Fast Witness Extraction Using a Decision Oracle

Andreas Björklund[1], Petteri Kaski[2] and Łukasz Kowalik[3] (speaker)

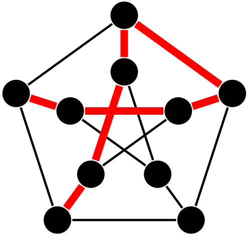
[1] Lund University (Sweden)
[2] Aalto University (Finland)
[3] University of Warsaw (Poland)

European Symposium on Algorithms (ESA 2014),
Wrocław, Poland
8th September 2014

LONGEST PATH problem

Problem

INPUT: directed/undirected graph G, integer k.
QUESTION: Does G contain a k-vertex path (shortly: k-path)?



Goal: solve it **in practice**

LONGEST PATH problem

Problem

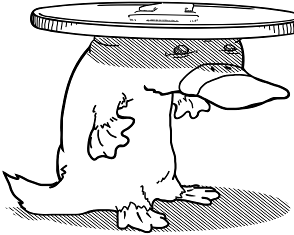
INPUT: directed/undirected graph G, integer k.
QUESTION: Does G contain a k-vertex path (shortly: k-path)?

A few facts

- NP-complete
- Monien 1985: $O(k!n^{O(1)})$ (first FPT algorithm: $O(f(k)n^{O(1)})$)
- Alon, Yuster, Zwick 1994: $5.44^k n^{O(1)}$ (color coding)
- Kneis et al. 2006, Chen et al. 2007: $4^k n^{O(1)}$ (divide-and-color)
- Koutis 2008: $2.83^k n^{O(1)}$ (group algebras, randomized)
- Williams 2009: $2^k n^{O(1)}$ (group algebras, randomized)
- Björklund, Husfeldt, Kaski, Koivisto 2010: $1.66^k n^{O(1)}$, undirected (polynomials over finite fields of characteristic two, randomized)
- Fomin, Lokshtanov, Saurabh 2013: $2.86^k n^{O(1)}$ (representative sets, deterministic)

A common theme

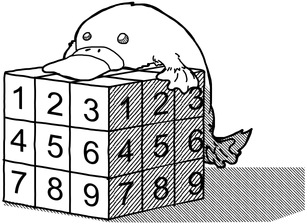
The currently fastest algorithms are **randomized** Monte Carlo with one-sided error.



- If YES is reported, it is always correct.
- If NO is reported, it is correct with some (constant) probability (**false negatives**).

Another common theme

The currently fastest algorithms use algebraic tools.



This means that

- they only solve **the decision problem**. (if YES is reported, there is no way to track the solution back from the computation.)
- use non-standard arithmetic, like addition/multiplication in finite fields $GF(2^q)$.

Motivating questions

- Are the algebraic algorithms for k-path practical?
- Which graph sizes (n) and path lengths (k) can we process?
- How can we **find** solutions ("witnesses") efficiently?
- Is randomization a problem?
- How should we implement $GF(2^q)$ finite field arithmetic?

Finding the solution by self-reducibility

The inclusion oracle

For any $A \subseteq E(G)$,

$$\textsc{Includes}(A) = \texttt{true} \text{ iff exists } k\text{-path } P \text{ such that } E(P) \subseteq A.$$

Observation

Using a decision algorithm we can implement the inclusion oracle.
(run the algortithm in the graph induced by edge set A)

Finding k-paths naively

1: $S \leftarrow E(G)$
2: **for** $e \in E(G)$ **do**
3:     **if** $\textsc{Includes}(S \setminus \{e\})$ **then**
4:         $S \leftarrow S \setminus \{e\}$
5: **return** S.

$|E|$ queries.
Expensive!

More problems

The same situation appears in state-of-art algorithms for a number of problems:

- 3-DIMENSIONAL MATCHING,
- k-PACKING,
- STEINER CYCLE (AKA K-CYCLE),
- RURAL POSTMAN,
- GRAPH MOTIF AND RELATED PROBLEMS,
- ...

Abstract problem

Problem

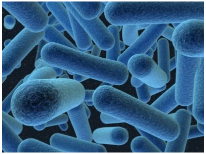
Given a set U,
find **any member** of an unknown **family** of subsets $\mathcal{S} \subseteq 2^U$,
using oracle $\textsc{Includes}$, where for any $A \subseteq U$,

$$\textsc{Includes}(A) = \texttt{true} \text{ iff exists } S \in \mathcal{S} \text{ such that } S \subseteq A.$$

A seemingly unrelated story...

- World War II, 1943.

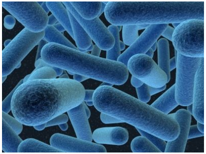
?



Group testing story

- World War II, 1943.

?

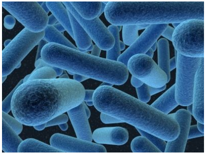


- testing a single recruit is expensive

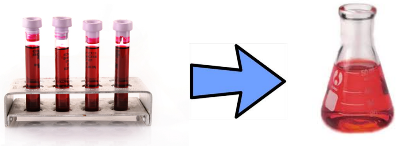
Group testing story

- World War II, 1943.

?



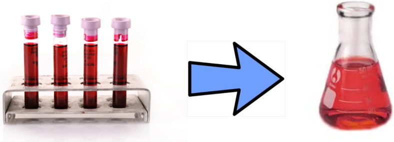
- testing a single recruit is expensive
- idea: mix blood of a group of soldiers



Group testing story

- World War II, 1943.

- testing a single recruit is expensive
- idea: mix blood of a group of soldiers



- how many tests do we need?

Group testing: model

Problem

Given a set U find an unknown subset $S \subseteq U$ using oracle $\textsc{Intersects}$, where for any $A \subseteq U$,

