Theoretical Computer Science 710 (2018) 126–138

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the string consensus problem and the Manhattan sequence consensus problem

Tomasz Kociumaka [a], Jakub W. Pachocki [b,1], Jakub Radoszewski [a,*], Wojciech Rytter [a], Tomasz Waleń [a]

[a] Institute of Informatics, University of Warsaw, Poland
[b] OpenAI, US

ARTICLE INFO

Article history:
Received 15 June 2016
Received in revised form 15 March 2017
Accepted 23 March 2017
Available online 25 March 2017

Keywords:
Sequence consensus problem
Center string problem
Closest string problem

ABSTRACT

We study the Manhattan Sequence Consensus problem (MSC problem) in which we are given k integer sequences, each of length ℓ, and we are to find an integer sequence $\mathbf{x}$ of length ℓ (called a *consensus sequence*) such that the maximum Manhattan distance of $\mathbf{x}$ from each of the input sequences is minimized. A related problem, with Hamming distance instead of Manhattan distance, is called Hamming String Consensus (HSC), also known under the names of string center problem or closest string problem. For binary sequences Manhattan distance coincides with Hamming distance, hence in this case HSC is a special case of MSC. We design a practically efficient $\mathcal{O}(\ell)$-time algorithm solving MSC for $k \leq 5$ sequences. It improves upon the quadratic algorithm by Amir et al. (2012) [1] for HSC for $k = 5$ binary strings. Similarly as in the algorithm of Amir et al., we use a column-based framework. We replace the implied general integer linear programming by its easy special cases due to combinatorial properties of MSC for $k \leq 5$. Practicality of our algorithms has been verified experimentally. We also show that for a general parameter k any instance can be reduced in linear time to a kernel of size $k!$, so the problem is fixed-parameter tractable. Nevertheless, for $k \geq 4$ this is still too large for any naive solution to be feasible in practice. This is a full version of an article published at SPIRE 2014 [15].

1. Introduction

In the sequence consensus problems, given a set of sequences of length ℓ we are searching for a new sequence of length ℓ which minimizes the maximum distance to all the given sequences in some particular metric. Finding the consensus sequence is a tool for many clustering algorithms and as such has applications in unsupervised learning, classification, databases, spatial range searching, data mining etc. [4]. It is also one of popular methods for detecting data commonalities of many strings (see [1]) and has a considerable number of applications in coding theory [6,9], data compression [12], and bioinformatics [13,17]. The consensus problem has previously been studied mainly in $\mathbb{R}^\ell$ space with the Euclidean distance and in Σ^ℓ (that is, the space of sequences over a finite alphabet Σ) with the Hamming distance. Other metrics

* Corresponding author.
E-mail addresses: kociumaka@mimuw.edu.pl (T. Kociumaka), merettm@gmail.com (J.W. Pachocki), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), walen@mimuw.edu.pl (T. Waleń).

[1] This work was done while the author was a student at the University of Warsaw.

http://dx.doi.org/10.1016/j.tcs.2017.03.022
0304-3975/

were considered in [2]. We study the sequence consensus problem for Manhattan metric (ℓ_1 norm) in correlation with the Hamming-metric variant of the problem.

The *Euclidean* variant of the sequence consensus problem is also known as the bounding sphere, enclosing sphere, or enclosing ball problem. It was initially introduced in 2 dimensions (i.e., the smallest circle problem) by Sylvester in 1857 [24]. For an arbitrary number of dimensions, a number of approximation algorithms [4,16,23] and practical exact algorithms [8, 11] for this problem have been proposed.

The *Hamming distance* variant of the sequence consensus problem is known under the names of string consensus, center string or closest string problem. The problem is known to be NP-complete even for binary alphabet [9]. The algorithmic study of Hamming string consensus (HSC) problem started in 1999 with the first approximation algorithm with the approximation ratio $\frac{4}{3} + o(1)$ [17]. Afterwards several polynomial-time approximation schemes (PTASes) with different running times were presented: $(1+\epsilon)$-approximation of the result was computed in $\ell k^{\mathcal{O}(\frac{1}{\epsilon^5})}$ time in [19], in $\ell k^{\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon})}$ time in [3], and in $\ell k^{\mathcal{O}(\frac{1}{\epsilon^2})}$ time in [21,22]. A number of exact algorithms have also been presented; in many of these one considers a decision version of the problem, in which we are to check if there is a solution to HSC problem with distance at most d to the input sequences. Thus FPT algorithms with time complexities $\mathcal{O}(k\ell + kd^{d+1})$ and $\mathcal{O}(k\ell + kd(16|\Sigma|)^d)$ were presented in [13] and [21], respectively.

An FPT algorithm parameterized only by k was given in [13]. It uses Lenstra's algorithm [18] for a solution of an integer linear program of size exponential in k (which requires $\mathcal{O}(k!^{4.5k!}\ell)$ operations on integers of magnitude $\mathcal{O}(k!^{2k!}\ell)$; see [1]) and due to extremely large constants is not feasible for $k \geq 4$. This opened a line of research with efficient algorithms for small constant k. Thus a specialized linear-time algorithm for $k = 3$ was presented in [13], a linear-time algorithm for $k = 4$ and binary alphabet was given in [5], and recently an $\mathcal{O}(\ell^2)$-time algorithm for $k = 5$ and also binary alphabet was developed in [1].

For two integer sequences $\mathbf{x} = (x_1, \ldots, x_\ell)$ and $\mathbf{y} = (y_1, \ldots, y_\ell)$, the *Manhattan distance* (also known as rectilinear or taxicab distance) between $\mathbf{x}$ and $\mathbf{y}$ is defined as:

$$dist(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{\ell} |x_j - y_j|.$$

The Manhattan version of the consensus problem is formally defined as follows:

> Manhattan Sequence Consensus (MSC) problem
> **Input:** A collection $\mathcal{A}$ of k integer sequences $\mathbf{a}_i$, each of length ℓ
> **Output:** An integer sequence $\mathbf{x}$ of length ℓ that minimizes $dist(\mathbf{x}, \mathcal{A}) = \max\{dist(\mathbf{x}, \mathbf{a}_i) : 1 \leq i \leq k\}$ and $\mathrm{OPT}(\mathcal{A})$ equal to $dist(\mathbf{x}, \mathbf{a}_i)$ for this sequence $\mathbf{x}$.

We assume that integers $a_{i,j}$ satisfy $|a_{i,j}| \leq M$, and all ℓ, k, M fit in a machine word, so that arithmetic operations on integers of magnitude $\mathcal{O}(\ell M)$ take constant time.

Example 1. Let $\mathcal{A} = ((120, 0, 80), (20, 40, 130), (0, 100, 0))$. Then $\mathrm{OPT}(\mathcal{A}) = 150$ and a consensus sequence is $\mathbf{x} = (30, 40, 60)$; see also Fig. 1.

For two strings $\mathbf{x}$ and $\mathbf{y}$ (i.e., two sequences of letters over some alphabet) of the same length, we define their Hamming distance $dist_H(\mathbf{x}, \mathbf{y})$ as the number of positions where the two strings differ. We define the Hamming version of the consensus problem.

> Hamming String Consensus (HSC) problem
> **Input:** A collection $\mathcal{A}$ of k strings $\mathbf{a}_i$, each of length ℓ
> **Output:** A string $\mathbf{x}$ of length ℓ that minimizes $dist_H(\mathbf{x}, \mathcal{A}) = \max\{dist_H(\mathbf{x}, \mathbf{a}_i) : 1 \leq i \leq k\}$ and $\mathrm{OPT}(\mathcal{A})$ equal to $dist_H(\mathbf{x}, \mathbf{a}_i)$ for this string $\mathbf{x}$.

1.1. Our results

Our main results are as follows.

- We show that Manhattan Sequence Consensus problem has a kernel composed of k sequences of length $\ell \leq k!$ and is fixed-parameter linear with the parameter k.
- We present a practical linear-time algorithm for the Manhattan Sequence Consensus problem for $k = 5$ (which obviously can be used for any $k \leq 5$).

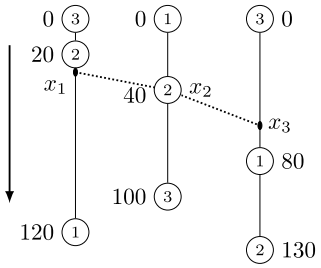

Fig. 1. Illustration of Example 2; $\pi_1 = (3, 2, 1)$, $\pi_2 = (1, 2, 3)$, $\pi_3 = (3, 1, 2)$.

Note that binary HSC problem is a special case of the MSC problem. Hence, the latter problem is NP-complete. Moreover, the efficient linear-time algorithm presented here for the MSC problem for $k = 5$ yields an equally efficient linear-time algorithm for the binary HSC problem and thus improves the result of [1].

Our approach is based on a reduction of the MSC problem to instances of integer linear programming (ILP). For general constant k we obtain a constant, though very large, number of instances with a constant number of variables that we solve using Lenstra's algorithm [18] which works in constant time (the constant coefficient of this algorithm is also very large). This idea is similar to the one used in the FPT algorithm for the HSC problem [13]. However, for MSC it requires an additional combinatorial observation. For $k \leq 5$ we obtain a more efficient reduction of MSC to at most 20 instances of very special ILP which we solve efficiently without applying a general ILP solver.

