based on the Alternating Group. *Networks*, vol. 23, pp. 315-326, 1993.
12. S. Lakshmivarahan, J. S. Jwo, and S. K. Dhall. Symmetry in Interconnection Networks Based on Cayley Graphs of Permutation Groups: A Survey. *Parallel Computing*, vol. 19, pp. 1-47, 1993.
13. F. T. Leighton. *Introduction to Parallel Algorithms and Architecture: Arrays. Trees. Hypercubes*. Morgan Kaufman, CA, 1992.
14. Y. Saad and M. H. Schultz. Topological Properties of Hypercubes. *IEEE Transactions on Computers*, vol. 37 (7), pp. 867-872, 1988.
15. C. H. Tsai. Linear Array and Ring Embeddings in Conditional Faulty Hypercubes. *Theoretical Computer Science*, vol. 314, pp. 431-443, 2004.
16. P. Y. Tsai, J. S. Fu, and G. H. Chen. Hamiltonian-Laceability in Star Graphs with Conditional Edge Faults. *Proceedings of the International Computer Symposium*, pp. 144-152, 2006.
17. D. B. West. *Introduction to Graph Theory* (2nd Edition). Prenctice Hall, Upper Saddle River, 2001.
18. http://inrg.csie.ntu.edu.tw/∼bytsai/

# Regular Expression Matching Algorithms using Dual Position Automata *

Hiroaki Yamamoto

Department of Information Engineering, Shinshu University,
4-17-1 Wakasato, Nagano-shi, 380-8553 Japan.
yamamoto@cs.shinshu-u.ac.jp

**Abstract.** This paper introduces a new automaton model called *a dual position automaton* (a dual PA), and then presents a faster translation algorithm from a regular expression (RE) to a dual PA and RE matching algorithms using a dual PA. For any RE $r$ over an alphabet $\Sigma$, our translation algorithm generates a dual PA consisting of $\tilde{m}(\tilde{m}+1)$ bits in $O(\lceil \tilde{m}/w \rceil \tilde{m})$ time and space, where $w$ is the length of a computer word, $\tilde{m} = \sum_{a \in \Sigma} m_a$ and $m_a$ is the number of occurrences of alphabet symbol $a$ in $r$. Furthermore, we give a method to construct a compact DFA representation consisting of only $(\tilde{m}+1)\sum_{a \in \Sigma} 2^{m_a}$ bits from a dual PA. Using such a DFA representation, we can solve an RE matching problem fast. Finally, we will generalize our algorithm by introducing a parameter $K_a$ for each $a \in \Sigma$.

## 1 Introduction

Regular expressions (REs) pattern matching problems play an important role in the field of computer science, computational biology and so on. For this reason, RE pattern matching algorithms have intensively been studied [1, 3, 12–16]. We are here concerned with the following RE pattern matching problem: Let $r$ be an RE and let $x$ be a text string. Then the RE matching problem is to decide whether or not there is a substring $y$ of $x$ such that $y \in L(r)$, where $L(r)$ denotes the language generated by $r$. In general, RE matching algorithms are divided into two parts, a preprocessing part and a matching part. The preprocessing part translates a given RE into a finite automaton and the matching part does a matching job using the generated finite automaton. This time, many algorithms make use of nondeterministic finite automata (NFAs) because the translation can be efficiently done. Hence constructing NFAs with a smaller size for given REs is crucial in practical applications and a lot of research for efficiently generating smaller NFAs has been done (see [2, 5, 7–11]). Deterministic finite automata (DFAs) are also useful models for the RE matching problem because we can do an RE matching in a linear time. However, the DFA corresponding to an RE may become an exponential size in the length of the RE in the worst case.

* This research has been supported in part of Grant-in-Aid for Scientific Research, Ministry of Education, Culture, Sports, Science and Technology, Japan.

Hence constructing as compact a representation of DFAs as possible is desired. In techniques using automata, the time for constructing an NFA or a DFA from an RE is also crucial when the length of a given RE is large. Therefore, a faster translation algorithm from an RE to a finite automaton is also required. The aim of the paper is to introduce a new automaton model and to design efficient RE matching algorithms as well as a faster translation algorithm.

Two types of NFAs for REs are widely known, one is *a Thompson automaton* and the other is *a position automaton* (PA for short, also called *a Glushkov automaton*). Let $r$ be an RE over an alphabet $\Sigma$, and let $m$ be the total number of occurrences of alphabet symbols and operator symbols in $r$ and let $\tilde{m}$ be the number of occurrences of alphabet symbols in $r$. Without loss of generality, we may assume $m = O(\tilde{m})$ as mentioned in [7]. As seen in [8], a Thompson automaton is an NFA with $\epsilon$-moves and has at most $2m$ states and $4m$ transitions. Thompson automata can recursively be constructed based on the inductive definition of REs in $O(\tilde{m})$ time and space. Given a text string of length $n$, the traditional algorithm using a Thompson automaton solves the RE matching problem in $O(mn)$ time and $O(m)$ space. Myers [12] has improved it using the Four Russians technique so that his algorithm can solve the RE matching problem in $O(mn/\log n)$ time and space. Recently, Bille [4] has proposed a new algorithm which improves $O(mn)$ time while preserving $O(m)$ space. His algorithm is also based on a Thompson automaton.

On the other hand, a PA is an $\epsilon$-free NFA (that is, an NFA without any $\epsilon$-moves) and has exactly $\tilde{m}+1$ states and at most $\tilde{m}^2+\tilde{m}$ transitions. PAs are also important models in practical applications because they become smaller than Thompson automata for some kinds of REs. For this reason, some studies for efficiently constructing PAs have been done, and $O(\tilde{m}^2)$ time and space algorithms have been developed [5, 6]. Recently, Yamamoto, Miyazaki and Okamoto [17] have presented a faster bit-parallel algorithm generating a PA and related automata. PAs have another important property that for any state, *all incoming transitions* to the state have the same symbol. This property has a potential for improving complexities of RE matching algorithms. Indeed, Navarro and Raffinot [14, 15] have made use of this property to obtain a compact DFA representation and have presented a faster RE matching algorithm. Their compact DFA representation requires only $O(\tilde{m}2^{\tilde{m}})$ bits while a traditional DFA representation obtained from a Thompson automaton requires $O(m2^{2m})$ bits.

In this paper, we will introduce a new automaton model called *a dual position automaton* (dual PA), and present an efficient translation algorithm from an RE to a dual PA and RE matching algorithms. A dual PA is also an $\epsilon$-free NFA with exactly $\tilde{m}+1$ states and at most $\tilde{m}^2+\tilde{m}$ transitions. Unlike a PA, however, it has a property that for any state, *all outgoing transitions* from the state have the same symbol. Clearly, by reversing a PA, we can get an NFA satisfying such a property, but it accepts a reversed string. We first give a faster algorithm for translating an RE into a dual PA using a Thompson automaton. Our algorithm translates a given RE into a dual PA in $O(\tilde{m}\lceil\tilde{m}/w\rceil)$ time and space, where $w$ is the length of a computer word. Furthermore, a generated dual PA is represented

with $\tilde{m}(\tilde{m}+1)$ bits. Hence if $\tilde{m} = O(w)$, then the algorithm runs in $O(\tilde{m})$ time and space. To the best of our knowledge, we do not know any algorithms which directly generate a dual PA from $r$. Furthermore, it would take $O(\tilde{m}^2)$ time even though a similar technique to a construction of a PA is used because it takes $O(\tilde{m}^2)$ time.

Next we show a compact DFA representation obtained from a dual PA. We improve an idea of Navarro and Raffinot [14, 15] by grouping all states of a dual PA by a symbol on the outgoing transitions. That is, we partition the set of states into at most $|\Sigma|$ subsets $Q_a$ according to a symbol $a$ on transitions. In other words, if the outgoing transitions of states $q$ and $p$ have the same symbol, then $q$ and $p$ belong to the same subset. Then, for each subset $Q_a$, we make DFA transitions. In addition, we introduce an idea of a lookahead symbol to a matching algorithm. By these improvements, a matching time gets to depend on $m_\alpha$ but not on $\tilde{m}$, where $m_\alpha$ is defined as $m_\alpha = \max\{m_a \mid a \in \Sigma\}$ when $m_a$ denotes the number of occurrences of an alphabet symbol $a$ in $r$. Our DFA representation requires only $(\tilde{m}+1)\sum_{a\in\Sigma} 2^{m_a}$ bits. Since Navarro and Raffinot's representation is $(\tilde{m}+1)\prod_{a\in\Sigma} 2^{m_a}$ bits, ours improves the space. Especially, the space required would be much smaller when an RE consists of many kinds of symbols. The RE matching algorithm using our representation also runs in $O(n\lceil m_\alpha/w\rceil)$ time. Furthermore, the preprocessing time for constructing a DFA representation from $r$ is $O(\lceil\tilde{m}/w\rceil(\tilde{m}+\sum_{a\in\Sigma} 2^{m_a}))$. Hence $m_\alpha = O(w)$, then the matching time is $O(n)$ and the preprocessing time is $O(\tilde{m}+\sum_{a\in\Sigma} 2^{m_a})$.

Finally, we apply a decomposition technique to our algorithm. Although Navarro and Raffinot [15] decomposed the whole set of states, we decompose each subset $Q_a$. That is, we introduce a parameter $1 \le K_a \le w$ for each $a \in \Sigma$ and decompose each $Q_a$ by a parameter $K_a$ into $\lceil m_a/K_a\rceil$ subsets. Again, by this improvement, a matching time gets to depend on $m_\alpha$ but not on $\tilde{m}$. Indeed, we show an RE matching algorithm running in $O(\lceil m_\alpha^2/(wK_\alpha)\rceil n)$ time using $O((\tilde{m}+1)\sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a})$ bits. Hence if $K_a = O(\log n)$ and $\tilde{m} = O(w)$, then we get a RE matching algorithm running in $O(\lceil m_\alpha/\log n\rceil n)$ time and $O(\tilde{m}n/\log n)$ space. This time, the preprocessing time is $O(\tilde{m}+\tilde{m}n/\log n)$.