$$\textsc{Intersects}(A) = \texttt{true} \text{ iff } A \cap S \neq \emptyset.$$

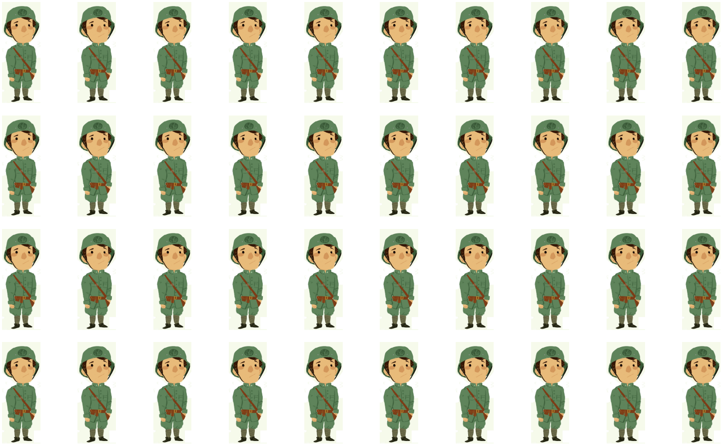
Another oracle

$$\textsc{CanDiscard}(A) = \texttt{true} \text{ iff we can safely discard } A.$$

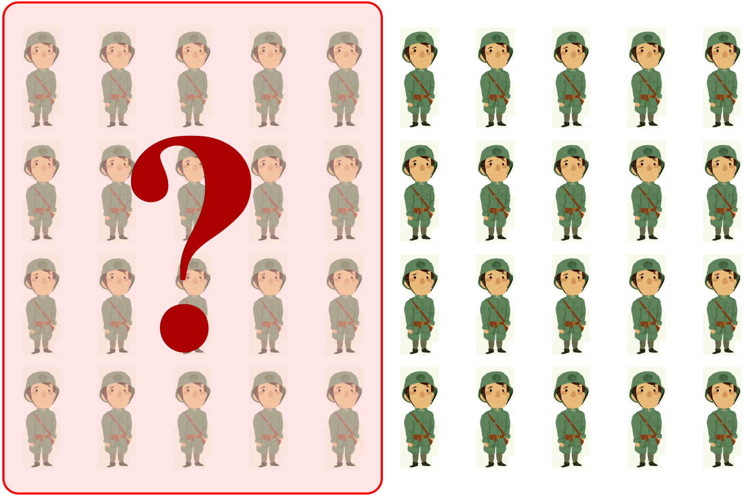
Observation

$\textsc{CanDiscard}(A) =$ **not** $\textsc{Intersects}(A)$.

Bisecting algorithm with CanDiscard oracle ($n = 40$)

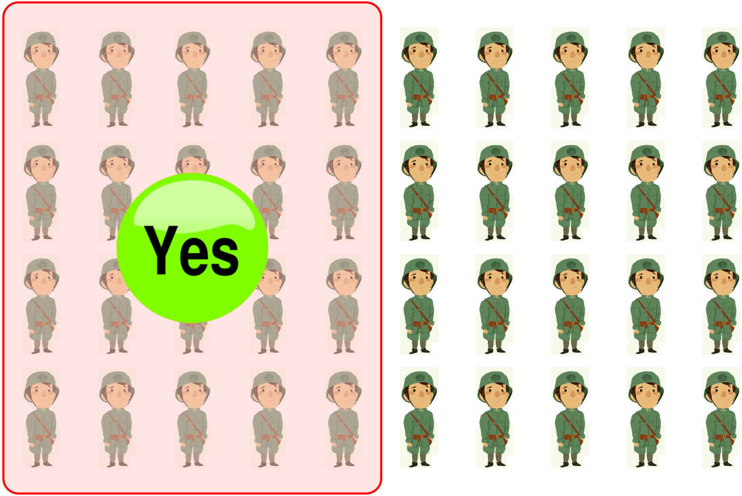


Bisecting algorithm with CanDiscard oracle ($n = 40$)



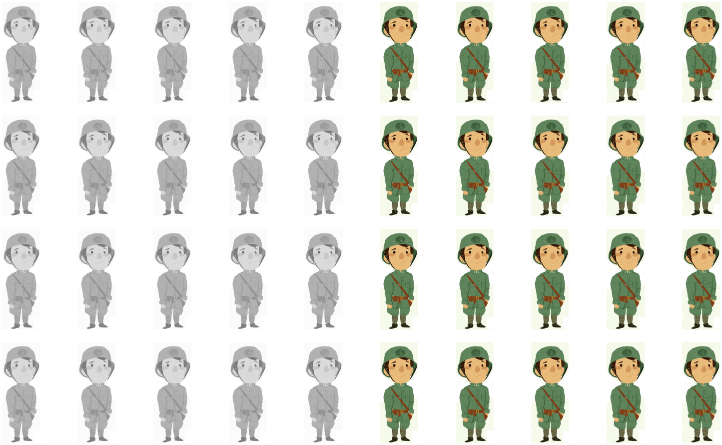
number of queries $= 1$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



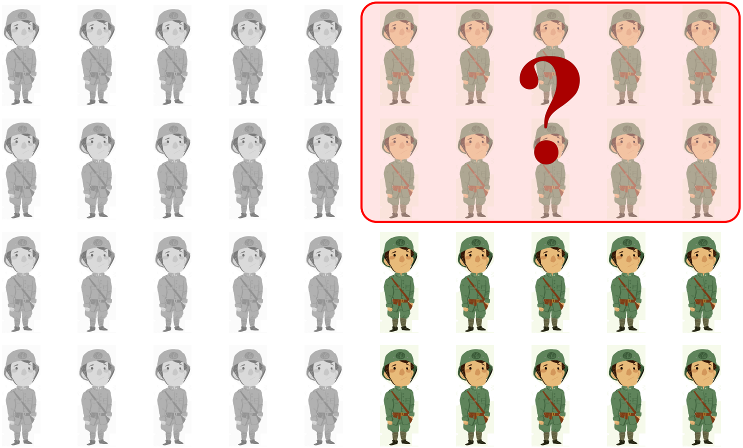
number of queries $= 1$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



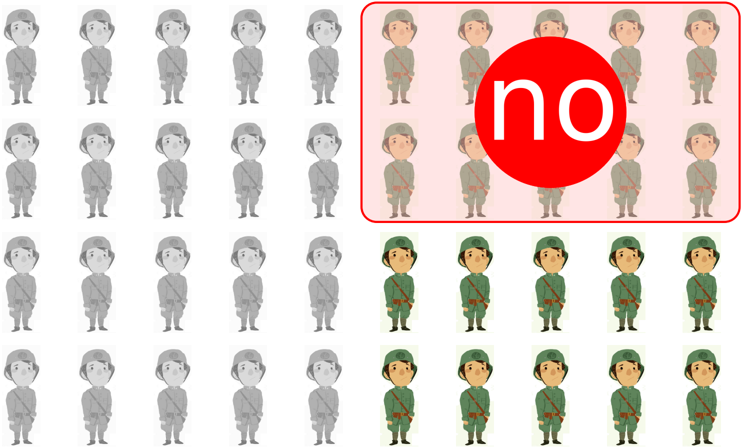
number of queries $= 1$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