This is a full version of an article published at SPIRE 2014 [15].

Organization of the paper. In Section 2 we show the first steps of the reduction of MSC to ILP. In Section 3 we construct a kernel for the problem of $\mathcal{O}(k!)$ size. In Section 4 we perform a combinatorial analysis of the case $k = 5$ which leaves 20 simple types of the sequence $\mathbf{x}$ to be considered. This analysis is used in Section 5 to obtain 20 special ILP instances with only 4 variables. They could be solved using Lenstra's ILP solver. However, there exists an efficient algorithm tailored for this type of special instances which we describe in Section 6. Finally we analyze the performance of a C++ implementation of the algorithm for $k \leq 5$ and provide some conclusions in Section 7.

2. From MSC problem to ILP

Let us fix a collection $\mathcal{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_k)$ of the input sequences. The elements of $\mathbf{a}_i$ are denoted by $a_{i,j}$ (for $1 \leq j \leq \ell$).

For $j \in \{1, \ldots, n\}$ let π_j be a permutation of $\{1, \ldots, k\}$ such that $a_{\pi_j(1),j} \leq \ldots \leq a_{\pi_j(k),j}$, i.e. π_j is the ordering permutation of elements $a_{1,j}, \ldots, a_{k,j}$. We also set $s_{i,j} = a_{\pi_j(i),j}$; see Example 2. As for some j there might be several possibilities for π_j (if $a_{i,j} = a_{i',j}$ for some $i \neq i'$), we fix a single choice for each j.

Example 2. Consider the following three sequences $\mathbf{a}_i$ and sequences $\mathbf{s}_i$ obtained by sorting columns:

$$[a_{i,j}] = \begin{bmatrix} 120 & 0 & 80 \\ 20 & 40 & 130 \\ 0 & 100 & 0 \end{bmatrix}, \quad [s_{i,j}] = \begin{bmatrix} 0 & 0 & 0 \\ 20 & 40 & 80 \\ 120 & 100 & 130 \end{bmatrix}.$$

The Manhattan consensus sequence is $\mathbf{x} = (30, 40, 60)$; see Fig. 1. In the figure, the circled numbers in the j-th column are $\pi_j(1), \pi_j(2), \ldots, \pi_j(k)$ (top-down).

Definition 1. A *basic interval* is an interval of the form $[i, i+1]$ (for $i = 1, \ldots, k-1$) or $[i, i]$ (for $i = 1, \ldots, k$). The former is called *unit* and the latter *degenerate*. An *interval system* is a sequence $\mathcal{I} = (I_1, \ldots, I_\ell)$ of basic intervals I_j.

By $[\![a, b]\!]$ we denote the interval of integers $\{a, \ldots, b\}$. For a basic interval I_j we say that a value x_j is *consistent* with I_j if $x_j \in [\![s_{i,j}, s_{i+1,j}]\!]$ when $I_j = [i, i+1]$ is unit, and if $x_j = s_{i,j}$ when $I_j = [i, i]$ is degenerate. A sequence $\mathbf{x}$ is called *consistent* with an interval system $\mathcal{I} = (I_j)_{j=1}^{\ell}$ if for each j the value x_j is consistent with I_j.

For an interval system $\mathcal{I}$ we define $\mathrm{OPT}(\mathcal{A}, \mathcal{I})$ as the minimum $dist(\mathbf{x}, \mathcal{A})$ among all integer sequences $\mathbf{x}$ consistent with $\mathcal{I}$. Due to the following trivial observation, for every $\mathcal{A}$ there exists an interval system $\mathcal{I}$ such that $\mathrm{OPT}(\mathcal{A}) = \mathrm{OPT}(\mathcal{A}, \mathcal{I})$.

Observation 1. *If* $\mathbf{x}$ *is a Manhattan consensus sequence, then for each* j, $s_{1,j} \leq x_j \leq s_{k,j}$.

2.1. Transformation of the input to an ILP

We will show that the problem of finding $\mathrm{OPT}(\mathcal{A}, \mathcal{I})$ can be formulated as an ILP. If a sequence $\mathbf{x}$ is consistent with a fixed interval system $\mathcal{I}$, the Manhattan distance $dist(\mathbf{x}, \mathbf{a}_i)$ can be expressed as:

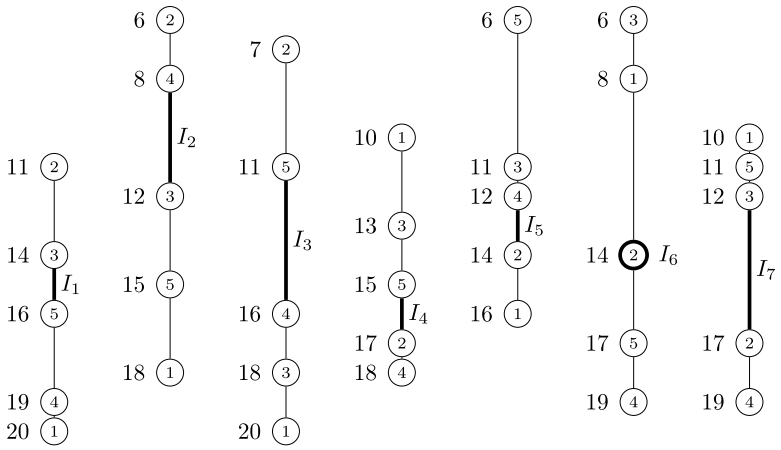

Fig. 2. Illustration of Example 3: 5 sequences of length 7 together with an interval system. Notice that I_6 is a degenerate interval.

$$dist(\mathbf{x}, \mathbf{a}_i) = \sum_{\substack{I_j=[b,b+1],\\ a_{i,j}\le s_{b,j}}} (x_j - a_{i,j}) + \sum_{\substack{I_j=[b,b+1],\\ a_{i,j}\ge s_{b+1,j}}} (a_{i,j} - x_j) + \sum_{I_j=[b,b]} |s_{b,j} - a_{i,j}|. \tag{1}$$

In other words, for a unit interval $I_j = [b, b+1]$ we avoid the absolute value, as we know the relative order of x_j and $a_{i,j}$ based on the relative order of $a_{i,j}$ and the endpoints of the interval $[\![s_{b,j}, s_{b+1,j}]\!]$. For a degenerate interval I_j, x_j is fixed to $s_{b,j}$.

In the ILP we introduce a variable x_j with a range $[\![s_{b,i}, s_{b,i+1}]\!]$ for every unit interval $I_j = [b, b+1]$. Also an auxiliary variable z is used to represent $\mathrm{OPT}(\mathcal{A}, \mathcal{I})$. We want to minimize z under the constraints "$dist(\mathbf{x}, \mathbf{a}_i) \le z$", each expressed using the formula (1). We denote the resulting ILP by $\mathrm{ILP}(\mathcal{I})$; see Example 3.

Lemma 1. *The optimal value of* $\mathrm{ILP}(\mathcal{I})$ *is equal to* $\mathrm{OPT}(\mathcal{A}, \mathcal{I})$.

Proof. Any sequence $\mathbf{x}$ that is consistent with $\mathcal{I}$ is represented by an assignment of variables x_j in $\mathrm{ILP}(\mathcal{I})$. The i-th constraint of $\mathrm{ILP}(\mathcal{I})$ algebraically represents $dist(\mathbf{x}, \mathbf{a}_i)$. In $\mathrm{ILP}(\mathcal{I})$ we are looking for an assignment of the variables that minimizes $z = \max_i\{dist(\mathbf{x}, \mathbf{a}_i)\}$. Hence, the resulting value of z equals $\mathrm{OPT}(\mathcal{A}, \mathcal{I})$. □

Example 3. Consider the following 5 sequences of length 7:

$$[a_{i,j}] = \begin{bmatrix} 20 & 18 & 20 & 10 & 16 & 8 & 10 \\ 11 & 6 & 7 & 17 & 14 & 14 & 17 \\ 14 & 12 & 18 & 13 & 11 & 6 & 12 \\ 19 & 8 & 16 & 18 & 12 & 19 & 19 \\ 16 & 15 & 11 & 15 & 6 & 17 & 11 \end{bmatrix}$$

and an interval system $\mathcal{I} = ([2,3],\ [2,3],\ [2,3],\ [3,4],\ [3,4],\ [3,3],\ [3,4])$. An illustration of both can be found in Fig. 2.