We will rely on a $w$-bit uniform RAM to estimate the complexities of algorithms. In general, since most papers assume $w \ge \log n$, we also do so. Note that we will describe algorithms using $\tilde{m}$-bit vectors or $m_\alpha$-bit vectors for the sake of convenience. In a practical implementation, however, these bit vectors must be divided into $w$-bit vectors. Hence factors of $\lceil\tilde{m}/w\rceil$ and $\lceil m_\alpha/w\rceil$ appear in the time and space.

The paper is organized as follows. In Section 2, we will give basic definitions of REs. In Section 3, we will explain a Thompson automaton and a dual PA, and then give a translation algorithm from an RE into a dual PA. In Section 4 we will give RE matching algorithms.

## 2 Regular Expressions and Some Notations

We here give some definitions for regular expressions.

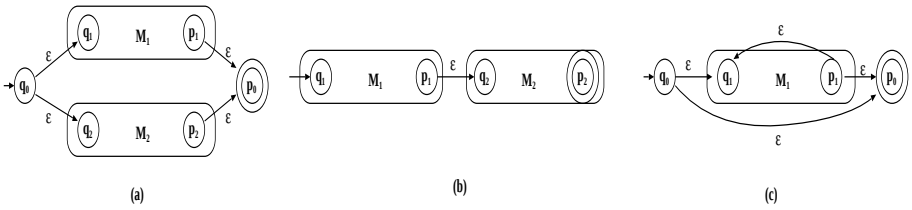



**Fig. 1.** Translation of an RE into an NFA. (a) union, (b) concatenation and (c) Kleene closure

**Definition 1.** *Let $\Sigma$ be an alphabet. The regular expressions (REs) over $\Sigma$ are defined as follows.*

1. *$\emptyset$, $\epsilon$ (the empty string) and $a$ ($\in \Sigma$) are REs that denote the empty set, the set $\{\epsilon\}$ and the set $\{a\}$, respectively.*
2. *Let $r_1$ and $r_2$ be REs denoting the sets $R_1$ and $R_2$, respectively. Then $(r_1 \vee r_2)$, $(r_1 r_2)$ and $(r_1^*)$ are also REs that denote the sets $R_1 \cup R_2$ (union), $R_1 R_2$ (concatenation), and $R_1^*$ (Kleene closure or star), respectively.*

In this paper, we use some notations as follows.

- By $L(r)$ we denote the language generated by an RE $r$.
- By $m$ we denote the number of occurrences of alphabet symbols and operator symbols in $r$. The length of $r$ means this $m$.
- By $m_a$ we denote the number of occurrences of an alphabet symbol $a$ in $r$. In addition, we define $\tilde{m} = \sum_{a \in \Sigma} m_a$ and $m_\alpha = \max\{m_a \mid a \in \Sigma\}$.

As mentioned in Introduction, we may assume $m = O(\tilde{m})$. Hence we will use a parameter $\tilde{m}$ to state our results.

## 3 From Thompson Automata to Dual Position Automata

### 3.1 Thompson Automata

Thompson automata are recursively constructed based on the definition of REs, whose construction algorithm is widely known (for example, see [8]). We give an outline of the construction in Fig. 1. In Fig.1, (a), (b) and (c) show recursive constructions for union $(r_1 \vee r_2)$, concatenation $(r_1 r_2)$ and Kleene closure $(r_1^*)$, respectively. Here $M_1$ and $M_2$ denote NFAs for REs $r_1$ and $r_2$, respectively. Let $M = (Q, \Sigma, \delta, q_0, q_f)$ be a Thompson automata obtained from an RE $r$ of length $m$, where $Q$ is a set of states, $\Sigma$ is an alphabet, $\delta$ is a transition function, $q_0$ is the initial state and $q_f$ is the final state. Note that a Thompson automaton has just one initial state and one final state. Then $M$ has at most $2m$ states and $4m$ transitions. Furthermore, for any state $q \in Q$, all outgoing transitions from $q$ are caused either by the empty string $\epsilon$ or by an alphabet

symbol $a \in \Sigma$. If the transitions from $q$ are caused by an alphabet symbol, then we call state $q$ *a sym-state*; otherwise *an $\epsilon$-state*. We partition a set of *sym*-states into several subsets according to an alphabet symbol. Let us call a *sym*-state *an $a$-state* if the alphabet symbol of the *sym*-state is $a \in \Sigma$. Then we define $Q_a = \{q \mid q \text{ is an } a\text{-state}\}$. Such subsets $Q_a$ play an important role in improving an efficiency of an algorithm. We have the following proposition.

**Proposition 1.** *For any RE $r$ of length $m$, we can construct the Thompson automaton with at most $2m$ states and $4m$ transitions in $O(\tilde{m})$ time and space.*

We define the reversed automaton $M^R$ of $M$ by reversing all transitions and interchanging the initial state and the final state, but remaining *sym*-states and $\epsilon$-state unchanged. That is, for any states $q$ and $p$ of $M$, $p \in \delta(q, a)$ if and only if $q \in \delta^R(p, a)$, where $\delta^R$ is the transition function of $M^R$. Hence if $q$ is a *sym*-state in $M$, then the incoming transition is done by an alphabet symbol in $M^R$.

Let us introduce some notions for $M$ and $M^R$. We call a state with two incoming transitions *a junction state*. Also we say that a state $p$ is *a predecessor* of a state $q$ if there is a transition from $p$ to $q$. Furthermore, a sequences of transitions from a state to a state is called *a path*. As seen in (c) of Fig.1, a star operator generates a transition going back to a previous state. We call this transition *a back transition*. That is, the transition from $p_1$ to $q_1$ is a back transition. Note that a back transition in $M$ becomes a back transition also in $M^R$. By removing such back transitions from $M$, we can sort the states of $M$ from the initial state in a topological order. Here, by a topological order, we mean that for any states $p$ and $q$, $p < q$ if and only if there is a directed path from $p$ to $q$. It is clear that a topological order of states of $M^R$ is defined by reversing that of $M$.

Thompson automata have the following important property.

**Lemma 1 ([13],Lemma 1).** *Let $M$ be a Thompson automaton. Then, any loop-free path in $M$ has at most one back transition.*

By the symmetrical structure of a Thompson automaton $M$, we notice that Proposition 1 and Lemma 1 also hold for $M^R$. With Lemma 1 and a reversed automaton, we can efficiently generate a bit vector representation of a dual position automaton.

*Example 1.* Let us consider an RE $r = (00 \vee 10)^*1$ over $\Sigma = \{0, 1\}$. Fig. 2 shows the Thompson automaton for $r$.

### 3.2 Position Automata and Dual Position Automata

First we will explain a position automaton (a PA). Let $r$ be an RE with $\tilde{m}$ occurrences of alphabet symbols. We number all alphabet symbols in $r$ with its position and denote the obtained RE by $\bar{r}$. The existing algorithms for generating a PA have focused on computing sets of positions **First**, **Last** and **Follow(i)**. Here the set **First** is defined to be $\{i \mid a_i\alpha \in L(\bar{r})\}$, the set **Last** is defined to be $\{i \mid \alpha a_i \in L(\bar{r})\}$, and the set **Follow(i)** is defined to be $\{j \mid \alpha a_i a_j \beta \in L(\bar{r})\}$.

A PA $G$ is an $\epsilon$-free NFA and has the following properties.

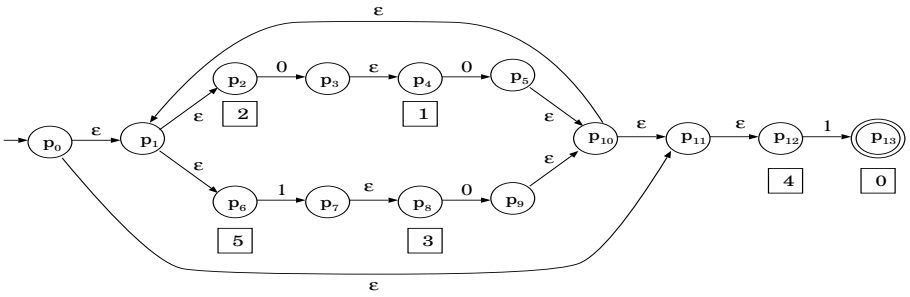



**Fig. 2.** Thompson automaton for $r = (00 \vee 10)^*1$

**Property 1** For any state $q$ of $G$, all incoming transitions to $q$ are activated by the same alphabet symbol $a \in \Sigma$.

**Property 2** The number of initial states is just one, and the number of final states is more than or equal to one.

**Property 3** The number of states is just $\tilde{m} + 1$ and the number of transitions is at most $\tilde{m}^2 + \tilde{m}$.

Now let us define a dual version of PAs. For any RE $r$, we call an NFA $\bar{G}$ satisfying the following properties *a dual position automaton* (a dual PA).

**Property 1'** For any state $q$ of $\bar{G}$, all outgoing transitions from $q$ are activated by the same alphabet symbol $a \in \Sigma$.

**Property 2'** The number of initial states is more than or equal to one, and the number of final states is just one.

**Property 3** The number of states is just $\tilde{m} + 1$ and the number of transitions is at most $\tilde{m}^2 + \tilde{m}$.

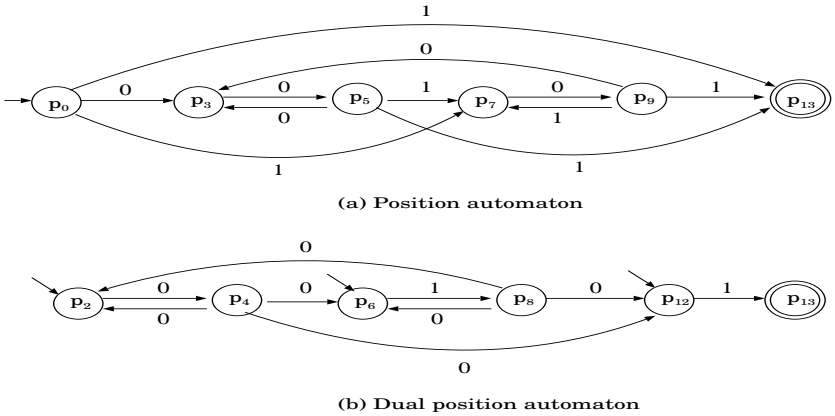
As we can see, Property 1 and 1', and Property 2 and 2' become symmetric. Property 3 holds for both of a PA and a dual PA.