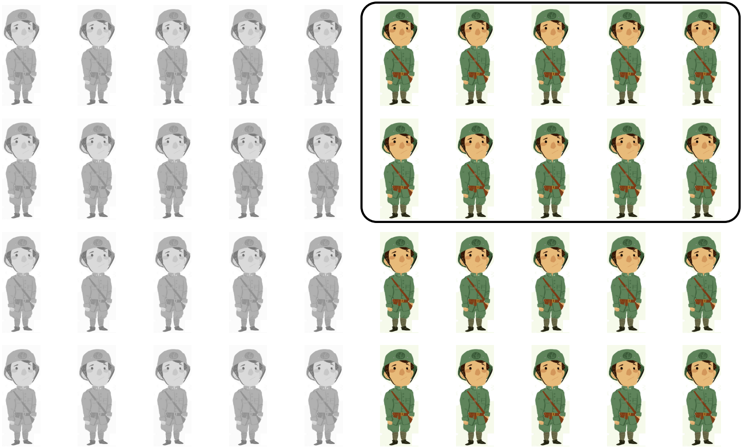
number of queries $= 2$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



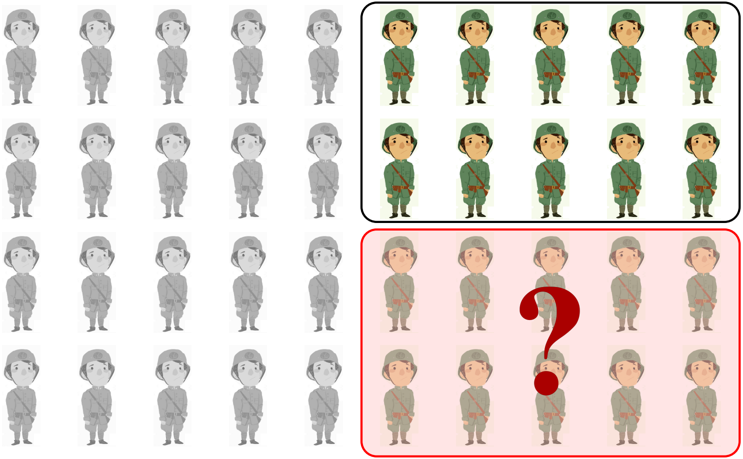
number of queries $= 2$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



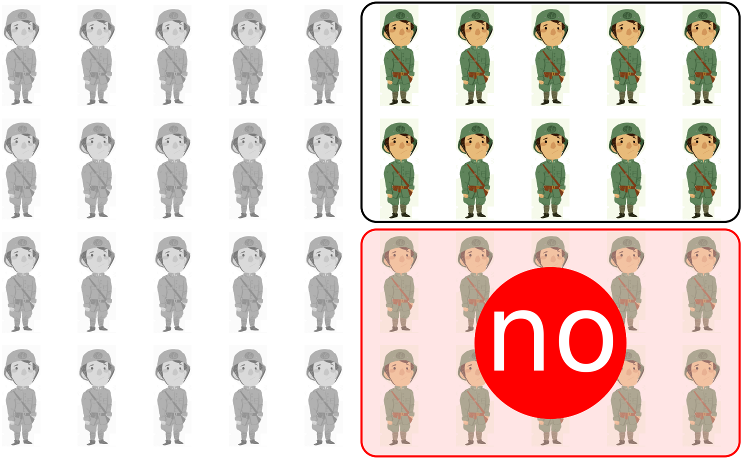
number of queries $= 2$

Bisecting algorithm with CANDISCARD oracle ($n = 40$)



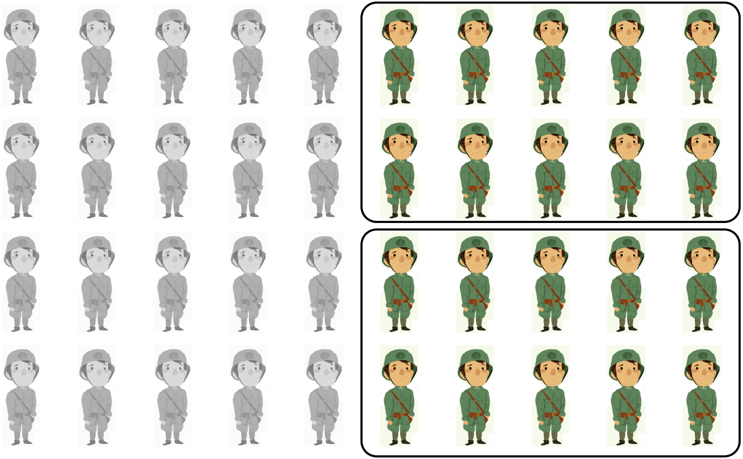
number of queries $= 3$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



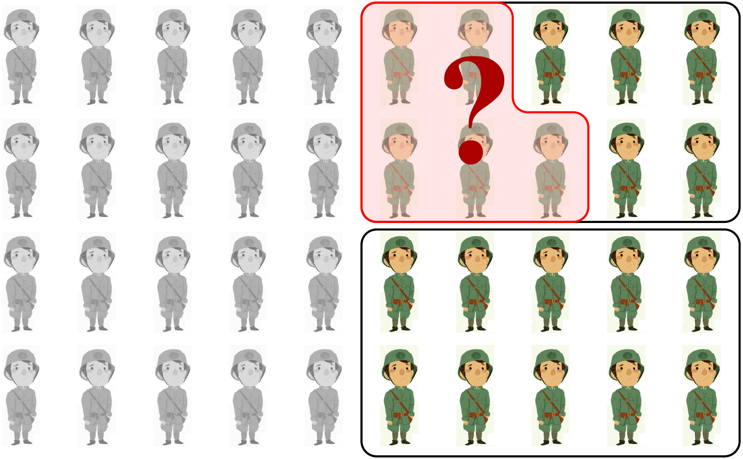
number of queries $= 3$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



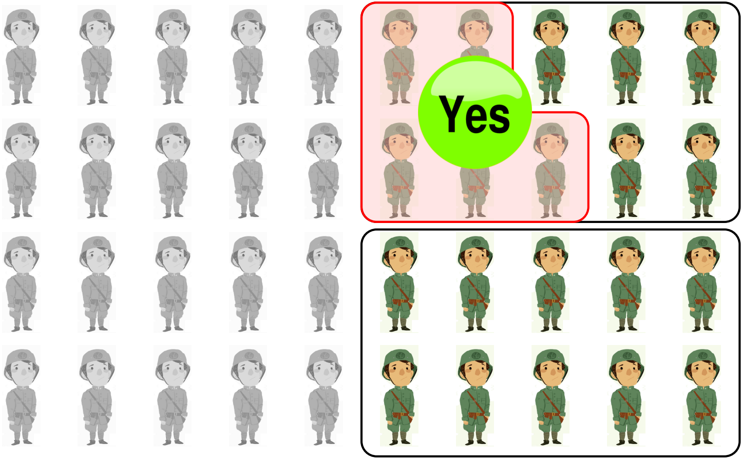
number of queries $= 3$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