We obtain the following $\mathrm{ILP}(\mathcal{I})$, where $x_1 \in [\![14, 16]\!]$, $x_2 \in [\![8, 12]\!]$, $x_3 \in [\![11, 16]\!]$, $x_4 \in [\![15, 17]\!]$, $x_5 \in [\![12, 14]\!]$, $x_7 \in [\![12, 17]\!]$ and the sequence $\mathbf{x}$ can be retrieved as $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 14, x_7)$:

$$\begin{array}{rcrcrcrcrcrcrl} & & & & & & & & & & & & & \min z \\ 20 - x_1 & + & 18 - x_2 & + & 20 - x_3 & + & x_4 - 10 & + & 16 - x_5 & + & 6 & + & x_7 - 10 & \le z \\ x_1 - 11 & + & x_2 - 6 & + & x_3 - 7 & + & 17 - x_4 & + & 14 - x_5 & + & 0 & + & 17 - x_7 & \le z \\ x_1 - 14 & + & 12 - x_2 & + & 18 - x_3 & + & x_4 - 13 & + & x_5 - 11 & + & 8 & + & x_7 - 12 & \le z \\ 19 - x_1 & + & x_2 - 8 & + & 16 - x_3 & + & 18 - x_4 & + & x_5 - 12 & + & 5 & + & 19 - x_7 & \le z \\ 16 - x_1 & + & 15 - x_2 & + & x_3 - 11 & + & x_4 - 15 & + & x_5 - 6 & + & 3 & + & x_7 - 11 & \le z \end{array}$$

Note that $P = \mathrm{ILP}(\mathcal{I})$ has the following special form, which we call *basic ILP*:

$$\begin{aligned} &\min z \\ d_i + \sum_j x_j e_{i,j} &\le z \\ x_j &\in R_P(x_j) \end{aligned}$$

where $e_{i,j} = \pm 1$ and $R_P(x_j) = [\![\ell_j, r_j]\!]$ for integers $\ell_j \le r_j$. Whenever we refer to variables, it does not apply to z, which is of auxiliary character. Also, "$x_j \in R_P(x_j)$" are called variable ranges rather than constraints. We say that $(e_{1,j}, \ldots, e_{k,j})$ is a *coefficient vector* of x_j and denote it as $E_P(x_j)$. If the program P is apparent from the context, we omit the subscript.

2.2. Simplification of ILP

The following two lemmas are used to reduce the number of variables of a basic ILP. For $A, B \subseteq \mathbb{Z}$ we define $-A = \{-a : a \in A\}$ and $A + B = \{a + b : a \in A, b \in B\}$.

Lemma 2. *Let P be a basic ILP with n variables. Let P' be a program obtained from P by replacing a variable x_j with $-x_j$, i.e. setting $E_{P'}(x_j) = -E_P(x_j)$ and $R_{P'}(x_j) = -R_P(x_j)$. Then $\mathrm{OPT}(P) = \mathrm{OPT}(P')$. Moreover, knowing the optimal variables' assignment for P', we can compute the optimal variables' assignment for P in $\mathcal{O}(1)$ time.*

Proof. For every feasible solution $(z, x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n)$ of P, $(z, x_1, \ldots, x_{j-1}, -x_j, x_{j+1}, \ldots, x_n)$ is a feasible solution of P', and *vice versa*. □

Lemma 3. *Let P be a basic ILP with n variables. Assume $E_P(x_j) = E_P(x_{j'})$ for $j \neq j'$. Let P' be a program obtained from P by removing the variable $x_{j'}$ and replacing x_j with $x_j + x_{j'}$, i.e. setting $R_{P'}(x_j) = R_P(x_j) + R_P(x_{j'})$. Then $\mathrm{OPT}(P) = \mathrm{OPT}(P')$. Moreover, knowing the optimal variables' assignment for P', we can compute the optimal variables' assignment for P in $\mathcal{O}(1)$ time.*

Proof. Let $(z, x_1, \ldots, x_n)$ be a feasible solution of P. Then setting $x_j := x_j + x_{j'}$ and removing the variable $x_{j'}$ we obtain a feasible solution of P'. Therefore $\mathrm{OPT}(P') \le \mathrm{OPT}(P)$. For the proof of the other inequality, take a feasible solution $(z, x_1, \ldots, x_n)$ (with $x_{j'}$ missing) of P'. Note that $x_j \in R_{P'}(x_j) = R_P(x_j) + R_P(x_{j'})$. Therefore one can split x_j into $x_j + x_{j'}$ so that $x_j \in R_P(x_j)$ and $x_{j'} \in R_P(x_{j'})$. This way we obtain a feasible solution of P and thus prove that $\mathrm{OPT}(P) \le \mathrm{OPT}(P')$. □

Corollary 1. *For a basic ILP with k constraints one can compute in linear time an equivalent basic ILP with k constraints and up to 2^{k-1} variables.*

Proof. We apply Lemma 2 to obtain $e_{1,1} = e_{1,2} = \ldots = e_{1,\ell}$. This leaves at most 2^{k-1} different coefficient vectors. Afterwards we apply Lemma 3 as many times as possible so that there is exactly one variable with each coefficient vector. □

Example 4. Consider the basic ILP P from Example 3. Observe that $E_P(x_4) = E_P(x_7) = -E_P(x_2)$ and thus Lemmas 2 and 3 let us merge x_2 and x_7 into x_4 with

$$R_{P'}(x_4) = R_P(x_4) + R_P(x_7) - R_P(x_2) = [\![15, 17]\!] + [\![12, 17]\!] + [\![-12, -8]\!] = [\![15, 26]\!].$$

By simplifying the constant terms, we obtain the following basic ILP P':

$$\begin{array}{rcrcrcrcrl} & & & & & & & & & \min z \\ -x_1 & - & x_3 & + & x_4 & - & x_5 & + & 60 & \le z \\ +x_1 & + & x_3 & - & x_4 & - & x_5 & + & 24 & \le z \\ +x_1 & - & x_3 & + & x_4 & + & x_5 & - & 12 & \le z \\ -x_1 & - & x_3 & - & x_4 & + & x_5 & + & 57 & \le z \\ -x_1 & + & x_3 & + & x_4 & + & x_5 & - & 9 & \le z \end{array}$$

3. Kernel of MSC for arbitrary k

In this section we give a kernel for the MSC problem parameterized with k, which we then apply to develop a linear-time FPT algorithm. To obtain the kernel we need a combinatorial observation that if $\pi_j = \pi_{j'}$ then the j-th and the j'-th column in $\mathcal{A}$ can be merged. This is stated formally in the following lemma.

Lemma 4. *Let $\mathcal{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_k)$ be a collection of sequences of length ℓ and assume that $\pi_j = \pi_{j'}$ for some $1 \le j < j' \le \ell$. Let $\mathcal{A}' = (\mathbf{a}'_1, \ldots, \mathbf{a}'_k)$ be a collection of sequences of length $\ell - 1$ obtained from $\mathcal{A}$ by removing the j'-th column and setting $a'_{i,j} = a_{i,j} + a_{i,j'}$. Then $\mathrm{OPT}(\mathcal{A}) = \mathrm{OPT}(\mathcal{A}')$. Moreover, the optimal sequence $\mathbf{x}$ for $\mathcal{A}$ can be computed in $\mathcal{O}(k \log k)$ time from the optimal sequence $\mathbf{x}'$ for $\mathcal{A}'$.*

Proof. First, let us show that $\mathrm{OPT}(\mathcal{A}') \le \mathrm{OPT}(\mathcal{A})$. Let $\mathbf{x}$ be a Manhattan consensus sequence for $\mathcal{A}$ and let $\mathbf{x}'$ be obtained from $\mathbf{x}$ by removing the j'-th entry and setting $x'_j = x_j + x_{j'}$. We claim that $dist(\mathbf{x}', \mathcal{A}') \le dist(\mathbf{x}, \mathcal{A})$. Note that it suffices to show that $|x'_j - a'_{i,j}| \le |x_j - a_{i,j}| + |x_{j'} - a_{i,j'}|$ for all i. However, with $x'_j = x_j + x_{j'}$ and $a'_{i,j} = a_{i,j} + a_{i,j'}$, this is a direct consequence of the triangle inequality.

It remains to prove that $\mathrm{OPT}(\mathcal{A}) \leq \mathrm{OPT}(\mathcal{A}')$. Let $\mathbf{x}'$ be a Manhattan consensus sequence for $\mathcal{A}'$. By Observation 1, x'_j is consistent with some unit basic interval $[i, i+1]$. Let $d'_{i,j} = x'_j - s'_{i,j}$ and $D'_{i,j} = s'_{i+1,j} - s'_{i,j}$. Also, let $D_{i,j} = s_{i+1,j} - s_{i,j}$ and $D_{i,j'} = s_{i+1,j'} - s_{i,j'}$. Note that, since $\pi_j = \pi_{j'}$, $D'_{i,j} = D_{i,j} + D_{i,j'}$. Thus, one can write $d'_{i,j}$ as $d_{i,j} + d_{i,j'}$ so that both $d_{i,j}$ and $d_{i,j'}$ are non-negative integers not exceeding $D_{i,j}$ and $D_{i,j'}$, respectively. Particularly, one may choose $d_{i,j} = \min(d'_{i,j}, D_{i,j})$, $d_{i,j'} = d'_{i,j} - d_{i,j}$. We set $x_j = s_{i,j} + d_{i,j}$ and $x_{j'} = s_{i,j'} + d_{i,j'}$, and the remaining components of $\mathbf{x}$ correspond to components of $\mathbf{x}'$. Note that $x'_j = x_j + x_{j'}$ and that both x_j and $x_{j'}$ are consistent with $[i, i+1]$. Consequently, for any sequence $\mathbf{a}_m$ it holds that $dist(\mathbf{x}, \mathbf{a}_m) = dist(\mathbf{x}', \mathbf{a}_m)$ and therefore $dist(\mathbf{x}, \mathcal{A}) = dist(\mathbf{x}', \mathcal{A}')$, which concludes the proof.