We will make use of a dual PA for an RE matching algorithm. Hence, to generate a bit representation of a dual PA, we also make use of a Thompson automaton as in Yamamoto [17].

*Example 2.* Let us consider an RE $r = (00 \vee 10)^*1$. Fig. 3 shows the PA and the dual PA for $r$. The PA is made from the Thompson automaton in Fig. 2 by considering only states $p_3, p_5, p_7, p_9, p_{13}$ and the initial state $p_0$. On the other hand, the dual PA is made by considering only states $p_2, p_4, p_6, p_8, p_{12}$ and the final state $p_{13}$. The PA has one initial state $p_0$ and one final state $p_{13}$, while the dual PA has three initial states $p_2, p_6, p_{12}$ and one final state $p_{13}$.

### 3.3 Bit-Parallel Translation Algorithm from an RE to a Dual PA

We focus on *sym*-states and the final state of a Thompson automaton. By computing the reachability between these states, we can construct a dual PA from




(a) Position automaton

(b) Dual position automaton

**Fig. 3.** Position automaton and dual position automaton for $r = (00 \vee 10)^*1$

a Thompson automaton. We will present an efficient bit-parallel algorithm for translating an RE into a dual PA by using a Thompson automaton. We number *sym*-states to implement a set of *sym*-states as a bit vector. To do this, when $\Sigma = \{a_1, \cdots, a_l\}$, we number states in order of $Q_{a_1}$, $Q_{a_2}$, $\cdots$, $Q_{a_l}$. That is, states of $Q_{a_1}$ are numbered from 1 to $m_{a_1}$, states of $Q_{a_2}$ are numbered from $m_{a_1}+1$ to $m_{a_1}+m_{a_2}$, $\cdots$, and states of $Q_{a_l}$ are numbered from $\sum_{j<l} m_{a_j}+1$ to $\sum_{j\leq l} m_{a_j}$. We assign the number 0 to the final state. For any state $q$, we denote by $num(q)$ a number assigned to $q$. In Fig. 2, a figure attached to each *sym*-state is the number of the state.

Let $r$ be an RE over $\Sigma$ and let $M = (Q, \Sigma, \delta, q_0, q_f)$ be the Thompson automaton constructed from $r$. To compute the transition function of a dual PA, we compute an array $NEXT[q, \sigma]$ whose elements are bit vectors of $\tilde{m}+1$ bits, where $q \in Q$ and $\sigma \in \Sigma \cup \{\epsilon\}$. Here note that if $\sigma = a \in \Sigma$, then $q \in Q_a$. In addition, the array $NEXT[q, \sigma]$ satisfies that the $i$th bit of $NEXT[q, \sigma]$ is equal to 1 if and only if there is a path from state $q$ to a *sym*-state $p$ with number $i$. If $q$ is an $a$-state, then this means that $M$ can move from state $q$ to $p$ by an alphabet symbol $a$. We can efficiently compute $NEXT[q, a]$ by using the property of Lemma 1 and the reversed automaton $M^R$ of $M$. Note that Lemma 1 also holds for $M^R$. Finally, the desired dual PA is represented by $NEXT[q, a]$ on all *sym*-states $q$ and alphabet symbols $a$. Hence the size of $NEXT[q, a]$ becomes $\tilde{m}(\tilde{m}+1)$ bits.

The algorithm starts with *REtoDPA* given in Fig. 4. In the algorithm, the operator | denotes bitwise OR. Furthermore, we use two functions *BitSet* and *BitCheck*. *BitSet*$(v, i)$ sets the $i$th bit of $v$ to 1. *BitCheck*$(v, i)$ checks whether or not the $i$th bit of $v$ is equal to 1, and if equal, then it returns 1; otherwise returns 0. Since these functions can easily be implemented so that they can run in $O(1)$ time, the details are omitted here. For any RE $r$, *REtoDPA* translates $r$ into the

**Algorithm REtoDPA($r$)**
Input: an RE $r$.
Output: Dual PA $\bar{G} = (Q', \Sigma, \delta', I, q_f)$.

**Step 1.** Translate $r$ into the Thompson automaton $M$. This time, also compute $\Sigma_r$ which is the set of alphabet symbols occurring in $r$.
**Step 2.** For all $a \in \Sigma_r$, number each state of $Q_a$.
**Step 3.** Let $q_0$ be the initial state of $M$. Then, add a new initial state $q_{ini}$ and a transition from $q_{ini}$ to $q_0$ by $\epsilon$ to $M$.
**Step 4.** For all states $q$ of $M$, if $q$ is an $a$-state for an alphabet symbol $a \in \Sigma_r$, then $NEXT[q, a] := 0$; otherwise $NEXT[q, \epsilon] := 0$.
**Step 5.** Generate $M^R$ from $M$ and do *ReachState*($M^R$,*NEXT*,$\Sigma_r$).
**Step 6.** Generate $\bar{G}$ as follows:
1. define $Q'$ to be the set $\{q \mid q$ is a *sym*-state or the final of $M\}$,
2. define the final state $q_f$ to be the final state of $M$,
3. for all $a \in \Sigma$
   for all $q \in Q_a$, define $\delta'(q, a)$ to be $NEXT[q, a]$.
4. for all states $q \in Q'$, if $BitCheck(NEXT[q_{ini}, \epsilon], num(q)) = 1$, then add $q$ to $I$.

**Fig. 4.** The algorithm *REtoDPA*

Thompson automaton $M$, and then invokes the procedure *ReachState* for $M^R$. The procedure *ReachState*, given in Fig. 5, computes array $NEXT[q, a]$ using $M^R$. This time, Lemma 1 guarantees that we can correctly compute $NEXT[q, a]$ by traversing all states of $M^R$ twice in a topological order. We have the following theorem.

**Theorem 1.** *Let $r$ be an RE with $\tilde{m}$ occurrences of alphabet symbols. Then the algorithm* REtoDPA *correctly translates $r$ into the dual PA in $O(\tilde{m}\lceil\tilde{m}/w\rceil)$ time and space. If $\tilde{m} = O(w)$, then it runs in $O(\tilde{m})$ time and space.*

The proof will be given in appendix.

## 4 RE Matching Algorithms

We first give a compact DFA representation and a matching algorithm using it, and then extend them by grouping parameters $K_a$.

### 4.1 Compact DFA Representation and Matching Algorithm

We generate a compact DFA representation $D[a, b, I_a]$ and $FINAL[a]$ from a dual PA $\bar{G}$ with $NEXT[p, a]$, where $a, b \in \Sigma$ and $0 \le I_a \le 2^{m_a} - 1$. The element of $D[a, b, I_a]$ is a bit vector of $m_b$ bits, and if $D[a, b, I_a] = I_b$ ($0 \le I_b \le 2^{m_b} - 1$), then it means that there is a transition from a subset $I_a$ of $a$-states to a subset $I_b$ of $b$-states in $\bar{G}$. Note that $I_a$ is a bit vector representation for the set $Q_a$ of $a$-states of $\bar{G}$. The element $v$ of array $FINAL[a]$ is a bit vector of $2^{m_a} - 1$ bits, which satisfies

**Procedure ReachState($M^R$,*NEXT*,$\Sigma_r$)**

Repeat the following twice:
for all states $q$ of $M^R$ do the following in a topological order:
1. if $q$ is the initial state, then $BitSet(NEXT[q,\epsilon], num(q))$,
2. if $q$ is an $a$-state with an incoming transition from a state $p_1$ (note that in this case $q$ has just one incoming transition), then $NEXT[q,a] := NEXT[p_1,\epsilon]$ and $BitSet(NEXT[q,\epsilon], num(q))$,
3. if $q$ is a junction state, then $NEXT[q,\epsilon] := NEXT[p_1,\epsilon] \mid NEXT[p_2,\epsilon]$, where $p_1$ and $p_2$ are two predecessors of $q$;
4. otherwise $NEXT[q,\epsilon] := NEXT[p_1,\epsilon]$, where $p_1$ is a predecessor of $q$.

**Fig. 5.** The procedure *ReachState*

that the $i$-th bit of $v$ is 1 if and only if there is a transition on $a$ from a state of the $i$-th subset of $Q_a$ to the final state of $\bar{G}$, where subsets of $Q_a$ are numbered from 0 to $2^{m_a}-1$. Then $D[a,b,I_a]$ and $FINAL[a]$ are generated by procedure $GenDFA(\bar{G})$ given in Fig.7, which is an extension of a technique used in [14, 15]. To compute $D[a,b,I_a]$, we first compute the array $E[I_a]$ denoting a transition from a set $I_a$ of $a$-states to *sym*-states, and then compute a transition from a set $I_a$ of $a$-states to $b$-states. The size of $D[a,b,I_a]$ is $\tilde{m}\sum_{a\in\Sigma}2^{m_a}$ bits because $\tilde{m}=\sum_{a\in\Sigma}m_a$, and the size of $FINAL[a]$ is $\sum_{a\in\Sigma}2^{m_a}$ bits. Hence the total size is $(\tilde{m}+1)\sum_{a\in\Sigma}2^{m_a}$ bits. In the algorithm, the operator & denotes bitwise *AND* and the operator $>>$ denotes *Shift Right*. Furthermore, $B[b]$ denotes a bit-mask for $b$-states, that is, its value is an $\tilde{m}+1$-bit vector and only bits corresponding to $b$-states are 1; other bits are 0.

The matching algorithm $REMatchDFA(r,x)$, given in Fig.6, outputs endpoints of all substrings of $x$ matching $r$. This algorithm makes use of one lookahead symbol, if any, to hold only $a$-states for some alphabet symbol $a$ at a time. In $D[a,b,I_a]$, symbol $a$ corresponds to current symbol and symbol $b$ corresponds to the next symbol (that is, the lookahead symbol).

Now let us check the time for generating a DFA representation $D[a,b,I_a]$ from a given RE $r$. By Theorem 1, it takes $O(\tilde{m})$ time to generate a dual PA $M$. The procedure $GenDFA(\bar{G})$ generates $D[a,b,I_a]$ in $O(\lceil\tilde{m}/w\rceil(\sum_{a\in\Sigma}2^{m_a}))$ time. Hence it takes $O(\lceil\tilde{m}/w\rceil(\tilde{m}+\sum_{a\in\Sigma}2^{m_a}))$ time in total. We get the following theorem. Here, as mentioned before, $\tilde{m}$ is the number of occurrences of alphabet symbols in an given RE $r$ and $m_a$ is the number of occurrences of an alphabet symbol $a$, and $m_\alpha=\max\{m_a \mid a\in\Sigma\}$.