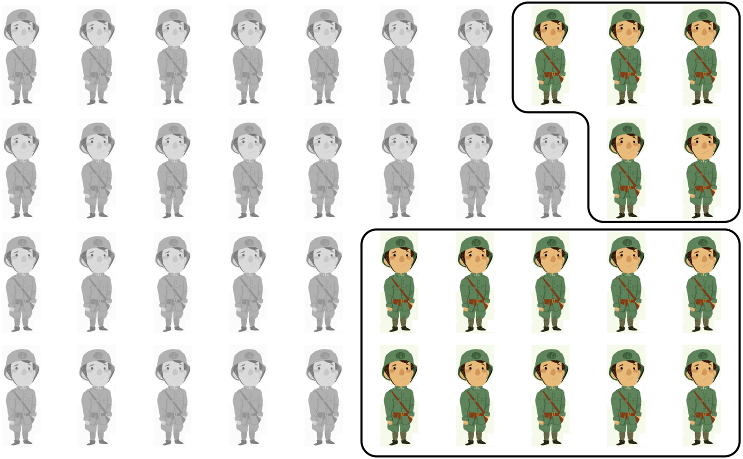
number of queries $= 4$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



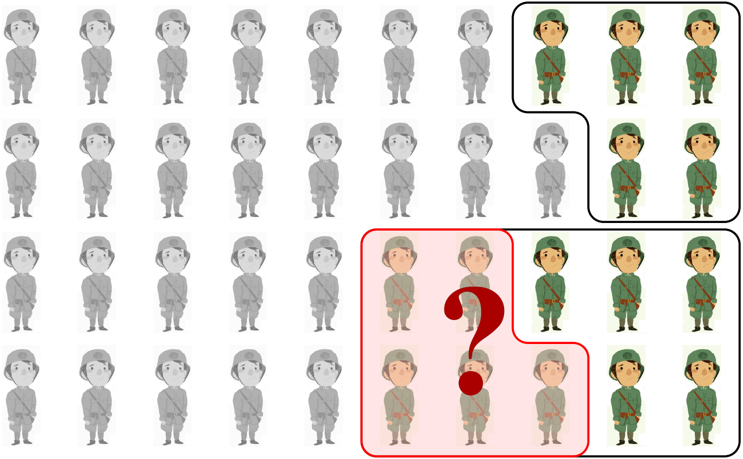
number of queries $= 4$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



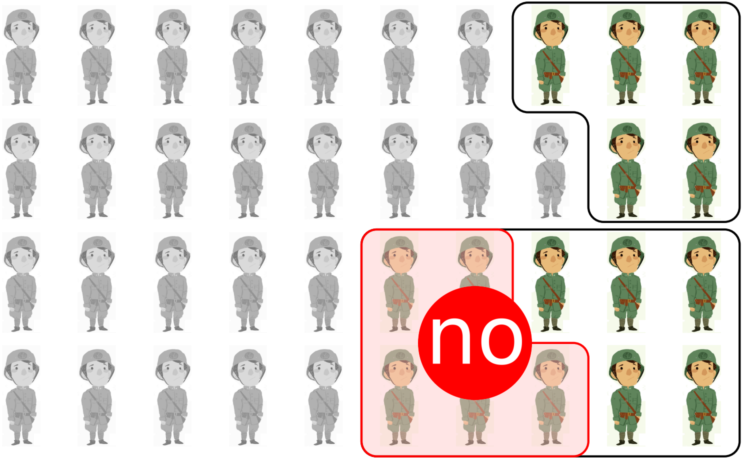
number of queries $= 4$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



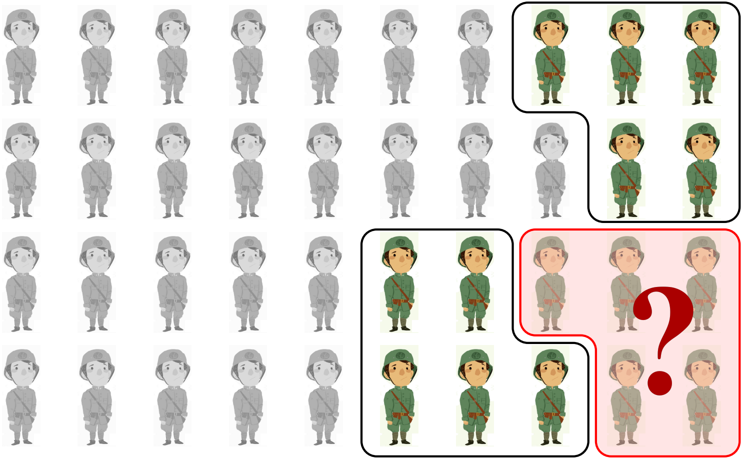
number of queries $= 5$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



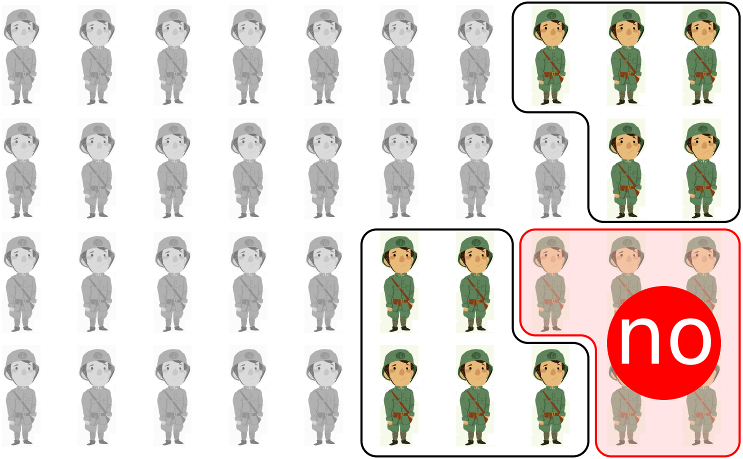
number of queries $= 5$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



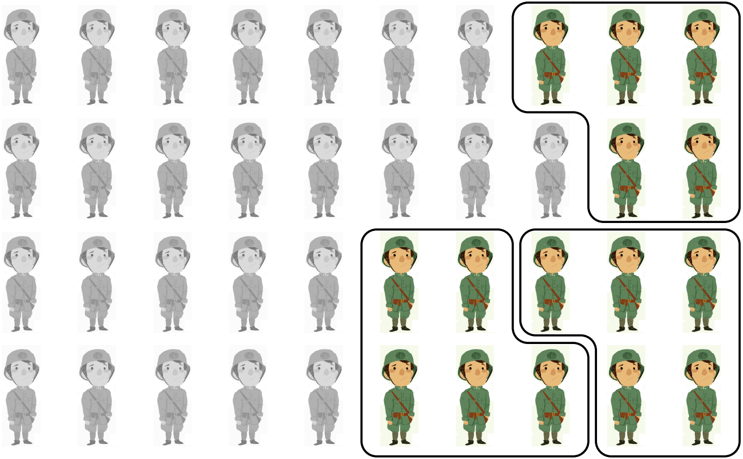
number of queries $= 6$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