The above paragraph shows a procedure to convert the optimum solution $\mathbf{x}'$ for $\mathcal{A}'$ to the optimum solution $\mathbf{x}$ for $\mathcal{A}$. To implement it, we require the values $s_{i,j}$, $s_{i,j'}$, and $s'_{i,j}$ for $i = 1 \ldots, k$, which can be computed via sorting the columns $a_{i,j}$, $a_{i,j'}$, and $a'_{i,j}$ for $i = 1 \ldots, k$ in $\mathcal{O}(k \log k)$ time. □

By Lemma 4, to obtain the desired kernel we need to sort the elements in columns of $\mathcal{A}$ and afterwards sort the resulting permutations π_j.

Theorem 1. *In $\mathcal{O}(\ell k \log k)$ time one can reduce any instance of MSC to an instance with k sequences of length ℓ', with $\ell' \leq k!$.*

Proof. Sorting of the ℓ columns takes $\mathcal{O}(lk \log k)$ time in total. Afterwards we obtain the permutations π_j in $\mathcal{O}(k\ell)$ time. The next step is sorting of permutations π_j, which takes $\mathcal{O}(k\ell)$ time if we use radix sort [7]. Finally, merging of columns using Lemma 4 takes $\mathcal{O}(\ell k \log k)$ time in total. □

Remark 1. For binary instances, if permutations π_j are chosen appropriately, it holds that $\ell' \leq 2^k$.

Proof. For binary sequences there are at most 2^k different possible columns. By merging all equal columns together, we obtain a collection of sequences for which $\ell' \leq 2^k$. □

Theorem 2. *For any integer k, the* Manhattan Sequence Consensus *problem can be solved in $\mathcal{O}(\ell k \log k + 2^{k! \log k + O(2^k k)} \log M)$ time.*

Proof. We solve the kernel from Theorem 1 by considering all possible interval systems $\mathcal{I}$ composed of unit intervals. The sequences in the kernel have length at most $k!$, which gives $(k-1)^{k!}$ basic ILPs of the form ILP($\mathcal{I}$) to solve.

Each of the basic ILPs initially has k constraints on $k!$ variables but, due to Corollary 1, the number of variables can be reduced to 2^{k-1}. Lenstra's algorithm with further improvements [14,10,20] solves ILP with p variables in $\mathcal{O}(p^{2.5p+o(p)} \log L)$ time, where L is the bound on the scope of variables. In our case $L = \mathcal{O}(\ell M)$, which gives the time complexity of:

$$\mathcal{O}\left((k-1)^{k!} \cdot 2^{(k-1)(2.5 \cdot 2^{k-1} + o(2^{k-1}))} \log M\right) = \mathcal{O}\left(2^{k! \log k + O(2^k k)} \log M\right).$$

This concludes the proof of the theorem. □

4. Combinatorial characterization of solutions for $k = 5$

In this section we characterize those Manhattan consensus sequences $\mathbf{x}$ which additionally minimize $\sum_i dist(\mathbf{x}, \mathbf{a}_i)$ among all Manhattan consensus sequences. Such sequences are called here *sum-MSC sequences*. We show that, if $k = 5$, then for every input one can determine a collection of 20 interval systems, so that any sum-MSC sequence is guaranteed to be consistent with one of them. We also prove some structural properties of these systems, which are then useful to efficiently solve the corresponding basic ILPs.

We say that x_j is in the *center* if $x_j = s_{3,j}$, i.e. x_j is equal to the column median. Note that if $x_j \neq s_{3,j}$, then moving x_j by one towards the center decreases by one $dist(\mathbf{x}, \mathbf{a}_i)$ for at least three sequences $\mathbf{a}_i$.

Definition 2. We say that $\mathbf{a}_i$ *governs* x_j if x_j is in the center or moving x_j towards the center increases $dist(\mathbf{x}, \mathbf{a}_i)$. The set of indices i such that $\mathbf{a}_i$ governs x_j is denoted as $G_j(\mathbf{x})$.

Observe that if x_j is in the center, then $|G_j(\mathbf{x})| = 5$, and otherwise $|G_j(\mathbf{x})| \leq 2$; see Fig. 3. For $k = 5$ we have 4 unit basic intervals: $[1, 2]$, $[2, 3]$, $[3, 4]$, and $[4, 5]$. We call $[1, 2]$ and $[4, 5]$ *border* intervals, and the other two *middle intervals*. We define $G_j([1, 2]) = \{\pi_j(1)\}$, $G_j([2, 3]) = \{\pi_j(1), \pi_j(2)\}$, $G_j([3, 4]) = \{\pi_j(4), \pi_j(5)\}$, and $G_j([4, 5]) = \{\pi_j(5)\}$. Note that if we know $G_j(\mathbf{x})$ and $|G_j(\mathbf{x})| \leq 2$, then we are guaranteed that x_j is consistent with the basic interval I_j for which $G_j(I_j) = G_j(\mathbf{x})$.

Observe that if $\mathbf{x}$ is a Manhattan consensus sequence, then $G_j(\mathbf{x}) \neq \emptyset$ for any j. If we additionally assume that $\mathbf{x}$ is a sum-MSC sequence, we obtain a stronger property.

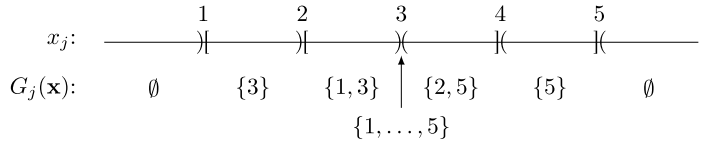

Fig. 3. Assume $a_{1,j}=2$, $a_{2,j}=4$, $a_{3,j}=1$, $a_{4,j}=3$, $a_{5,j}=5$. Then $G_j(x)$ depends on the interval of x_j as shown in the figure. E.g., if $1 \le x_j < 2$ then $G_j(\mathbf{x}) = \{3\}$.

Lemma 5. *Let* $\mathbf{x}$ *be a sum-MSC sequence. Then* $G_j(\mathbf{x}) \cap G_{j'}(\mathbf{x}) \neq \emptyset$ *for any* j, j'.

Proof. For a proof by contradiction assume $G_j(\mathbf{x})$ and $G_{j'}(\mathbf{x})$ are disjoint. This implies that neither x_j nor $x_{j'}$ is in the center and thus $|G_j(\mathbf{x})|, |G_{j'}(\mathbf{x})| \le 2$. Let us move both x_j and $x_{j'}$ by one towards the center. Then $dist(\mathbf{x}, \mathbf{a}_i)$ remains unchanged for $i \in G_j(\mathbf{x}) \cup G_{j'}(\mathbf{x})$ (by disjointness), and decreases by two for the remaining sequences $\mathbf{a}_i$. There must be at least one such remaining sequence, since $k = 5$, which contradicts our choice of $\mathbf{x}$. □

Additionally, if a sum-MSC sequence $\mathbf{x}$ has a position j with $|G_j(\mathbf{x})| = 1$, the structure of $\mathbf{x}$ needs to be even more regular.

Definition 3. A sequence $\mathbf{x}$ is called an *i-border sequence* if for each j it holds that $x_j = a_{i,j}$ or $G_j(\mathbf{x}) = \{i\}$.

Lemma 6. *Let* $\mathbf{x}$ *be a sum-MSC sequence. If* $G_j(\mathbf{x}) = \{i\}$ *for some* j, *then* $\mathbf{x}$ *is an* i*-border sequence.*

Proof. For a proof by contradiction assume $G_{j'}(\mathbf{x}) \neq \{i\}$ and $x_{j'} \neq a_{i,j'}$ for some j'. Let us move x_j towards the center and $x_{j'}$ towards $a_{i,j'}$, both by one. Then for every i', $dist(\mathbf{x}, \mathbf{a}_{i'})$ does not increase. By Lemma 5, $i \in G_{j'}(\mathbf{x})$, so $x_{j'}$ is moved away from the center. Moreover, $x_{j'}$ is moved towards some $a_{i',j'}$ with $i' \neq i$, since $G_{j'}(\mathbf{x}) \neq \{i\}$. Consequently, $dist(\mathbf{x}, \mathbf{a}_{i'})$ decreases by two, which contradicts our choice of $\mathbf{x}$. □

Definition 4. A sequence $\mathbf{x}$ is called an *i-middle sequence* if for each j it holds that $i \in G_j(\mathbf{x})$ and $|G_j(\mathbf{x})| \ge 2$.

Definition 5. For a 3-element set $\Delta \subseteq \{1, \ldots, 5\}$, a sequence $\mathbf{x}$ is called a *Δ-triangle sequence* if for each j it holds that $|\Delta \cap G_j(\mathbf{x})| \ge 2$.