**Theorem 2.** *The matching part of the algorithm* REMatchDFA *runs in* $O(n\lceil m_\alpha/w\rceil)$ *time using* $O((\tilde{m}+1)\sum_{a\in\Sigma}2^{m_a})$ *bits. Furthermore, the preprocessing time for constructing a DFA representation is* $O(\lceil\tilde{m}/w\rceil(\tilde{m}+\sum_{a\in\Sigma}2^{m_a}))$. *If* $\tilde{m}=O(w)$, *then the matching time is* $O(n)$ *and the preprocessing time is* $O(\tilde{m}+\sum_{a\in\Sigma}2^{m_a})$.

**Algorithm REMatchDFA($r$,$x$)**
Input: an RE $r$ and a text string $x = x_1 \cdots x_n$, where $x_i \in \Sigma$.
Output: endpoints $i$ of all substring of $x$ matching $r$.

**Step 1.** Generate a dual PA $\bar{G} = (Q, \Sigma, \delta, I_q, q_f)$ using *REtoDPA*($r$).
**Step 2.** Generate a DFA $D[a, b, I_a]$ using *GenDFA*($\bar{G}$).
**Step 3.** /* Setting initial states. Each bit vector $INIT[a]$ is set for all initial $a$-states */
1. $INIT := 0$, /* $INIT$ is an $\tilde{m}+1$-bit vector */
2. for all states $q \in I_q$, *BitSet*($INIT$, *num*($q$)),
3. $J := 1$,
4. for $a = a_1, \cdots, a_l$,
   (a) $INIT[a] := (INIT \ \& \ B[a]) >> J$,
   (b) $J := J + m_a$,

**Step 4.** if the final state $q_f$ is included in $I_q$, then output 0, /* This means that $\epsilon$ matches $r$ */
**Step 5.** $CSTATE := INIT[x_1]$.
**Step 6.** for $i = 1$ to $n-1$ do
1. if *BitCheck*($FINAL[x_i]$, $CSTATE$]) $= 1$, then output the position $i$,
2. $CSTATE := D[x_i, x_{i+1}, CSTATE]$,
3. /* set self-loop on initial states to find all substrings matching $r$*/
   $CSTATE := CSTATE \mid INIT[x_{i+1}]$

**Step 7.** if *BitCheck*($FINAL[x_n]$, $CSTATE$]) $= 1$, then output the position $n$

**Fig. 6.** The algorithm *REMatchDFA*

### 4.2 Generalization of Algorithm by Grouping States

Navarro and Raffinot [15] partition the whole set of states of a PA into several subsets and construct DFA-like transitions for each subset of states. For each $a \in \Sigma$, we introduce a parameter $1 \le K_a \le w$, and then partition each subset $Q_a$ but not the whole set into $t = \lceil m_a/K_a \rceil$ subsets $Q_a^0, \cdots, Q_a^{t-1}$ each of which consists of $K_a$ states. Then we generate a DFA representation for every subset. We call such a DFA representation *a partial DFA representation*, which is represented by an array $D[a, b, I_a^{h_a}, h_a]$, where $a, b \in \Sigma$, $0 \le I_a^{h_a} \le 2^{K_a}$ and $0 \le h_a \le t-1$. The array $D[a, b, I_a^{h_a}, h_a]$ is computed by the procedure *GenPartial* given in Fig. 9 and satisfies

$$D[a, b, I_a] = D[a, b, I_a^0, 0] \mid D[a, b, I_a^1, 1] \mid \cdots \mid D[a, b, I_a^{t-1}, t-1]$$

This leads to a more efficient matching algorithm *REMatchPartial* given in Fig.8.

**Theorem 3.** *The matching part of the algorithm* REMatchPartial *runs in* $O(\lceil m_\alpha^2/(wK_\alpha)\rceil n)$ *time using* $O((\tilde{m}+1)\sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a})$ *bits. Furthermore the preprocessing time for constructing a partial DFA representation is* $O(\lceil \tilde{m}/w\rceil(\tilde{m} + \sum_{a\in\Sigma}\lceil m_a/K_a\rceil 2^{K_a}))$.

Each parameter $K_a$ can be regarded as a kind of a measure to decide a degree of determinism. If $\tilde{m} = O(w)$ and $K_a = O(\log n)$, then we get an RE

**Procedure GenDFA($\bar{G}$)**
Input: a dual PA $\bar{G}$ with $NEXT[q, a]$.

```
1.  I := 1 and E[0] := 0
2.  for a = a_1, ..., a_l /* Σ = {a_1, ..., a_l} */
3.      for i = 0, ..., m_a − 1
4.          for j = 0, ..., 2^i − 1
5.              E[2^i + j] := E[j] | NEXT[I + i, a]
6.              if BitCheck(E[2^i + j], 0) = 1, then BitSet(FINAL[a], 2^i + j)
7.              J := 1
8.              for b = a_1, ..., a_l
9.                  D[a, b, 2^i + j] := (E[2^i + j] & B[b]) >> J
10.                 J := J + m_b
11.             for-end
12.         for-end
13.     for-end
14.     I := I + m_a
15. for-end
```

**Fig. 7.** The algorithm *GenDFA*

matching algorithm running in $O(m_\alpha n/\log n)$ time with preprocessing time $O(\tilde{m} + \tilde{m}n/\log n)$.

## References


1. A.V. Aho, *Algorithms for finding patterns in strings*, In J.V. Leeuwen, ed. Handbook of theoretical computer science, Elsevier Science Pub., 1990.
2. V. Antimirov, *Partial derivative of regular expressions and finite automaton construction*, Theoret. Comput. Sci., 155, 291-319, 1996.
3. A. Apostolico, Z. Galil ed., *Pattern Matching Algorithms*, Oxford University Press, 1997.
4. P. Bille, *New Algorithms for Regular Expression Matching*, Proc. of ICALP 2006, LNCS4501, 643-654, 2006.
5. A. Brüggemann-Klein, *Regular expressions into finite automata*, Theoret. Comput. Sci., 120, 197-213, 1993.
6. C.H. Chang and R. Paige, *From regular expressions to DFA's using compressed NFA's*, Theoret. Comput. Sci., 178, 1-36, 1997.
7. J.-M. Champarnaud, *Evaluation of three implicit structures to implement nondeterministic automata from regular expressions*, IJFCS, 13, 99-113, 2002.
8. J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory language and computation*, Addison Wesley, Reading Mass, 1979.
9. J. Hromkovič, S. Seibert and T. Wilke, *Translating Regular Expressions into Small $\epsilon$-free Nondeterministic Finite Automata*, JCSS, 62, 565-588, 2001.
10. L. Ilie and S. Yu, *Constructing NFAs by optimal use of positions in regular expressions*, Proc. of CPM2002, LNCS2373,279-288, 2002.
11. L. Ilie and S. Yu, *Follow Automata*, Information and Computation,186,140-162, 2003.

**Algorithm REMatchPartial($r$,$x$)**
Input: an RE $r$ and a text string $x = x_1 \cdots x_n$, where $x_i \in \Sigma$..
Output: endpoints $i$ of all substring of $x$ matching $r$.

**Step 1.** Generate a dual PA $\bar{G} = (Q, \Sigma, \delta, I_q, q_f)$ using $REtoDPA(r)$.
**Step 2.** Generate a DFA $D[a, b, I_a]$ using $GenDFA(\bar{G})$.
**Step 3.** /* Setting initial states. Each bit vector $INIT[a]$ is set for all initial $a$-states */
1. $INIT := 0$, /* $INIT$ is an $\tilde{m} + 1$-bit vector */
2. for all states $q \in I_q$, $BitSet(INIT, num(q))$,
3. $J := 1$,
4. for $a = a_1, \cdots, a_l$,
   (a) $INIT[a] := (INIT \ \& \ B[a]) >> J$,
   (b) $J := J + m_a$,

**Step 4.** if the final state $q_f$ is included in $I_q$, then output 0, /* This means that $\epsilon$ matches $r$ */
**Step 5.** $CSTATE := INIT[x_1]$.
**Step 6.** for $i = 1$ to $n - 1$ do
1. if $BitCheck(FINAL[x_i], CSTATE) = 1$, then output $i$,
2. $Temp := 0$ and $J := 0$,
3. for $h = 0, \cdots, \lceil m_{x_i}/K_{x_i} \rceil - 1$,
   (a) $KSTATE := CSTATE \ \& \ 0 \cdots 01^{K_{x_i}}$, /* extract $K_{x_i}$ bits from $CSTATE$ */
   (b) $Temp := Temp \mid D[x_i, x_{i+1}, KSTATE, h]$,
   (c) $J := J + K_{x_i}$,
   (d) $CSTATE := CSTATE >> J$,
4. $CSTATE := Temp \mid INIT[x_{i+1}]$, /* update the current state and set self-loop */

**Step 7.** if $BitCheck(FINAL[x_n], CSTATE) = 1$, then output $n$,

**Fig. 8.** The algorithm *REMatchPartial*


12. G. Myers, *A Four Russians Algorithm for Regular Expression Pattern Matching*, J. ACM. 39, 4, 430-448, 1992.
13. E. Myers and W. Miller, Approximate Matching of Regular Expressions, Bull. of Mathematical Biology, 51, 1, 5-37, 1989.
14. G. Navarro and M. Raffinot, *Compact DFA Representation for Fast Regular Expression Search*, Proc. of WAE2001, LNCS 2141, 1-12, 2001.
15. G. Navarro and M. Raffinot, *New Techniques for Regular Expression Searching*, Algorithmica, 41, 89-116, 2004.
16. S. Wu, U. Manber and E. Myers, A Sub-Quadratic Algorithm for Approximate Regular Expression Matching, J. of Algorithm, 19, 346-360, 1995.
17. H. Yamamoto, T. Miyazaki and M. Okamoto, *Bit-Parallel Algorithms for Translating Regular Expressions into NFAs*, IEICE Trans. Inf. & Syst., Vol.E90-D, No.2, 418-427, 2007.