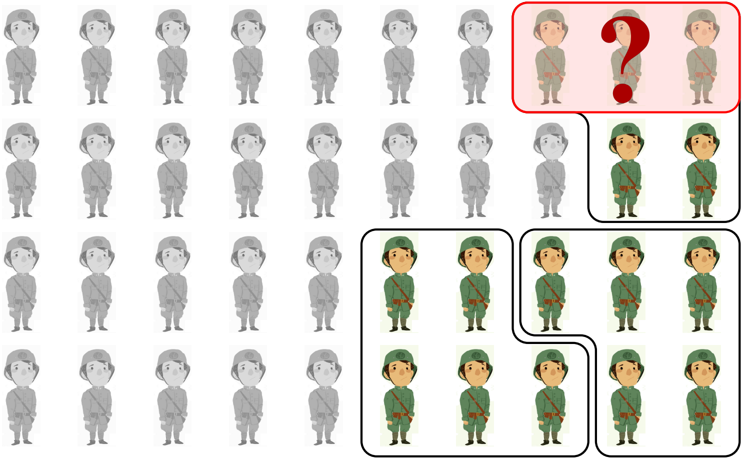
number of queries $= 6$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



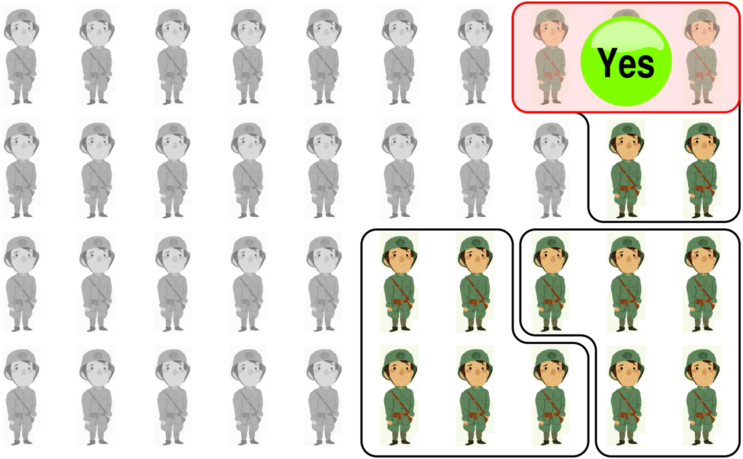
number of queries $= 6$

Bisecting algorithm with CANDISCARD oracle ($n = 40$)



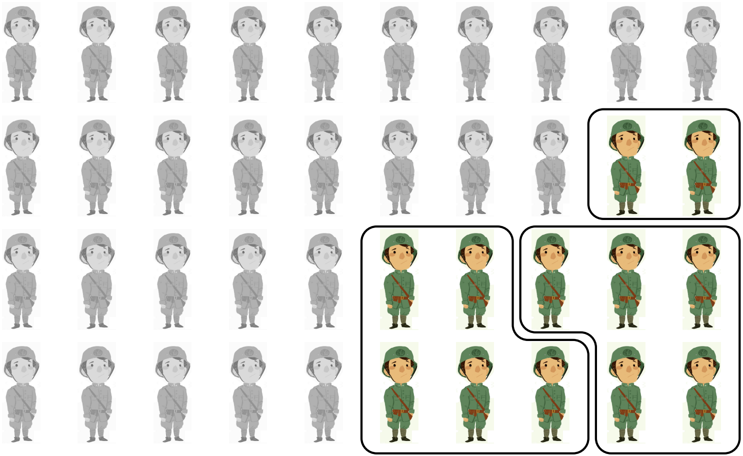
number of queries $= 7$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



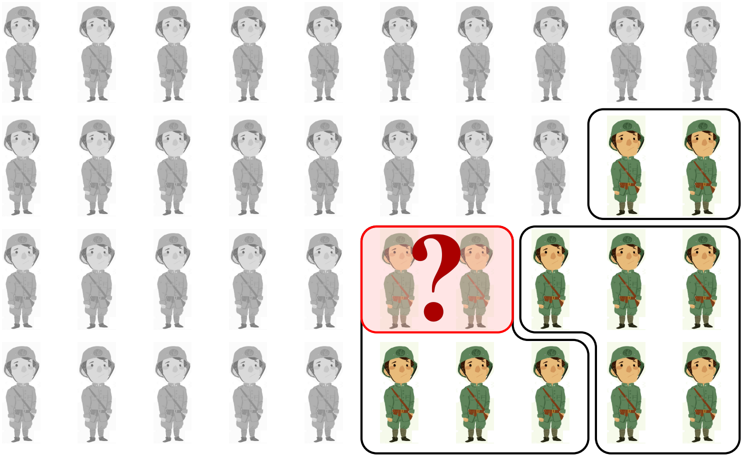
number of queries $= 7$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



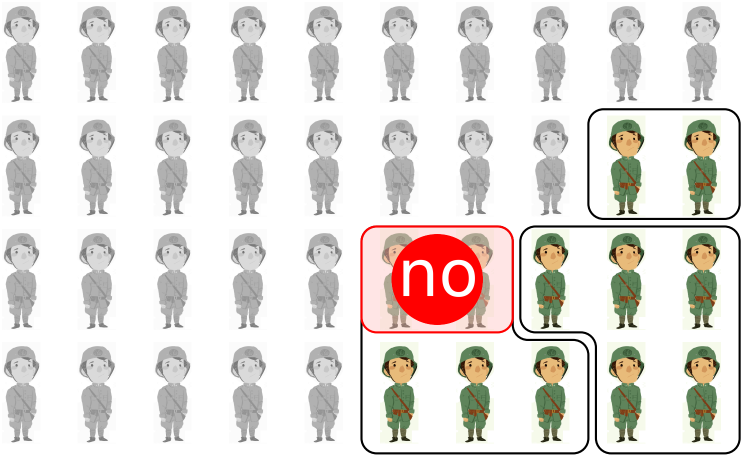
number of queries $= 7$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