Lemma 7. *Let* $\mathbf{x}$ *be a sum-MSC sequence. Then* $\mathbf{x}$ *is a border sequence, a middle sequence, or a triangle sequence.*

Proof. Recall that $G_j(\mathbf{x}) \neq \emptyset$ for each j. By Lemma 6, if $G_j(\mathbf{x}) = \{i\}$ for some j, then $\mathbf{x}$ is an i-border sequence. Otherwise, for each j we have $|G_j(x)| \ge 2$, i.e. $|G_j(\mathbf{x})| \in \{2, 5\}$. Let $\mathcal{F}$ be the family of 2-element sets among $G_j(\mathbf{x})$. By Lemma 5 every two of them intersect, so we can apply the following easy set-theoretical claim.

Claim 1. *Let* $\mathcal{G}$ *be a family of 2-element sets such that every two sets in* $\mathcal{G}$ *intersect. Then all sets in* $\mathcal{G}$ *share a common element or* $\mathcal{G}$ *contains exactly three sets with three elements in total.*

Proof. Assume that there are two sets $A, B \in \mathcal{G}$ such that $A \neq B$; otherwise the claim is trivial. Let $A \cap B = \{u\}$, $A = \{u, v\}$, $B = \{u, w\}$. Then either all the remaining elements of $\mathcal{G}$ contain u, or there exists a set $C \in \mathcal{G}$ such that $u \notin C$. In the latter case, $C \cap A = \{v\}$ and $C \cap B = \{w\}$. Consequently, A, B, C form a desired collection of three sets with three elements in total. □

If all the sets in $\mathcal{F}$ share an element i, then $\mathbf{x}$ is clearly an i-middle sequence. Otherwise $\mathbf{x}$ is a Δ-triangle sequence for $\Delta = \bigcup \mathcal{F}$. □

Lemma 7 allows $5 + 5 + \binom{5}{3} = 20$ different types of sequences $\mathbf{x}$. The following, final combinatorial lemma shows that, for every collection $\mathcal{A}$, each type of sequence $\mathbf{x}$ can be characterized by a single interval system.

Lemma 8. *There exist 20 interval systems* $\mathcal{B}_i, \mathcal{M}_i$ *(for* $i \in \{1, \ldots, 5\}$*) and* $\mathcal{T}_\Delta$ *(for 3-element sets* $\Delta \subseteq \{1, \ldots, 5\}$*) such that:*

(a) $\mathcal{B}_i = (\mathcal{B}_{i,1}, \ldots, \mathcal{B}_{i,\ell})$ *is consistent with each* i*-border sequence and* $G_j(\mathcal{B}_{i,j}) = \{i\}$ *for unit* $\mathcal{B}_{i,j}$*;*
(b) $\mathcal{M}_i = (\mathcal{M}_{i,1}, \ldots, \mathcal{M}_{i,\ell})$ *is consistent with each* i*-middle sequence and if* $\mathcal{M}_{i,j}$ *is unit then* $|G_j(\mathcal{M}_{i,j})| = 2$ *and* $i \in G_j(\mathcal{M}_{i,j})$*;*
(c) $\mathcal{T}_\Delta = (\mathcal{T}_{\Delta,1}, \ldots, \mathcal{T}_{\Delta,\ell})$ *is consistent with each* Δ*-triangle sequence and if* $\mathcal{T}_{\Delta,j}$ *is unit then* $|G_j(\mathcal{T}_{\Delta,j})| = 2$ *and* $G_j(\mathcal{T}_{\Delta,j}) \subseteq \Delta$.

Proof. (a) Let us fix a position j. For any i-border sequence $\mathbf{x}$, we know that $x_j = a_{i,j}$ or $G_j(\mathbf{x}) = \{i\}$. If either of the border intervals I satisfies $G_j(I) = \{i\}$, we set $\mathcal{B}_{i,j} := I$ (observe that $a_{i,j}$ is then consistent with I). Otherwise we choose $\mathcal{B}_{i,j}$ so that it is degenerate and corresponds to $x_j = a_{i,j}$.

(b) Again fix j. For any i-middle sequence $\mathbf{x}$, we know that x_j is consistent with at least one of the two middle intervals (both if x_j is in the center). If either of the middle intervals I satisfies $i \in G_j(I)$, we choose $\mathcal{M}_{i,j} := I$. (Note that this condition cannot hold for both middle intervals.) Otherwise we know that x_j is in the center and set $\mathcal{M}_{i,j}$ so that it is degenerate and corresponds to x_j in the center, i.e. $\mathcal{M}_{i,j} := [3, 3]$.

(c) We act as in (b), i.e. if either of the middle intervals I satisfies $|G_j(I) \cap \Delta| = 2$, we choose $\mathcal{T}_{\Delta,j} := I$ (because sets $G_j(I)$ are disjoint for both middle intervals, this condition cannot hold for both of them). Otherwise, we set $\mathcal{T}_{\Delta,j} := [3, 3]$, since x_j is guaranteed to be in the center for any Δ-triangle sequence $\mathbf{x}$. □

5. Practical algorithm for $k \le 5$

It suffices to consider $k = 5$, as otherwise we may add $5 - k$ copies of one of the sequences. Using Lemma 8 we reduce the number of interval systems from $(k-1)^{k!} = 4^{5!} > 10^{72}$ to 20 compared to the algorithm of Section 3. Moreover, we show below that for each of them ILP($\mathcal{I}$) admits structural properties, which lets us compute OPT($\mathcal{A}, \mathcal{I}$) much more efficiently than using a general ILP solver.

Definition 6. A basic ILP is called an *easy ILP* if for each constraint the number of $+1$ coefficients is 0, 1, or n, where n is the number of variables, and no two constraints have equal coefficients for each variable.

Note that if two constraints have equal coefficients for each variable, then the constraint with smaller constant term is weaker than the other, and thus it can be removed.

Lemma 9. *For each $\mathcal{I}$ being one of the 20 interval systems $\mathcal{B}_i$, $\mathcal{M}_i$, and $\mathcal{T}_\Delta$,* ILP($\mathcal{I}$) *can be reduced to an equivalent easy ILP with up to 4 variables.*

Proof. Recall that for degenerate intervals I_j, we do not introduce variables. On the other hand, if I_j is unit, possibly negating the variable x_j (Lemma 2), we can make sure that the coefficient vector $E(x_j)$ has $+1$ entries corresponding to $i \in G_j(I_j)$ and -1 entries for the remaining i. Moreover, merging the variables (Lemma 3), we end up with a single variable per possible value $G_j(I_j)$. Now we use structural properties stated in Lemma 8 to claim that the basic ILP we obtain this way, possibly after further variable negations, becomes an easy ILP.

Border sequences. By Lemma 8(a), if $\mathcal{B}_{i,j}$ is unit, then $G_j(\mathcal{B}_{i,j}) = \{i\}$, and thus the basic ILP has at most 1 variable and, consequently, is an easy ILP.

Middle sequences. By Lemma 8(b), if $\mathcal{M}_{i,j}$ is unit, then $G_j(\mathcal{M}_{i,j}) = \{i, i'\}$ for some $i' \neq i$. Thus there are up to 4 variables, the constraint corresponding to i has only $+1$ coefficients, and the remaining constraints have at most one $+1$.

Triangle sequences. By Lemma 8(c), if $\mathcal{T}_{\Delta,j}$ is unit, then $G_j(\mathcal{T}_{\Delta,j})$ is a 2-element subset of Δ, and thus there are up to three variables. Any basic ILP with up to two variables is an easy ILP, and if we obtain three variables, then the constraints corresponding to $i \in \Delta$ have exactly two $+1$ coefficients, while the constraints corresponding to $i \notin \Delta$ have just -1 coefficients. Now, negating each variable (Lemma 2), we get one $+1$ coefficient in constraints corresponding to $i \in \Delta$ and all $+1$ coefficients for $i \notin \Delta$. □

The algorithm of Lenstra [18] with further improvements [14,10,20], which runs in roughly $n^{2.5n+o(n)}$ time, could perform reasonably well for $n = 4$. However, in the following section we design a simple $\mathcal{O}(n^2)$-time algorithm solving easy ILP.

This is the last ingredient for an efficient $\mathcal{O}(\ell)$-time solution of the MSC problem with $k = 5$. In conclusion, the algorithm for MSC problem first proceeds as described in Lemma 8 to obtain the interval systems $\mathcal{B}_i$, $\mathcal{M}_i$ and $\mathcal{T}_\Delta$. For each of them it computes ILP($\mathcal{I}$), as described in Section 2, and converts it to an equivalent easy ILP following Lemma 9. Finally, it uses the efficient algorithm to solve each of these 20 basic ILPs. The final result is the minimum of the optima obtained. Note that, as an optimization, the algorithm may use the kernelization procedure of Theorem 1 to reduce the length of the sequences to $5! = 120$ or even to $2^5 = 32$ if we aim to solve the HSC problem for binary sequences.