**Procedure GenPartial($\bar{G}$)**
Input: a dual PA $\bar{G}$ with $NEXT[q, a]$.

```
1. I := 1 and E[0] := 0
2. for a = a_1, ..., a_l /* Σ = {a_1, ..., a_l} */
3.     for h = 0, ..., ⌈m_a/K_a⌉ − 1
4.       if h ≠ ⌈m_a/K_a⌉ − 1, then K := K_a; otherwise K := m_a − K_a h
5.       for i = 0, ..., K − 1
6.         for j = 0, ..., 2^i − 1
7.           E[2^i + j] := E[j] | NEXT[I + i, a]
8.           if BitCheck(E[2^i + j], 0) = 1, then BitSet(FINAL[a], 2^i + j)
9.           J := 1
10.          for b = a_1, ..., a_l
11.            D[a, b, 2^i + j, h] := (E[2^i + j] & B[b]) >> J
12.            J := J + m_b
13.          for-end
14.        for-end
15.        I := I + K + 1
16.      for-end
17.    for-end
18. for-end
```

**Fig. 9.** The algorithm *GenPartial*

# Appendix

## 5 Proof of Theorem 1

Let $M$ be the Thompson automaton for a given RE $r$. First we will prove the following lemma to show the correctness of the algorithm.

**Lemma 2.** *The ith bit of $NEXT[q, a]$ becomes* 1 *if and only if $M$ can move from an a-state $q$ to a sym-state $p$ with $num(p) = i$ by the alphabet symbol $a$.*

*Proof.* First let us show the *only-if-part*. For any states $q$ and $p$ of $M$, it is clear that there is a path from $q$ to $p$ if and only if there is a path from $p$ to $q$ in $M^R$. We can easily see that if the $i$th bit of $NEXT[q, \sigma]$ is set to 1 by *ReachState*, then for a *sym*-state $p$ with $num(p) = i$, there is a path from $p$ to $q$ in $M^R$. This time, if $q$ is an $a$-state for any alphabet symbol $a$, then $\sigma = a$ and the sequence of symbols over the path is $\epsilon \cdots \epsilon \cdot a$. That is, this means that $M$ can move from $q$ to $p$ by the alphabet symbol $a$.

Next let us show the reverse direction, that is, the *if-part*. To do this, it is sufficient to show the following claim. Here $Q' = \{q \mid$ $q$ is a *sym*-state or the final state of $M$ $\}$.

*Claim.* For any states $q, p \in Q'$, if $M$ can move from $q$ to $p$ by a symbol $a$, then *ReachState* sets the $num(p)$th bit of $NEXT[q, a]$ to 1.

*Proof of the claim.* Since $M$ can move from $q$ to $p$ by a symbol $a$, there is a loop-free path from $q$ to $p$ in $M$. Hence the reversed path from $p$ to $q$ is also loop-free in $M^R$. Furthermore, this reversed path have at most one back transition because Lemma 1 holds for $M^R$. Now let this path be $Z = p_1(= p), p_2, \ldots, p_t(= q)$. If $Z$ does not contain any back transitions, then *ReachState* sets the $num(p)$th bit of $NEXT[q, a]$ to 1 in the first traverse of Step 2. This is because we have $p_1 < p_2 < \cdots < p_t$ in a topological order and all states other than $q$ and $p$ are $\epsilon$-states. Next suppose that $Z$ contains one back transition. Here, without loss of generality, let the transition from $p_l$ to $p_{l+1}$ be a back transition. This time, we have $p_1 < \cdots < p_l$ and $p_{l+1} < \cdots < p_t$ in a topological order. Hence, since *ReachState* can see that there is a path from $p$ to $p_l$ in the first traverse of Step 2, it sets the $num(p)$th bit of $NEXT[p_l, \epsilon]$ to 1. However, at this moment, it may not know whether or not there is a path from $p_l$ to $q$. In the second traverse of Step 2, *ReachState* can see that there is a path from $p$ to $q$ because it can use the value of $NEXT[p_l, \epsilon]$ by the back transition from $p_l$ to $p_{l+1}$. Hence *ReachState* sets the $num(p)$th bit of $NEXT[q, a]$ to 1. Thus we have the claim.

It follows from this lemma that the algorithm *REtoGlu* correctly computes a Glushkov automaton.

Next let us discuss the complexity. By Proposition 1, we can construct the Thompson automaton $M$ and the reversed automaton $M^R$ with $O(\tilde{m})$ states and transitions in $O(\tilde{m})$ time and space. The procedure *ReachState* traverses all states and transitions of $M$ at most twice. When $W$ is word-length of a computer, since it takes $O(\lceil \tilde{m}/w \rceil)$ time to process each state, Step 3 takes $O(\tilde{m}\lceil \tilde{m}/w \rceil)$ time. Step 4 also takes $O(\tilde{m}\lceil \tilde{m}/w \rceil)$ time. Hence the total time becomes $O(\tilde{m}\lceil \tilde{m}/w \rceil)$. The space mainly depends on an array $NEXT[q, a]$, which requires $O(\tilde{m}\lceil \tilde{m}/w \rceil)$ space. Hence the space becomes $O(\tilde{m}\lceil \tilde{m}/w \rceil)$. If $\tilde{m} = O(w)$, then the algorithm runs in $O(\tilde{m})$ time and space. Thus the theorem has proven.

# Appendix A

## Abstracts of Invited Talks
## for
## International Workshop on Combintatoric Algorithms 07

# The Volume of the Birkhoff Polytope

E. Rodney Canfield[1] and Brendan D. McKay[2]

[1] Department of Computer Science,
University of Georgia,
Athens, GA 30602, USA
ercanfie@uga.edu
[2] Department of Computer Science,
Australian National University,
Canberra, ACT 0200, Australia
bdm@cs.anu.edu.au

**Abstract.** For integer $n$, define $\mathcal{B}_n$ to be the polytope of all $n \times n$ non-negative real matrices whose rows and columns each sum to 1. Using a recent asymptotic enumeration of non-negative integer matrices (Canfield and McKay, 2007), we determine the asymptotic volume of $\mathcal{B}_n$ as $n \to \infty$.

A *doubly-stochastic matrix* of order $n$ is an $n \times n$ non-negative real matrix whose rows and columns each sum to 1. As is well-known, the set of all doubly-stochastic matrices of order $n$ form a polytope, the *Birkhoff-von Neumann polytope*, whose vertices are the permutation matrices. We are concerned with the volume of this polytope.

The Birkhoff polytope is an example of a *lattice polytope*, since its vertices lie on the integer lattice. It sits in the space $\mathbb{R}^{n\times n}$, but the constraints on the row and column sums imply that it spans an affine subspace of dimension only $(n-1)^2$.

Two types of volume are customarily defined for lattice polytopes. We can illustrate the difference using the example

$$\mathcal{B}_2 = \left\{ \begin{pmatrix} z & 1-z \\ 1-z & z \end{pmatrix} \;\middle|\; 0 \le z \le 1 \right\} = \left[ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right],$$

where the last notation indicates a closed line-segment in $\mathbb{R}^{2\times 2}$. The length of this line-segment is the *volume* $\mathrm{vol}(B_2) = 2$. We can also consider the lattice induced by $\mathbb{Z}^{2\times 2}$ on the affine span of $\mathcal{B}_2$: this consists of the points $\left(\begin{smallmatrix} z & 1-z \\ 1-z & z \end{smallmatrix}\right)$ for integer $z$. The polytope $\mathcal{B}_2$ consists of a single basic cell of this lattice, so it has *relative volume* $\nu(\mathcal{B}_2) = 1$. In general, $\mathrm{vol}(\mathcal{B}_n)$ is the volume in units of the ordinary $(n-1)^2$-dimensional Lebesgue measure, while $\nu(\mathcal{B}_n)$ is the volume in units of basic cells of the lattice induced by $\mathbb{Z}^{n\times n}$ on the affine span of $\mathcal{B}_n$.

It was proved by Diaconis and Efron [3] that $\mathrm{vol}(\mathcal{B}_n) = n^{n-1}\,\nu(\mathcal{B}_n)$. If we expand $\mathcal{B}_n$ by a large factor, its volume can be approximated by the number of points of the integer lattice which lie inside it. This can be made rigorous:

$$\nu(\mathcal{B}_n) = \lim_{s\to\infty} \frac{M(n,s)}{s^{(n-1)^2}},$$

where $M(n,s)$ is the number of non-negative integer matrices of order $n$ such that all the rows and columns sum to $s$. The latter is known asymptotically for $n \to \infty$ due to recent work of Canfield and McKay [2]. This leads eventually to

$$\mathrm{vol}(\mathcal{B}_n) = \frac{1}{(2\pi)^{n-1/2} n^{(n-1)^2}} \exp\Big(\tfrac{1}{3} + n^2 + O(n^{-1/2+\epsilon})\Big).$$

This asymptotic formula compares very well with the exact values known up to $n = 10$ [1].

We also achieve some generalization to non-square matrices.

## References

1. M. Beck and D. Pixton, The Ehrhart polynomial of the Birkhoff polytope, *Discrete Comput. Geom.*, **30** (2003) 623-637.
2. E. R. Canfield and B. D. McKay, Asymptotic enumeration of contingency tables with constant margins, submitted (2007). Preprint available at `http://www.arxiv.org/abs/math.CO/0703600`.
3. P. Diaconis and B. Efron, Testing for independence in a two-way table: New interpretations of the chi-square statistic, *Ann. Stat.*, **13** (1985) 845–874.

# Orthogonal Drawings of Series-Parallel Graphs*

Takao Nishizeki

Graduate School of Information Sciences, Tohoku University, Aoba-yama 6-6-05, Sendai, 980-8579, Japan.

**Abstract**

In an orthogonal drawing of a planar graph $G$, each vertex is drawn as a point, each edge is drawn as a sequence of alternate horizontal and vertical line segments, and any two edges do not cross except at their common end. A bend is a point where an edge changes its direction. A drawing of $G$ is called an optimal orthogonal drawing if the number of bends is minimum among all orthogonal drawings of $G$.