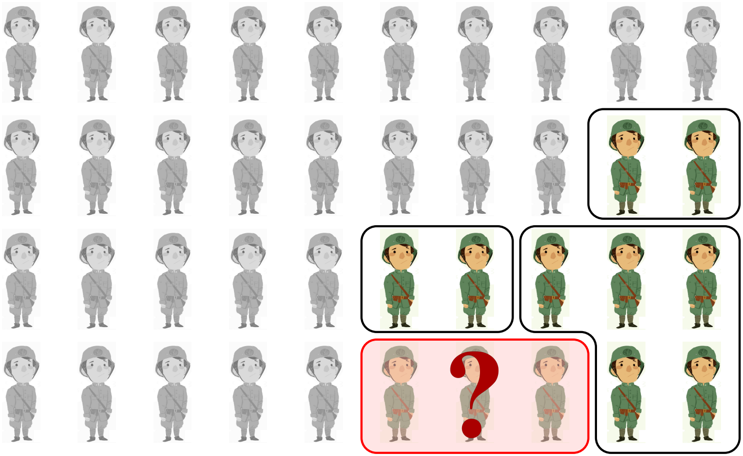
number of queries $= 8$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



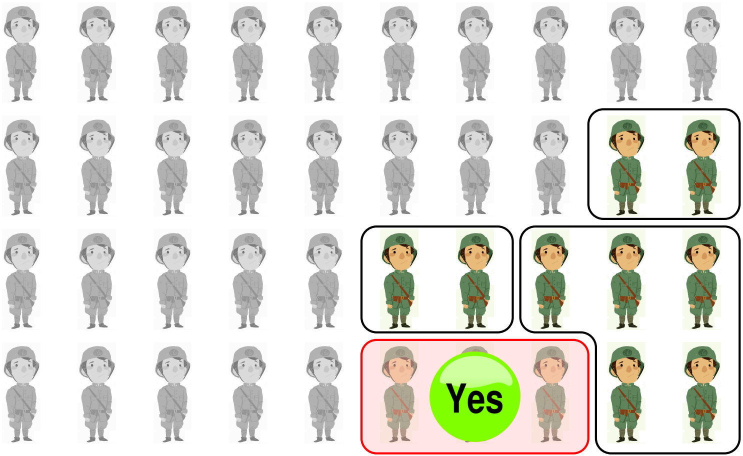
number of queries $= 8$

Bisecting algorithm with CANDISCARD oracle ($n = 40$)



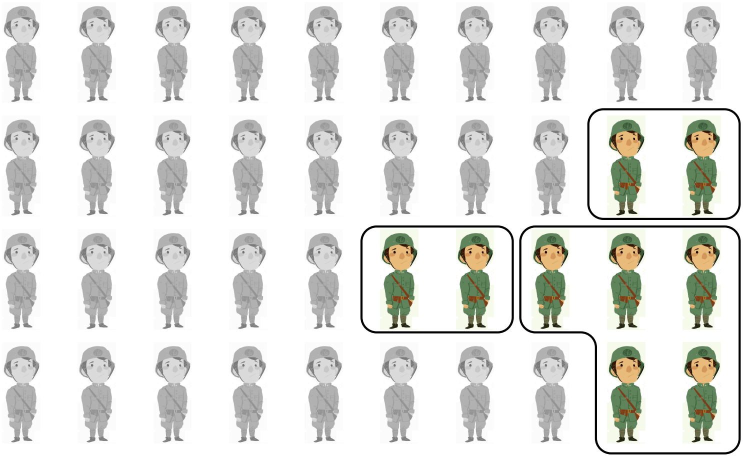
number of queries $= 9$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



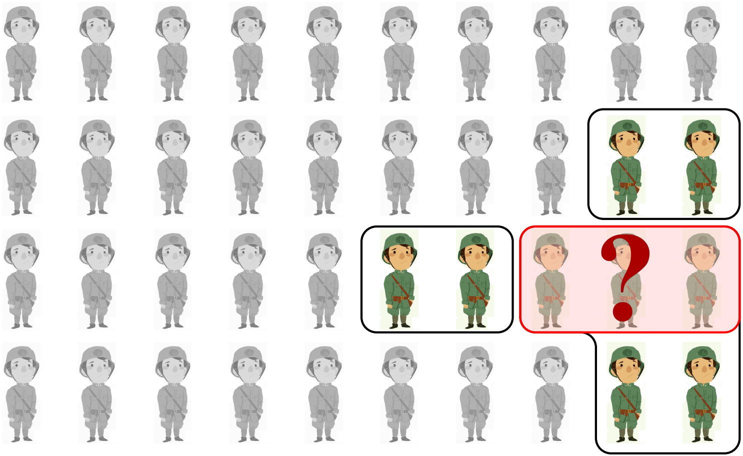
number of queries $= 9$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



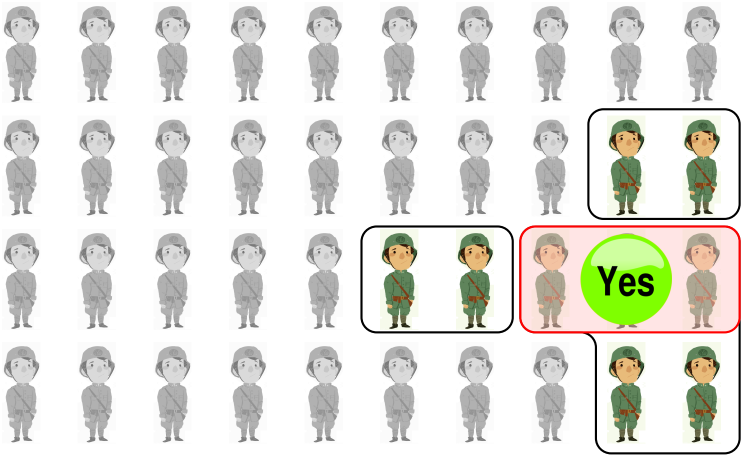
number of queries $= 9$

Bisecting algorithm with CANDISCARD oracle ($n = 40$)



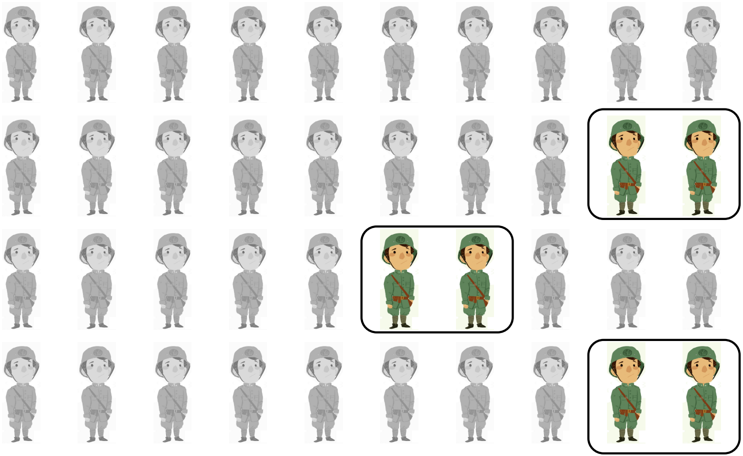
number of queries $= 10$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