6. Solving easy ILP

Recall the definition of an easy ILP:

$$\min z$$
$$d_i + \sum_j x_j e_{i,j} \le z$$
$$x_j \in R(x_j)$$

where $e_{i,j} = \pm 1$, and for each constraint the number of $+1$ coefficients is 0, 1 or n, where n is the number of variables. Moreover $R(x_j) = [\![\ell_j, r_j]\!]$ for integers $\ell_j \le r_j$.

We define a *canonical* form of easy ILP as follows:

$$\begin{aligned} &\min \quad z \\ y_1 + y_2 + \ldots + y_n + K' &\le z \\ y_1 - y_2 - \ldots - y_n + K_1 &\le z \\ -y_1 + y_2 - \ldots - y_n + K_2 &\le z \\ &\vdots \\ -y_1 - y_2 - \ldots + y_n + K_n &\le z \\ y_i &\in [\![0, D_i]\!] \end{aligned}$$

where $D_i \in \mathbb{Z}_{\ge 0}$ and $K', K_1, \ldots, K_n$ are of the same parity.

Lemma 10. *Any easy ILP with n variables and at least one constraint can be reduced in $\mathcal{O}(n^2)$ time to two easy ILPs in the canonical form, with $n+1$ variables, so that the optimum value of the input basic ILP is equal to the smaller optimum value of the two resulting basic ILPs. Moreover, one can reconstruct in $\mathcal{O}(n)$ time the optimal variables' assignment for the original basic ILP from the assignments of the two constructed basic ILPs.*

Proof. First, observe that the substitution $x_i = y_i + \ell_i$ affects only the constant terms and the ranges of variables, and gives a basic ILP where the range of y_i is $[\![0, D_i]\!]$ for $D_i = r_i - \ell_i$.

Recall that the number of $+1$ coefficients in a constraint of an easy ILP is 0, 1 or n. Consequently, there are at most $n+2$ constraints: one (denoted C_+) only with coefficients $+1$, one (denoted C_-) only with coefficients -1, and for each variable y_j one constraint (denoted C_j) with coefficient $+1$ for y_j and -1 for the remaining variables. Also, unless there are no constraints at all, if some of these $n+2$ constraints are missing, one may add it, setting the constant term to a sufficiently small number, e.g. $K - 2\sum_i D_i$, where K is the constant term of any existing constraint C. Then, with such a constant term, the introduced constraint follows from the existing one.

Finally, we introduce a dummy variable y_{n+1} with $D_{n+1} = 0$ and coefficient $+1$ in C_- and C_+, and coefficient -1 in the constraints C_j. This way C_- can be interpreted as C_{n+1}, and we obtain a basic ILP in the canonical form. However, it does not satisfy the parity condition yet.

Now, we replace the basic ILP obtained so far, denoted as P, with two copies: in one of them, P_1, we increment (by one) each odd constant term, and in the other, P_2, we increment (by one) each even constant term. Clearly, both these basic ILPs are stronger than the original basic ILP. Nevertheless, we claim that the originally feasible solution remains feasible for one of them so it suffices to return the minimal of the optimal solutions to P_1 and P_2.

Consider a feasible solution $(z, y_1, \ldots, y_{n+1})$ for P and the parity of $q = z + y_1 + \ldots + y_{n+1}$. Note that changing a $+1$ coefficient to a -1 coefficient does not change the parity, so only constraints with constant terms of parity $q \bmod 2$ may be tight in P. For the remaining ones, we can increment the constant term without violating the feasibility. If $q \bmod 2 = 0$, this shows that $(z, y_1, \ldots, y_{n+1})$ is a feasible solution for P_1, and otherwise it is a feasible solution for P_2.

Now it is easy to reconstruct the optimal solution for the original basic ILP from the optimal solutions to P_1 and P_2: we choose the one with minimal z, dispose of the variable y_{n+1}, and set $x_i = y_i + \ell_i$. This completes the proof of the lemma. □

In the following we focus on solving easy ILPs in the canonical form. We will apply subsequent algorithm transformations to finally arrive at an efficient solution.

From now on we assume that $K_1 \le \ldots \le K_n$ (constraints and variables can be renumbered accordingly). By C_+ and C_i we denote the constraint with all coefficients equal to $+1$ and with the coefficient $+1$ at y_i, respectively. The following function `Solve` is our first approach to solving easy ILPs of the specified form. This function is rather inefficient; we improve it later on. An explanation of the pseudocode is contained in the proof of the following lemma.

Lemma 11. `Solve` *correctly determines the optimum value of the easy ILP in the canonical form, provided that $K_1 \le \ldots \le K_n$. It can be augmented with computing the optimal variables' assignment without increase of time complexity.*

Proof. If $n = 1$ then the constraints are $y_1 + K' \le z$ and $y_1 + K_1 \le z$, therefore the optimum is clearly $y_1 = 0$ and $z = \min(K', K_1)$. Consequently, line 1 is correct and in the following we may assume that $n \ge 2$.

Due to the constraint $\sum_i y_i + K' \le z$ and the fact that y_i are non-negative, the value K' is clearly a lower bound on z. Moreover, if $K_n \le K'$, then taking $y_1 = \ldots = y_n = 0$ and $z = K'$ is feasible, since in that case all the constraints reduce to $K_i \le K'$ which is true due to monotonicity of K_i. Thus, we may exit the while-loop and return K' in line 12 when $K_n \le K'$.

```
1  if n = 1 then return max(K', K_1);
2  while K' < K_n do
3  |  ℓ := max{i : K_i = K_1};
4  |  for i := 1 to ℓ do
5  |  |  if D_i = 0 then
6  |  |  |  return Solve(K', K_1, ..., K_{i-1}, K_{i+1}, ..., K_n, D_1, ..., D_{i-1}, D_{i+1}, ..., D_n);
7  |  K' := K' + 1;
8  |  for i := 1 to ℓ − 1 do K_i := K_i − 1;
9  |  K_ℓ := K_ℓ + 1;
10 |  D_ℓ := D_ℓ − 1;
11 |  for i := ℓ + 1 to n do K_i := K_i − 1;
12 return K';
```

Function Solve$(K', K_1, \ldots, K_n, D_1, \ldots, D_n)$.

Let us define ℓ as in line 3. If $D_i = 0$ for some $1 \le i \le \ell$, then y_i must be equal to 0, and thus this variable can be removed. Then the constraint C_i becomes $-\sum_{j \ne i} y_j + K_i \le z$. However, since $n \ge 2$, there is at least one other constraint $C_{i'}$ for $i' \ne i$. It has the form $y_{i'} - \sum_{j \ne i'} y_j + K_{i'} \le z$, which becomes $2y_{i'} - \sum_{j \ne i} y_j + K_{i'} \le z$ after y_i is removed. As $K_i = K_1$, we have $K_{i'} \ge K_i$, and due to non-negativity of $y_{i'}$, $C_{i'}$ gives a stronger lower bound on z than C_i, i.e. the constraint C_i can be removed. Thus lines 4–6 are correct. Also, note that in the recursive call the monotonicity assumption still holds, and the constant terms are still of the same parity.

Consequently, in lines 7–11 we may assume that $D_i > 0$ for $i \le \ell$. The arithmetic operations in these lines correspond to replacing y_ℓ with $y_\ell + 1 \in [\![1, D_\ell]\!]$. This operation is valid provided that the following claim holds.

Claim 2. *If $D_i > 0$ for $i \le \ell$, there exists an optimal solution $(z, y_1, \ldots, y_n)$ with $y_\ell \ge 1$.*

Proof. Let us take an optimal solution $(z, y_1, \ldots, y_n)$. If $y_\ell \ge 1$, we are done, so assume $y_\ell = 0$. We consider several cases. First, assume that there exists some i with $y_i \ge 1$. Replace y_i with $y_i - 1$ and set y_ℓ to 1. By $D_\ell \ge 1$, this is feasible with respect to variable ranges. The only constraints that change are C_ℓ and C_i. Set $S = \sum_j y_j$, which also does not change. Note that C_ℓ changes from $z \ge K_\ell - S$ to $z \ge K_\ell - S + 2$ and C_i changes from $z \ge K_i - S + 2y_i$ to $z \ge K_i - S + 2y_i - 2$. Thus C_i only weakens and the new C_ℓ is weaker than the original C_i, since $y_i \ge 1$ and $K_i \ge K_\ell$. Consequently, unless $y_1 = \ldots = y_n = 0$ is the only optimum, there exists an optimum with $y_\ell = 1$.

Thus, assume $y_1 = \ldots = y_n = 0$ is an optimum. Since $K' < K_n$, we have $z = K_n$ for the optimum. Now, we consider two cases. First, assume $\ell < n$. We claim that after setting $y_\ell := 1$, the solution is still feasible. The only constraints that become stronger are C_+ and C_ℓ. For the former we need to show that $K' + 1 \le K_n$, which is satisfied since $K' < K_n$. For the latter we require $K_\ell + 1 \le K_n$, which is also true since, by $\ell < n$, we have $K_\ell < K_n$.