In this talk we deal with the class of series-parallel (multi)graphs of degrees at most 3, and give a simple linear algorithm to find an optimal orthogonal drawing in the variable embedding setting. The graph $G$ in Fig. 1 is series-parallel, and has various plane embeddings; two of them are illustrated in Figs. 1(b) and (c); there is no plane embedding having an orthogonal drawing with no bend; however, the embedding in Fig. 1(b) has an orthogonal drawing with one bend as illustrated in Fig. 1(a) and hence the drawing is optimal; the embedding in Fig. 1(c) needs three bends as illustrated in Fig. 1(d); given $G$, our algorithm finds an optimal drawing in Fig. 1(a). Our algorithm works well even if $G$ has multiple edges or is not biconnected, and is much simpler and faster than the known algorithms for biconnected series-parallel simple graphs; we use neither the min-cost flow technique nor the SPQ*R tree, but uses some structural features of series-parallel graphs. We furthermore obtain a best possible upper bound on the minimum number of bends.

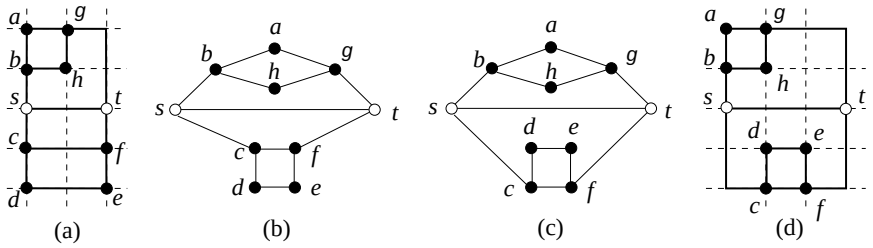



**Fig. 1.** (a) An optimal orthogonal drawing with one bend, (b), (c) two embeddings of the same planar graph, and (d) an orthogonal drawing with three bends.

* This work is supported by JSPS grants.

# Time-Constrained Graph Searching

Brian Alspach

School of Mathematical and Physical Sciences,
The University of Newcastle

**Abstract**

Searching graphs or digraphs for an intruder has been studied since the 1970s. There has been a boost in activity in the area because of mobile software agents. Most of the research has concentrated on the minimum number of searchers required to capture an intruder for a variety of searching models. There are applications for which the cost of searchers is negligible and minimizing the time required to capture an intruder is of interest. The latter is the subject of this talk.

# Computing the $k$ Most Representative Skyline Points

Xuemin Lin

School of Computer Science & Engineering
University of New South Wales
Sydney, NSW 2052, Australia
lxue@cse.unsw.edu.au

**Abstract**

Skyline computation has many applications including multi-criteria decision making. In this talk, we study the problem of selecting $k$ skyline points so that the number of points, which are dominated by at least one of these $k$ skyline points, is maximized. We first present an efficient dynamic programming based exact algorithm in a $2d$-space. Then, we show that the problem is NP-hard when the dimensionality is 3 or more and it can be approximately solved by a polynomial time algorithm with the guaranteed approximation ratio $1 - \frac{1}{e}$. To speed-up the computation, an efficient, scalable, index-based randomized algorithm is developed by applying the FM probabilistic counting technique. A comprehensive performance evaluation demonstrates that our randomized technique is very efficient, highly accurate, and scalable.

# Distance constrained graph labeling: From frequency assignment to graph homomorphisms

Jan Kratochvil

Charles University, Prague

**Abstract**

The notion of distance constrained graph labelings stems from a practical problem of assigning frequencies to transmitters with the aim of avoiding unwanted interference. Apart from having this applied motivation, the problem is rather interesting from theoretical point of view as well. We will survey recent results and open problems, and in particular explore the connections to the algebraically motivated notion of graph homomorphisms.

# The use of decomposition in the study of even-hole-free graphs

Kristina Vušković

School of Computing, University of Leeds, UK

**Abstract**

We consider finite and simple graphs. We say that a graph $G$ *contains* a graph $F$, if $F$ is isomorphic to an induced subgraph of $G$. A graph $G$ is *$F$-free* if it does not contain $F$. Let $\mathcal{F}$ be a (possibly infinite) family of graphs. A graph $G$ is *$\mathcal{F}$-free* if it is $F$-free, for every $F \in \mathcal{F}$.

Many interesting classes of graphs can be characterized as being $\mathcal{F}$-free for some family $\mathcal{F}$. Most famous such example is the class of perfect graphs. A graph $G$ is *perfect* if for every induced subgraph $H$ of $G$, $\chi(H) = \omega(H)$, where $\chi(H)$ denotes the chromatic number of $H$ and $\omega(H)$ denotes the size of a largest clique in $H$. The famous Strong Perfect Graph Theorem states that a graph is perfect if and only if it does not contain an odd hole nor an odd antihole (where a *hole* is a chordless cycle of length at least four).

In the last 15 years a number of other classes of graphs defined by excluding a family of induced subgraphs have been studied, perhaps originally motivated by the study of perfect graphs. The kinds of questions this line of research was focused on were whether excluding induced subgraphs affects the global structure of the particular class in a way that can be exploited for putting bounds on parameters such as $\chi$ and $\omega$, constructing optimization algorithms (problems such as finding the size of a largest clique or a minimum coloring), recognition algorithms and explicit construction of all graphs belonging to the particular class. A number of these questions were answered by obtaining a structural characterization of a class through their decomposition (as was the case with the proof of the Strong Perfect Graph Theorem).

In this talk we survey some of the most reacent uses of the decomposition theory in the study of classes of even-hole-free graphs. Even-hole-free graphs are related to $\beta$-perfect graphs in a similar way in which odd-hole-free graphs are related to perfect graphs. $\beta$-Perfect graphs are a particular class of graphs that can be polynomialy colored, by coloring greedily on a particular, easily constructable, ordering of vertices.

# Haplotype Inference Constrained by Plausible Haplotype Data

Gad M. Landau*

Department of Computer Science,
University of Haifa, Israel

**Abstract**

The haplotype inference problem (HIP) asks to find a set of haplotypes which resolve a given set of genotypes. This problem is of enormous importance in many practical fields, such as the investigation of diseases, or other types of genetic mutations. In order to find the haplotypes that are as close as possible to the real set of haplotypes that comprise the genotypes, two models have been suggested which by now have become widely accepted: The perfect phylogeny model and the pure parsimony model. All known algorithms up till now for the above problem may find haplotypes that are not necessarily plausible, i.e. very rare haplotypes or haplotypes that were never observed in the population. In order to overcome this disadvantage we study in this paper, for the first time, a new constrained version of HIP under the above mentioned models. In this new version, a pool of plausible haplotypes H is given together with the set of genotypes G, and the goal is to find a subset of H that resolves G.

* Joint work with Tzvika Hartman, Danny Hermelin and Liat Leventhal.

# Full-Text Indexing in a Changing World

Moshe Lewenstein

Department of Computer Science, Bar Ilan University, Israel

**Abstract**

Full-Text indices have been around for over 30 years now. Nevertheless, with the advent of the web, growing databases and a growing collection of applications dictates new needs from full-text indices. There is a need for them to be fast, space efficient, and to handle new issues such as errors in the text.

In this talk several of these problems will be presented and some of the new approaches to solutions will be presented.

# Appendix B

Open Problems
Presented
in
International Workshop on Combintatoric Algorithms 07

## 1 Open Problems in Dynamic Map Labeling

**Brief description** The problem of map labeling is well-studied in GIS and computational geometry. The standard setting for such problems is the traditional (static) map. We are interested in *dynamic maps*, the sort we are familiar with on the web. The user can zoom and pan to view different portions of some large map that does not fit in the screen. One idea for solving dynamic labeling is to reduce them directly to static labeling. This is essentially the approach of Petzold [3, 4, 6, 5]. The problem with this approach is the consistency issue as pointed out by [1]. They listed a set of consistency desiderata, formalized the dynamic labeling problem. In particular, they introduce a natural class of dynamic placements called *point-invariant placements* which as a nice interpretation as a geometric cone (the third dimension is scale space). They also provided a practical solution satisfying the desiderata. This solution is implemented in the `G-Vis System` [2], a dynamic map for continental USA which is accessible by any browser.

Since this problem is quite new, most issues are open. But here are two specific problems taken from [1]:

1. The solution in [1] assumes each label has a single placement. It should not be hard to generalize this to having a (small) finite set of discrete possibilities per label. More challenging is to accomodate labels with a continuous set of possibilities.

2. The complexity of the problem of *active range optimization* (ARO) in [1] is open. Intuitively, we can view a dynamic label as a cone (with a rectangular base) in 3-D. The problem is to truncate these cones so that they are non-overlapping subject to some optimization criteria. Recently, Sheung-Hung Poon (private communication) noted that the 1-D optimization version of the simplified ARO is polynomial-time, but the 2-D version remains open.

**More Detail** Here is the standard (simplified) setting: we are given a *map* $M$, viewed as rectangular region of the plane, and containing a set of *(map) features*. There are three kinds of features: *points*, *lines*, *regions*. Note that a line feature is really a polygonal line, and a region feature is really a connected polygonal region.

Each of these features has a label (viewed as a floating rectangle) which may be placed on the map satisfying suitable constraints. E.g., A point label must be within a certain radius of the point (but not cover it). A line label must be placed parallel to one on the line segments of the polyline, either above or below the line segment within some distance. A region label must intersect the region.

The computational problem is to (1) *select* a subset of the labels, and (2) *place* each selected label in a suitable position satisfying the above constraints, *such that no two labels overlap*. The optimization criterion is usually to maximize the number of selected labels. Sometimes, we assume the selection subproblem (1) has been solved, and we only want to do the placement subproblem (2).

In the dynamic setting, we assume that only a portion of the map $M$ is visible. That is we can choose a "viewing window" or *portal* $P$. Intuitively, we can zoom and pan this portal to view different parts of the map at different

scales. For simplicity, assume the portal shape is fixed (say, it is a square). A portal is parametrized by three numbers: $(s, x, y)$ where $s > 0$ is the *zoom scale*, and $(x, y)$ is the *portal position*. This determines a square $P(s, x, y)$ centered at position $(x, y)$ on $M$. The sides of $P(s, x, y)$ has length $s$. We imagine this portion of $M$ is then transformed to a fixed size window $W$ on the computer screen. Note that $W$ has the same shape as the portal, and a label displayed on $W$ has a fixed size (i.e., it does not depend on the scale $s$). This means that when we zoom in, labels grow in size on the screen, and two non-overlapping labels may begin to overlap.