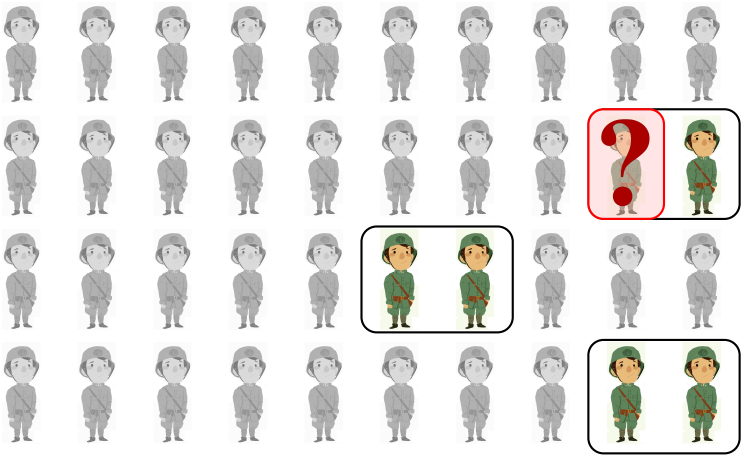
number of queries = 10

Bisecting algorithm with CanDiscard oracle ($n = 40$)



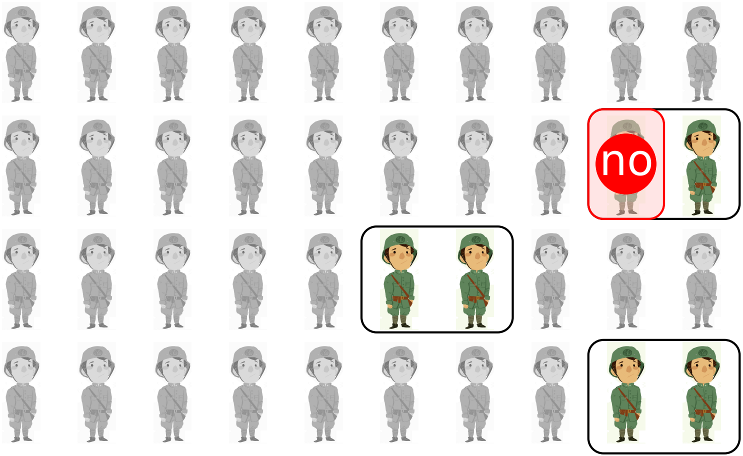
number of queries $= 10$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



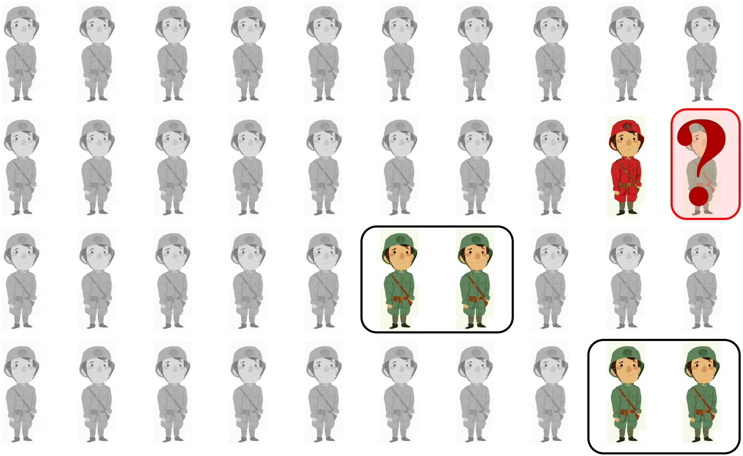
number of queries $= 11$

Bisecting algorithm with CANDISCARD oracle ($n = 40$)



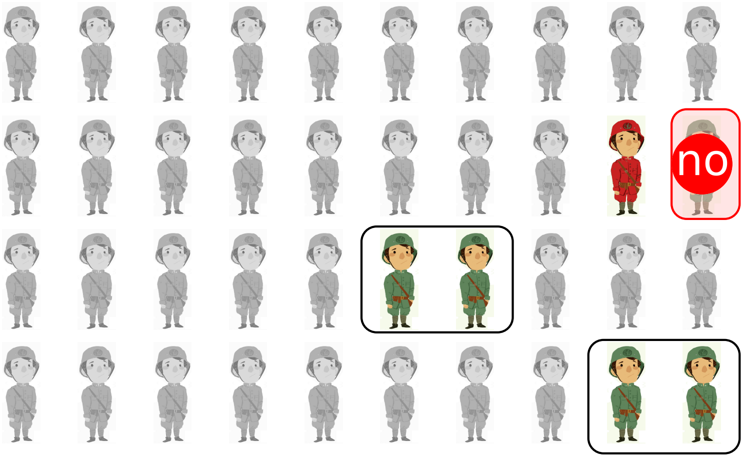
number of queries = 11

Bisecting algorithm with CanDiscard oracle ($n = 40$)



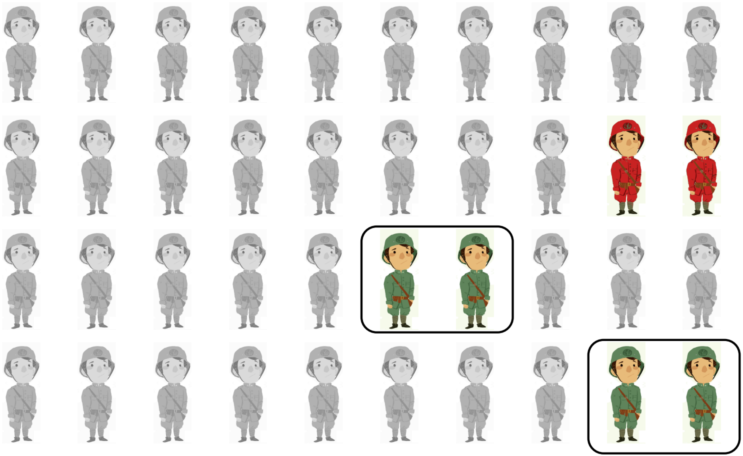
number of queries $= 12$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