Now, assume $\ell = n$. By $n \ge 2$ this means that $\ell > 1$. Therefore we can increment both y_1 and y_ℓ to 1 and observe that only C_+ became stronger, and we need to show that $K' + 2 \le K_n$. However, since K' and K_n are of the same parity, this is a consequence of $K' < K_n$. □

The claim proves that lines 7–11 are correct. Because K_ℓ and $K_{\ell+1}$ satisfied $K_\ell < K_{\ell+1}$, that is, $K_\ell + 2 \le K_{\ell+1}$, before these lines were executed and each K_i changed by at most 1, they now satisfy $K_\ell \le K_{\ell+1}$, so the monotonicity is also preserved. Also, observe that after operations in lines 7–11 all constant terms have the same parity.

Finally, observe that each iteration decreases $\sum_i (1 + D_i)$ and thus the procedure terminates.

To recover the variables' assignment, it suffices to keep record of which variables were set to 0 (lines 4–6) and for which the value was incremented (lines 7–11). □

Now we transform the `Solve` function into a more efficient version which we call `Solve2`. The latter simulates series of iterations of the while-loop of the former in single steps of its while-loop.

Lemma 12. *Functions* `Solve2` *and* `Solve` *are equivalent for all inputs corresponding to easy ILPs in the canonical form with $K_1 \le \ldots \le K_n$.*

Proof. Lines 1–6 of `Solve` and `Solve2` coincide, so we can restrict to inputs for which both functions reach line 7. Recall from the proof of Lemma 11 that this implies $n \ge 2$, $K' < K_n$ and $D_1, \ldots, D_\ell \ge 1$. We claim that lines 7–13 simulate the first ℓ iterations of the while-loop of `Solve` (or all $< \ell$ iterations if the loop execution is interrupted earlier). Note that the value of ℓ decrements by 1 in each of these iterations excluding the last one. Also, in each of these iterations the condition in line 5 is false, since in the previous iteration the respective D_i variables did not change. Thus, after an iteration with $\ell > 1$, the execution of the following $\ell - 1$ iterations of the while-loop may only be interrupted due to the $K' < K_n$ condition (line 2).

Note that, if $\ell < n$, then the difference $K_n - K'$ decreases by exactly 2 per iteration, as K_n is decremented while K' is incremented. Thus, if started with $\ell < n$, the loop fails to make ℓ steps only if $K_n - K' < 2\ell$, and this happens after

```
1  if n = 1 then return max(K', K_1);
2  while K' < K_n do
3  |  ℓ := max{i : K_i = K_1};
4  |  for i := 1 to ℓ do
5  |  |  if D_i = 0 then
6  |  |  |  return Solve2 (K', K_1, ..., K_{i-1}, K_{i+1}, ..., K_n, D_1, ..., D_{i-1}, D_{i+1}, ..., D_n);
7  |  if ℓ < n and K_n − K' < 2ℓ then return ½(K' + K_n);
8  |  if ℓ = n and K_n − K' < 2(n − 1) then return 1 + ½(K' + K_n);
9  |  K' := K' + ℓ;
10 |  for i := 1 to ℓ do
11 |  |  K_i := K_i + 2 − ℓ;
12 |  |  D_i := D_i − 1;
13 |  for i := ℓ + 1 to n do  K_i := K_i − ℓ;
14 return K';
```

Function Solve2($K', K_1, \ldots, K_n, D_1, \ldots, D_n$).

$\frac{1}{2}(K_n - K')$ iterations (due to same parity of constant terms this is an integer). Then, the value K', incremented at each iteration, is $\frac{1}{2}(K' + K_n)$ in terms of the original values.

On the other hand, if $\ell = n$, then in the first iteration $K_n - K'$ does not change, since K' and K_n are both incremented. The while-loop execution is interrupted before ℓ iterations only if $K_n - K' < 2(n-1)$, and this happens after $1 + \frac{1}{2}(K_n - K')$ iterations, when K', incremented at each iteration, is $1 + \frac{1}{2}(K' + K_n)$ in terms of the original values.

Consequently, lines 7 and 8 are correct, and if `Solve2` proceeds to line 9, we are guaranteed that `Solve` would execute at least ℓ iterations, with value ℓ decremented by 1 in each iteration. It is easy to see that lines 9–13 of `Solve2` simulate execution of lines 7–11 of `Solve` in these ℓ iterations, which corresponds to simultaneously replacing $y_1, \ldots, y_\ell$ with $y_1 + 1, \ldots, y_\ell + 1$.

Keeping track of variables' assignment here works the same as for the original `Solve` function. □

We perform the final improvement of the algorithm for solving easy ILP, which yields an efficient function `Solve3`. Again, what actually happens is that series of iterations of the while-loop from `Solve2` are now performed at once.

```
1  if n = 1 then return max(K', K_1);
2  while K' < K_n do
3  |  ℓ := max{i : K_i = K_1};
4  |  for i := 1 to ℓ do
5  |  |  if D_i = 0 then
6  |  |  |  return Solve3 (K', K_1, ..., K_{i-1}, K_{i+1}, ..., K_n, D_1, ..., D_{i-1}, D_{i+1}, ..., D_n);
7  |  if ℓ < n then
8  |  |  Δ := min(D_1, ..., D_ℓ, ½(K_{ℓ+1} − K_ℓ), ⌊1/(2ℓ) (K_n − K')⌋);
9  |  |  if Δ = 0 then return ½(K' + K_n);
10 |  else
11 |  |  Δ := min(D_1, ..., D_n, ⌊1/(2(n−1)) (K_n − K')⌋);
12 |  |  if Δ = 0 then return 1 + ½(K' + K_n);
13 |  K' := K' + Δ · ℓ;
14 |  for i := 1 to ℓ do
15 |  |  K_i := K_i + Δ(2 − ℓ);
16 |  |  D_i := D_i − Δ;
17 |  for i := ℓ + 1 to n do  K_i := K_i − Δ · ℓ;
18 return K';
```

Function Solve3($K', K_1, \ldots, K_n, D_1, \ldots, D_n$).

Lemma 13. *Functions* `Solve3` *and* `Solve2` *are equivalent for all inputs corresponding to easy ILPs in the canonical form with* $K_1 \le \ldots \le K_n$. *Moreover,* `Solve3` *runs in* $\mathcal{O}(n^2)$ *time (including recursive calls).*

Proof. As previously, lines 1–6 of `Solve2` and `Solve3` coincide, so we can restrict to inputs for which both functions reach line 7. This, in particular, means that $D_1, \ldots, D_\ell \ge 1$. Also, the definition of ℓ and the equal parity of constant terms imply that $K_{\ell+1} \ge K_\ell + 2$. Thus, if $\Delta = 0$ in line 9, then $K_n - K' < 2\ell$, and similarly in line 12, $\Delta = 0$ only if $K_n - K' < 2(n-1)$. These are exactly the conditions that `Solve2` checks in lines 7 and 8, and consequently `Solve3` simulates the execution of `Solve2` if one of these conditions is satisfied.

Consequently, in the following we may assume that $\Delta > 0$. Observe that during the execution of the while-loop in `Solve2`, the value ℓ may only increase. Indeed, the operations in lines 9–13 preserve the equality $K_1 = \ldots = K_\ell$ and decrease by 2 the difference $K_{\ell+1} - K_\ell$, which leads to an increase of ℓ after $\frac{1}{2}(K_{\ell+1} - K_\ell)$ steps. Also, in a single iteration, the difference $K_n - K'$ decreases by either 2ℓ if $\ell < n$ or $2(n-1)$ if $\ell = n$. Consequently, unless the execution is interrupted,

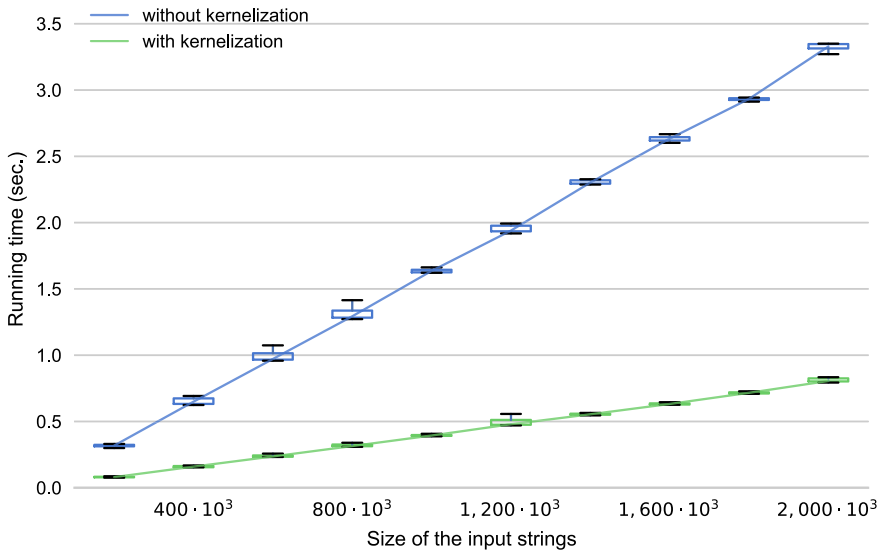

Fig. 4. The running times of our implementation of the efficient linear-time algorithm, without kernelization (upper graph) and with kernelization (lower graph). The graph presents the median of running times for 20 runs of the algorithm; the boxes span through first and third quartiles, and the error bars denote the extremal running times.

each value included in the minima in lines 8 and 11 decreases by exactly 1. Moreover, the execution is interrupted only if one of these values reaches 0, so we are guaranteed that Δ iterations will not be interrupted and can simulate them in bulk. This is what happens in lines 13–17 of `Solve3`, which correspond to lines 9–13 of `Solve2`. Thus, `Solve2` and `Solve3` are indeed equivalent. It remains to justify the quadratic running time of the latter.