Petzold et al. [3, 4, 6, 5] used a pre-processing solution that reduces dynamic to static map labeling. The idea is this: given $(s, x, y)$, we first retrieve some superset $L$ of the labels that are potentially visible in the portal $P(s, x, y)$. Then we run a static labeling algorithm on this set $L$. The problem with this solution is that there is no "consistency" for different choices of $s, x, y$. A set of consistency requirements is specified in [1], who also provided a solution that satisfies them.

## References

1. K. Been, E. Daiches, and C. Yap, Dynamic map labelling, *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773-780,2006. Proc. 12th Symp. on Information Visualization (InfoVis06), Baltimore, Maryland, October 2006.
2. G-Vis Dynamic Labeling Demo, 2002. URL http://sage.mc.yu.edu/gvis/.
3. I. Petzold. Textplazierung in dynamisch erzeugten Karten. Diploma-thesis, Institute for Computer Science, Bonn, 1996.
4. I. Petzold, *Beschrifting von Bildschirmkarten in Echtzeit - Konzept und Struktur*. Ph.D. thesis, Institute of Cartographt and Geoinformation, University of Bonn,2003.
5. I. Petzold, G. Gröger, and L. Plümer, Fast screen map labeling - data-structures and algorithms, In *Proc. 21th International Cartographic Conferences (ICC'03)*, pages 288-298, 2003.
6. I.Petzold, L. Plümer, and M. Heber, Label placement for dynamically generated screen maps. In *Proc. 19th International Cartographic Conferences (ICC'99)*, pages 893-903, 1999.

Chee Yap,
New York University,
USA.

## 2 Graphs with no equal length cycles

Let $f(n)$ be the maximum number of edges in a graph on $n$ vertices in which no two cycles have the same length. In 1975, Erdös raised the problem of determining $f(n)$ (see [1]). Y. Shi [4] proved that

$$f(n) \geq n + \lfloor(\sqrt{8n-23}+1)/2\rfloor$$

for $n \geq 3$. Boros et al. proved that

$$f(n) \leq n + 1.98\sqrt{n}(1+o(1)).$$

Chunhui Lai [3] proved that

$$\liminf_{n\to\infty} \frac{f(n)-n}{\sqrt{n}} \geq \sqrt{2.4}.$$

Combining this with the upper bound in [2], we get

$$1.98 \geq \limsup_{n\to\infty} \frac{f(n)-n}{\sqrt{n}} \geq \liminf_{n\to\infty} \frac{f(n)-n}{\sqrt{n}} \geq \sqrt{2.4}.$$

We make the following conjecture:

**Conjecture.**

$$\lim_{n\to\infty} \frac{f(n)-n}{\sqrt{n}} = \sqrt{2.4}.$$

## References

1. J.A. Bondy and U.S.R. Murty, Graph Theory with Applications (Macmillan, New York, 1976)], p.247, Problem 11.
2. E. Boros, Y. Caro, Z. Füredi and R. Yuster, Covering non-uniform hypergraphs, Journal of Combinatorial Theory, Series B 82(2001), 270-284.
3. Chunhui Lai, Graphs without repeated cycle lengths, Australasian Journal of Combinatorics 27(2003), 101-105.
4. Y. Shi, On maximum cycle-distributed graphs, Discrete Math. 71(1988) 57-71.

ChunHui Lai
Zhangzhou Teachers College
P.R. of China.

## 3 Entropy-compressed suffix trees

Given a text $T[1,n]$ over an alphabet of size $s$, a plain representation requires $n \log s$ bits (log is to base 2). A $k$-th order compressor can reduce its size to $nHk$ bits, where $Hk$ is the empirical $k$-th order entropy of $T$ [1]. Classical text indexes such as suffix trees and suffix arrays [2] require $O(n \log n)$ bits. This waste of space is troublesome when indexing large texts that could fit in main memory but whose indexes (sometimes 20 times larger than the text!) cannot. Thus there is a strong interest in reduced-space representations that retain reasonable efficiency.

Many recent developments [3] achieve suffix array functionality using $nHk + o(n \log s)$ bits, for any $k \leq a \log_s n$ and any constant $0 < a < 1$. This space also contains the text, in the sense that the structure is capable of reproducing any text substring. By "suffix array functionality" I mean counting the number of occurrences of any pattern, and enumerating its text positions. The former can be done, say, in $O(m \log s)$ time ($m$ being the pattern length), and the latter in $O(\text{polylog(n)})$ time per reported occurrence.

Suffix tree functionality is more ambitious. It permits navigating the (concrete or virtual) suffix tree with operations like parent, child-labeled-$a$, first-child, next-sibling, suffix-link (leading from node representing $ax$ to node representing $x$, $a$ being a symbol and $x$ a string), queries like subtree-size, first-leaf, last-leaf, and optionally other more ambitious ones like level-ancestor, lowest-common-ancestor, etc.

There have been recent achievements on succinct suffix trees with full functionality, most notably Sadakane's [4]. Yet all of them still require $O(n)$ extra bits of space on top of the entropy. In principle, $nHk + o(...)$ bits should be sufficient (as at worst one can uncompress the text, build the suffix tree, and do the operation!), but no one has devised a way to operate efficiently on a suffix tree structure that is fully entropy-compressed, without any extra linear space. I believe this should be possible.

## References

1. G. Manzini. An analysis of the Burrows-Wheeler transform. Journal of the ACM 48(3):407-430, 2001.
2. D. Gusfield. Algorithms on strings, trees, and sequences: Computer Science and Computational Biology. Cambridge University Press, 1997.
3. G. Navarro and V. Makinen. Compressed full-text indexes. ACM Computing Surveys 39(1):article 2, 2007.
4. K. Sadakane. Compressed suffix trees with full functionality. Theory of Computing Systems, to appear. Preliminary version available at http://tcslab.csce.kyushu-u.ac.jp/ sada/papers/cst.ps

Gonzalo Navarro,
University of Chile,
Chile.

## 4 Indexed approximate string matching

This is the problem of finding all the approximate occurrences, in a text $T[1, n]$, of a pattern $P[1, m]$, both over an alphabet of size $s$. By "approximate occurrence" I mean that at most $k$ "edit operations" need to be done on any text substring to make it match the pattern. The most popular edit operations are insertions, deletions, and substitution of characters [1]. In particular I refer to the indexed variant of the problem [2], where one builds an index on $T$ to speed up the searches for arbitrary patterns.

Although there has been progress on this problem, one still finds that either the index is of exponential size (in $k$ or $m$ or $s$), or the search takes exponential time. See e.g. [3, 4]. I believe this is a fundamental space/time barrier, but as far as I know this has not been proved.

## References

1. G. Navarro. A guided tour to approximate string matching. ACM Computing Surveys 33(1):31-88, 2001.
2. G. Navarro, R. Baeza-Yates, E. Sutinen, J. Tarhio. Indexing methods for approximate string matching. IEEE Data Engineering Bulletin 24(4):19-27, 2001.
3. R. Cole, L. Gottlieb, M. Lewenstein. Dictionary matching and indexing with errors and don't cares. Proc. STOC'04, pp 91-100, 2004.
4. M. Maas, J. Nowak. Text indexing with errors. Proc. CPM'05, pp. 21-32, 2005.

Gonzalo Navarro,
University of Chile,
Chile.

## 5 Does a polynomial maximising algorithm imply a polynomial minimising algorithm?

Optimisation problems come in two flavours: maximisation and minimisation. Among these, some are polynomially solvable. Polynomially solvable optimisation problems can be said to form a class $P_{opt}$. Thus $P_{opt} = P_{max} \cup P_{min}$, where $P_{max}$ ($P_{min}$) is the class of polynomially solvable maximisation (respectively, minimisation) problems. Of course, $P_{max} \cap P_{min} = \emptyset$.

For a problem $P \in P_{opt}$, let $P'$ be the dual of $P$.

(a) If $P \in P_{max}$, is $P' \in P_{min}$ always? (In other words, is $P'$ also always polynomially solvable?)
(b) Is the opposite always true — if $P \in P_{min}$, is $P' \in P_{max}$ always?
(c) Are both (a) and (b) true always? In other words, is $P_{opt}$ closed under duality ?

Are there any results known on these problems ?

Same "problem" as above, but let the optimisation problems be *polynomially bounded* — that is, the optimal solution value is bounded by a polynomial in the size of the problem instance.

Prabhu Manyem,
University of Ballarat,
Australia.

## 6 Certificate Dispersal Problems

We consider a mobile network, where each node $u$ has a private key $pri.u$ and a public key $pub.u$. Users themselves create their public and private keys. In this network, in order for a node $u$ to send a message $m$ to a node $v$ securely, $u$ needs to know $pub.u$ to encrypt the message with it, denoted by $pub.v < m >$. If a node $u$ knows the public key $pub.v$ of another node $v$ in the network, then the node $u$ can issue a certificate from $u$ to $v$ that identifies $pub.v$. A certificate from a node $u$ to a node $v$ is of the following form: $pri.u < u, v, pub.v >$. The certificate is encrypted by using $pri.u$ and it contains three items: (i) the identity of the certificate issuer $u$, (ii) the identity of the certificate subject $v$, and (iii) the public key of the certificate subject $pub.v$.

Any node who knows $pub.u$ can use it to decrypt the certificate from $u$ to $v$ for obtaining $pub.v$. When a node $u$ wants to obtain the public key of another node $v$, $u$ acquires a sequence of certificates $pri.u < u, v_0, pub.v_0 >, pri.v_0 < v_0, v_1, pub.v_1 >, \ldots pri.v_\ell < v_\ell, v, pub.v >$ which are stored in either $u$ or $v$.

All certificates issued by nodes in a network can be represented by a directed graph, called *a certificate graph*, denoted by $G = (V, E)$. We define a *dispersal* $D$ of a directed graph $G = (V, E)$ as a family of sets of edges indexed by $V$, where $D = \{D_v \subseteq E | v \in V\}$. We define a request for a certificate graph $G$ as a reachable ordered pair of nodes in $G$, denoted by $(u, v)$. A set of requests, denoted by $R$, is called *full* if all reachable pairs of nodes in $G$ are contained in $R$. We call a dispersal $D$ of a directed graph $G = (V, E)$ *satisfies* a set of requests $R$, if for any request $(u, v)$ in $R$, there exists a path from $u$ to $v$ in $D_u \cup D_v$, where $D_u$ and $D_v$ are dispersals of $u$ and $v$. Let $D$ be a dispersal of $G$ satisfying $R$. The *cost* of dispersal $D$, denoted by $c.D$, is the sum of cardinalities of each dispersal in $D$. A dispersal $D$ of $G$ satisfying $R$ is optimal if and only if for any other dispersal $D'$ satisfies $R$, $c.D \leq c.D'$.

The MINIMUM CERTIFICATE DISPERSAL PROBLEM(MCD) is defined as follows:

INPUT: A directed graph $G = (V, E)$ and a set of requests $R$
OUTPUT: A dispersal $D$ of $G$ satisfying $R$ with the minimum cost.

It has been shown that MCD is NP-complete, even if the input graph is restricted to a strongly connected one. Also a polynomial-time 2-approximation algorithm can be constructed for strongly connected graphs. Moreover, it can be shown that this algorithm outputs optimal dispersals for complete graphs, trees, rings and Cartesian product of graphs.

In general, MCD is NP-complete for strongly connected graphs. A remaining question is whether MCD remains NP-complete for bidirectional graphs(or undirected graphs) and/or full requests or not.

Can the approximation ratio of MCD algorithms can be improved? That is, can a polynomial-time $\alpha$-approximation algorithm such that $\alpha < 2$ be constructed for any directed graph?

The 2-approximation algorithm shown in [1] becomes optimal one for complete graphs, trees, rings hypercubes(more generally, Cartesian product of graphs). For some useful graph classes, such as chordal graphs or interval graphs, can we construct an optimal MCD algorithm?

## References

1. H. Zheng, S. Omura, K. Wada: A 2-approximation algorithm for minimum certificate dispersal problems, Proceedings of the 16th Australasian Workshop on Combinational Algorithms, 384-394 (2005).

Koichi Wada,
Nagoya Institute of Technology,
JAPAN.

## 7 The Maximum Number of Runs in a String

Given a nonempty string $\boldsymbol{u}$ and an integer $e \geq 2$, we call $\boldsymbol{u}^e$ a *repetition*; if $\boldsymbol{u}$ itself is not a repetition, then $\boldsymbol{u}^e$ is a *proper repetition*. Given a string $\boldsymbol{x}$, a *repetition in* $\boldsymbol{x}$ is a substring

$$\boldsymbol{x}[i..i+e|\boldsymbol{u}|1] = \boldsymbol{u}^e,$$

where $\boldsymbol{u}^e$ is a proper repetition and neither $\boldsymbol{x}\big[i+e|\boldsymbol{u}|..i+(e+1)|\boldsymbol{u}|-1\big]$ nor $\boldsymbol{x}[i-|\boldsymbol{u}|..i-1]$ equals $\boldsymbol{u}$. We say the repetition has *period* $|\boldsymbol{u}|$ and *exponent* $e$; it can be specified by the integer triple $(i, |\boldsymbol{u}|, e)$. It is well known [2] that the maximum number of repetitions in a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is $\Theta(n \log n)$, and that the number of repetitions in $\boldsymbol{x}$ can be computed in $\Theta(n \log n)$ time [2, 1, 10].

A string $\boldsymbol{u}$ is a *run* if and only if it is periodic of (minimum) period $p \leq |\boldsymbol{u}|/2$. Thus $\boldsymbol{x} = abaabaabaabaab = (aba)^4ab$ is a run of period $|aba| = 3$. A substring $\boldsymbol{u} = \boldsymbol{x}[i..j]$ of $\boldsymbol{x}$ is a *run* or *maximal periodicity in* $\boldsymbol{x}$ if and only if it is a run of period $p$ and neither $\boldsymbol{x}[i-1..j]$ nor $\boldsymbol{x}[i..j+1]$ is a run of period $p$. The run $\boldsymbol{u}$ has *exponent* $e = \lfloor |\boldsymbol{u}|/p \rfloor$ and possibly empty *tail* $\boldsymbol{t} = \boldsymbol{x}[i+ep..j]$ (proper prefix of $\boldsymbol{x}[i..i+p-1]$). Thus

$$\begin{array}{r} 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14 \\ \boldsymbol{x} = b\ a\ a\ a\ b\ a\ a\ b\ a\ a\ \ b\ \ a\ \ b\ \ a \end{array}$$

contains a run $\boldsymbol{x}[3..12]$ of period $p = 3$ and exponent $e = 3$ with tail $\boldsymbol{t} = a$ of length $t = |\boldsymbol{t}| = 1$. It can be specified by a 4-tuple $(i, p, e, t) = (3, 3, 3, 1)$. and it includes the repetitions $(aab)^3$, $(aba)^3$ and $(baa)^2$ of period $p = 3$. In general it is easy to see that for $e = 2$ a run *encodes* $t+1$ repetitions; for $e > 2$, $p$ repetitions. Clearly, computing all the runs in $\boldsymbol{x}$ specifies all the repetitions in $\boldsymbol{x}$. The idea of a run was introduced in [9].

Let $r_{\boldsymbol{x}}$ denote the number of runs that actually occur in a given string $\boldsymbol{x}$, and let $\rho(n)$ denote the maximum number of runs that can possibly occur in any string $\boldsymbol{x}$ of given length $n$. A string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ such that $r_{\boldsymbol{x}} = \rho(n)$ is said to be *run-maximal*.

In [7, 8] it was shown that there exist universal positive constants $k_1$ and $k_2$ such that

$$\rho(n)/n < k_1 k_2 \log_2 n/\sqrt{n},$$

but the proof was nonconstructive and provided no way of estimating the magnitude of $k_1$ and $k_2$. In [7], using a brute force algorithm, a table of $\rho(n)$ was computed for $n = 5, 6, \ldots, 31$, giving also for each $n$ an example of a run-maximal string; for every $n$ in this range, $\rho(n)/n < 1$ and $\rho(n) \leq \rho(n-1) + 2$. In [5] an infinite sequence $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots.\}$ of strings was described, with $|\boldsymbol{x_{i+1}}| > |\boldsymbol{x_i}|$ for every $i \geq 1$, such that

$$\lim_{i \to \infty} r_{\boldsymbol{x_i}}/|\boldsymbol{x_i}| = \frac{3}{2\phi},$$

where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden mean. Moreover, it was conjectured that in fact

$$\lim_{n\to\infty} \rho(n)/n = \frac{3}{2\phi}. \tag{1}$$

Recently a different and simpler construction was found [6] to yield another infinite sequence $X$ of strings for which the ratio $r_{\boldsymbol{x_i}}/|\boldsymbol{x_i}|$ approached the same limit; in addition, it was shown that for every $\epsilon > 0$ and for every sufficiently large $n = n(\epsilon)$, $\frac{3}{2\phi}\epsilon$ provides an asymptotic lower bound on $\rho(n)/n$.

In 2006 considerable progress was made on the estimation of an upper bound on $\rho(n)/n$:

* $\rho(n)/n \leq 5.0$ [12];
* $\rho(n)/n \leq 3.48$ [11];
* $\rho(n)/n \leq 3.44$ [13];
* $\rho(n)/n \leq 1.6$ [3].

Thus the problem may be stated as follows:

**Is conjecture (1) true?**
**If not, then characterize the function $\rho(n)/n$.**

Help may be found in recent work studying the limitations imposed on the existence and length of runs in neighbourhoods of positions where two runs are known to exist [4, 14].

## References


1. Alberto Apostolico & Franco P. Preparata, **Optimal off-line detection of repetitions in a string**, *Theoret. Comput. Sci. 22* (1983) 297–315.
2. Maxime Crochemore, **An optimal algorithm for computing the repetitions in a word**, *Inform. Process. Lett. 12–5* (1981) 244–250.
3. Maxime Crochemore & Lucian Ilie, **Maximal repetitions in strings**, submitted for publication (2006).
4. Kangmin Fan, Simon J. Puglisi, W. F. Smyth & Andrew Turpin, **A new periodicity lemma**, *SIAM J. Discrete Math. 20–3* (2006) 656–668.
5. Frantisek Franek, R. J. Simpson & W. F. Smyth, **The maximum number of runs in a string**, *Proc.* $14^{\text{th}}$ *Australasian Workshop on Combinatorial Algs.*, M. Miller & K. Park (eds.) (2003) 26–35.
6. Frantisek Franek & Qian Yang, **An asymptotic lower bound for the maximum-number-of-runs function**, *Proc. Prague Stringology Conference '06*, Jan Holub & Jan Žd'árek (eds.) (2006) 3–8.
7. Roman Kolpakov & Gregory Kucherov, *Maximal Repetitions in Words or How to Find all Squares in Linear Time*, Rapport LORIA 98-R-227, Laboratoire Lorrain de Recherche en Informatique et ses Applications (1998) 22 pp.
8. Roman Kolpakov & Gregory Kucherov, **On maximal repetitions in words**, *J. Discrete Algs. 1* (2000) 159–186.
9. Michael G. Main, **Detecting leftmost maximal periodicities**, *Discrete Applied Maths. 25* (1989) 145–153.

10. Michael G. Main & Richard J. Lorentz, **An $O(n \log n)$ algorithm for finding all repetitions in a string**, *J. Algs. 5* (1984) 422–432.
11. Simon J. Puglisi, R. J. Simpson & W. F. Smyth, **How many runs can a string contain?**, submitted for publication (2006).
12. Wojciech Rytter, **The number of runs in a string: improved analysis of the linear upper bound**, *Proc.* 23rd *Symp. Theoretical Aspects of Computer Science*, B. Durand & W. Thomas (eds.), LNCS 2884, Springer-Verlag (2006) 184–195.
13. Wojciech Rytter, **The number of runs in a string**, submitted for publication (2006).
14. R. J. Simpson, **Intersecting periodic words**, to appear, Theoret. Comput. Sci.

Bill Smyth
McMaster University, Canada
and Curtin University, Australia

# Author Index