Once again, it is easy to keep track of the variables' assignment, as in `Solve2`.

Observe that each iteration of the while-loop of `Solve3` can be executed in $\mathcal{O}(n)$ time. Thus, it suffices to show that, including the recursive calls, there are $\mathcal{O}(n)$ iterations in total. First, note that the case of $\ell = 1$ and $D_1 = 0$ can occur at most n times in total. Otherwise, we show that $2n - \ell$ decreases not less frequently than every second iteration. Note that the recursive call for $\ell > 1$ decrements both n and ℓ by one, and thus it also decrements $2n - \ell$. Therefore it suffices to consider an iteration which neither calls `Solve3` recursively nor terminates the whole procedure. Then lines 13–17 are executed. If Δ was chosen so that $\Delta = D_i$, the next iteration performs a recursive call, and thus $2n - \ell$ decreases after this iteration. On the other hand, if $\Delta = \frac{1}{2}(K_{\ell+1} - K_\ell)$, then after executing lines 13–17 it holds that $K_\ell = K_{\ell+1}$ and thus in the next iteration ℓ is increases, while n remains unchanged, so $2n - \ell$ decreases. Finally, if the previous conditions do not hold, then $\Delta = \left\lfloor \frac{K_n - K'}{2\min(\ell, n-1)} \right\rfloor$ and in the next iteration line 7 is reached again, but then in line 8 or 11 Δ is set to 0, and the procedure is terminated in line 9 or 12. This shows that it could not happen that three consecutive iterations (possibly from different recursive calls) do not decrease $2n - \ell$. This completes the proof that `Solve3` executes $\mathcal{O}(n)$ iterations in total. □

Lemmas 11–13 together show that `Solve3` correctly solves any easy ILP in the canonical form in $\mathcal{O}(n^2)$ time. Moreover, note that the resulting algorithm is elementary and has a small constant hidden in the $\mathcal{O}$ notation. Combined with the reduction of Lemma 10, we obtain the following result.

Theorem 3. *Any n-variate easy ILP can be solved in $\mathcal{O}(n^2)$ time.*

7. Conclusions

We have presented an $\mathcal{O}(\ell k \log k)$-time kernelization algorithm, which for any instance of the MSC problem computes an equivalent instance with $\ell' \leq k!$. Although for $k \leq 5$ this gives an instance with $\ell' \leq 120$, i.e. the kernel size is constant, solving it in a practically feasible time remains challenging. Therefore for $k \leq 5$ we have designed an efficient linear-time algorithm. As a consequence, we have obtained an efficient linear-time algorithm for the HSC problem with $k = 5$ binary sequences.

Our implementation of the algorithm for $k = 5$ is available at http://www.mimuw.edu.pl/~kociumaka/files/msc.cpp. As a preliminary test, we have run two versions of the algorithm—with and without the kernelization—on random input data with $\ell = 200\,000, 400\,000, \ldots, 2\,000\,000$ and $k = 5$. Fig. 4 presents the median of running times for 20 runs of the algorithm. The experiments were conducted on a MacBook Pro notebook (2.3 GHz Intel Core i7, 8 GB RAM).

Our approach for $k \leq 5$ constructs 20 ILP instances of an especially simple form, called here *easy ILP*, for which we craft a solution that is incomparably faster in practise than Lenstra's general ILP solver [18]. An interesting result would be to reduce the number of ILP instances constructed in this case.

A natural open question is whether our approach extends to larger values of k, e.g. to $k = 6$. It seems that while combinatorial properties could still be used to dramatically reduce the number of interval systems required to be processed, the resulting basic ILPs are not easy ILPs and have more variables. Consider, for example, a special case where for each column there are just 2 different values, each occurring 3 times. Then, there is a single interval sequence consistent with each $\mathbf{x}$ satisfying Observation 1; it also corresponds to taking a median in each column. Nevertheless, after simplifying the ILP there are still $\frac{1}{2}\binom{6}{3} = 10$ variables, and the ILP is not an easy ILP.

Acknowledgements

Tomasz Kociumaka is supported by Polish budget funds for science in 2013–2017 as a research project under the 'Diamond Grant' program, grant no. 0179/DIA/2013/42. Wojciech Rytter is supported by the Polish National Science Center, grant no. 2014/13/B/ST6/00770.

References

[1] Amihood Amir, Haim Paryenty, Liam Roditty, Configurations and minority in the string consensus problem, in: Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, Nivio Ziviani (Eds.), SPIRE, in: Lecture Notes in Computer Science, vol. 7608, Springer, 2012, pp. 42–53.
[2] Amihood Amir, Haim Paryenty, Liam Roditty, On the hardness of the consensus string problem, Inform. Process. Lett. 113 (10–11) (2013) 371–374.
[3] Alexandr Andoni, Piotr Indyk, Mihai Patrascu, On the optimality of the dimensionality reduction method, in: FOCS, IEEE Computer Society, 2006, pp. 449–458.
[4] Mihai Badoiu, Sariel Har-Peled, Piotr Indyk, Approximate clustering via core-sets, in: John H. Reif (Ed.), STOC, ACM, 2002, pp. 250–257.
[5] Christina Boucher, Daniel G. Brown, Stephane Durocher, On the structure of small motif recognition instances, in: Amihood Amir, Andrew Turpin, Alistair Moffat (Eds.), SPIRE, in: Lecture Notes in Computer Science, vol. 5280, Springer, 2008, pp. 269–281.
[6] Gérard D. Cohen, Iiro S. Honkala, Simon Litsyn, Patrick Solé, Long packing and covering codes, IEEE Trans. Inform. Theory 43 (5) (1997) 1617–1619.
[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 3rd ed., MIT Press, 2009.
[8] Kaspar Fischer, Bernd Gärtner, Martin Kutz, Fast smallest-enclosing-ball computation in high dimensions, in: Giuseppe Di Battista, Uri Zwick (Eds.), ESA, in: Lecture Notes in Computer Science, vol. 2832, Springer, 2003, pp. 630–641.
[9] Moti Frances, Ami Litman, On covering problems of codes, Theory Comput. Syst. 30 (2) (1997) 113–119.
[10] András Frank, Éva Tardos, An application of simultaneous diophantine approximation in combinatorial optimization, Combinatorica 7 (1) (1987) 49–65.
[11] Bernd Gärtner, Sven Schönherr, An efficient, exact, and generic quadratic programming solver for geometric optimization, in: Symposium on Computational Geometry, 2000, pp. 110–118.
[12] Ronald L. Graham, Neil J.A. Sloane, On the covering radius of codes, IEEE Trans. Inform. Theory 31 (3) (1985) 385–401.
[13] Jens Gramm, Rolf Niedermeier, Peter Rossmanith, Fixed-parameter algorithms for closest string and related problems, Algorithmica 37 (1) (2003) 25–42.
[14] Ravi Kannan, Minkowski's convex body theorem and integer programming, Math. Oper. Res. 12 (1987) 415–440.
[15] Tomasz Kociumaka, Jakub W. Pachocki, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, On the string consensus problem and the Manhattan sequence consensus problem, in: Edleno Silva de Moura, Maxime Crochemore (Eds.), SPIRE, in: Lecture Notes in Computer Science, vol. 8799, Springer, 2014, pp. 244–255.
[16] Piyush Kumar, Joseph S.B. Mitchell, E. Alper Yildirim, Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions, in: 5th Workshop on Algorithm Engineering and Experiments, 2003.
[17] J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, Louxin Zhang, Distinguishing string selection problems, in: Robert Endre Tarjan, Tandy Warnow (Eds.), SODA, ACM/SIAM, 1999, pp. 633–642.
[18] Hendrik W. Lenstra Jr., Integer programming with a fixed number of variables, Math. Oper. Res. 8 (1983) 538–548.
[19] Ming Li, Bin Ma, Lusheng Wang, On the closest string and substring problems, J. ACM 49 (2) (2002) 157–171.
[20] Daniel Lokshtanov, New Methods in Parameterized Algorithms and Complexity, PhD thesis, University of Bergen, 2009.
[21] Bin Ma, Xiaoming Sun, More efficient algorithms for closest string and substring problems, SIAM J. Comput. 39 (4) (2009) 1432–1443.
[22] Arya Mazumdar, Yury Polyanskiy, Barna Saha, On Chebyshev radius of a set in Hamming space and the closest string problem, in: ISIT, IEEE, 2013, pp. 1401–1405.
[23] J. Ritter, An efficient bounding sphere, in: A.S. Glassner (Ed.), Gems, Academic Press, Boston, MA, 1990.
[24] James Joseph Sylvester, A question in the geometry of situation, Quart. J. Pure Appl. Math. 1 (1857) 79.