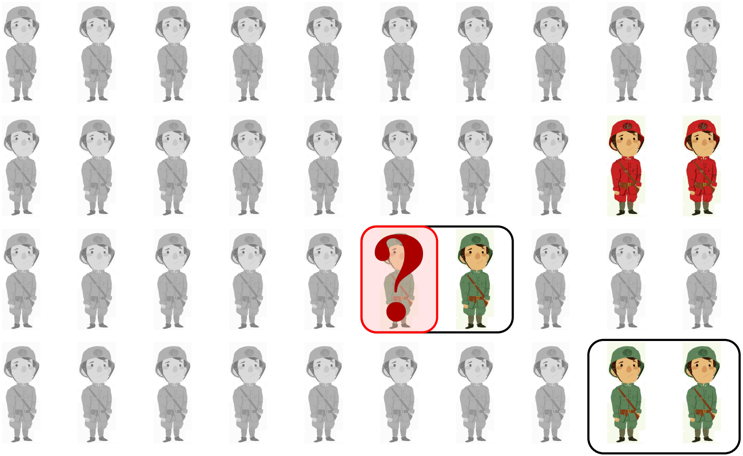
number of queries $= 12$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



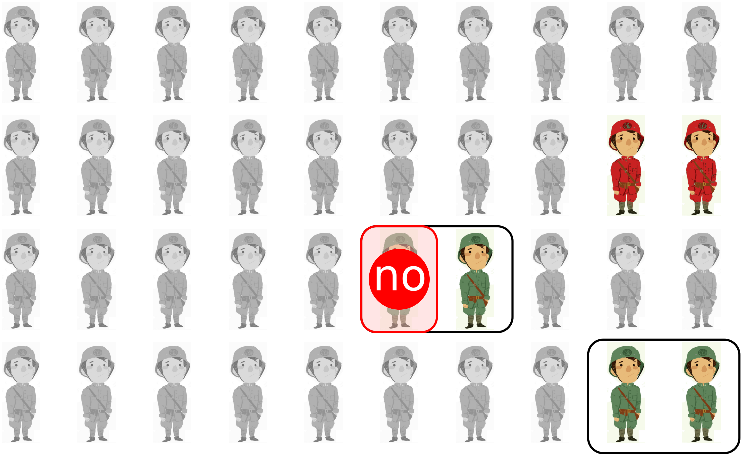
number of queries $= 12$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



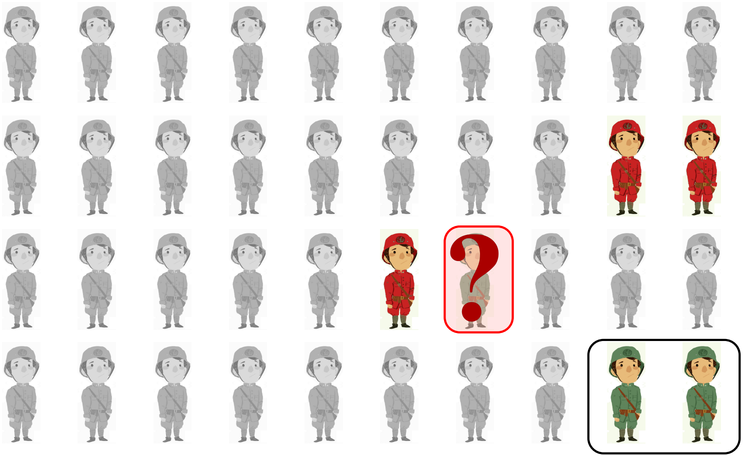
number of queries $= 13$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



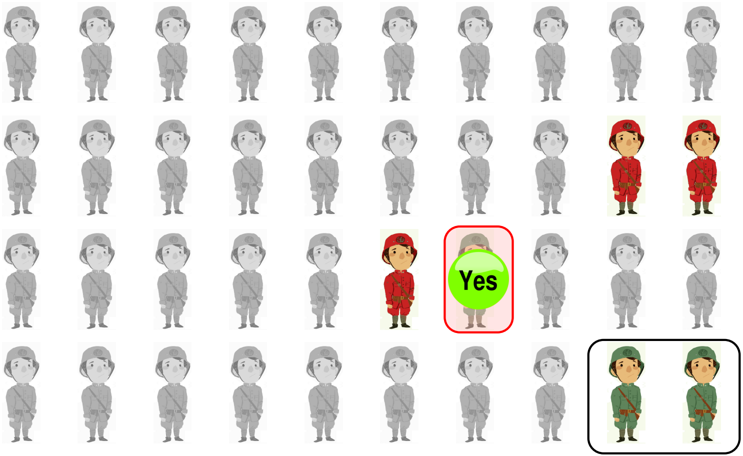
number of queries $= 13$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



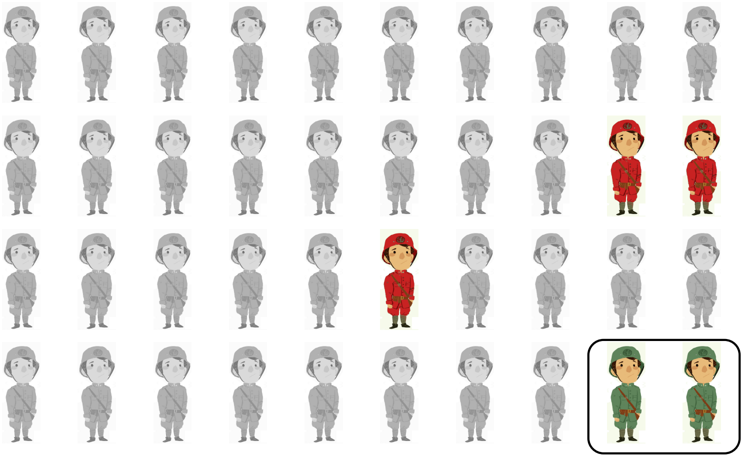
number of queries $= 14$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



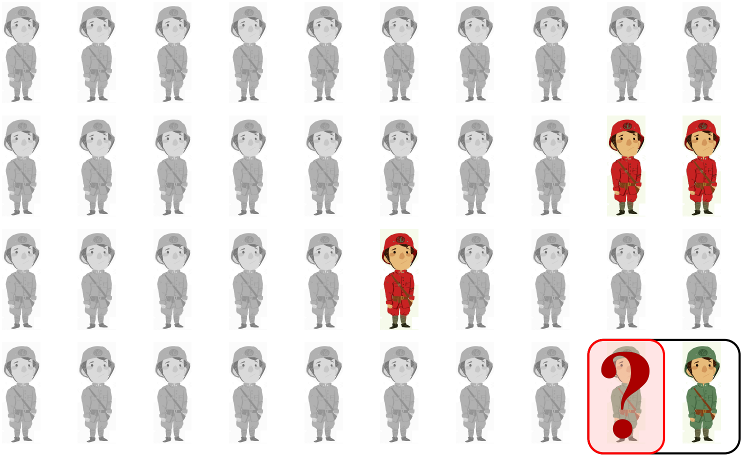
number of queries $= 14$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



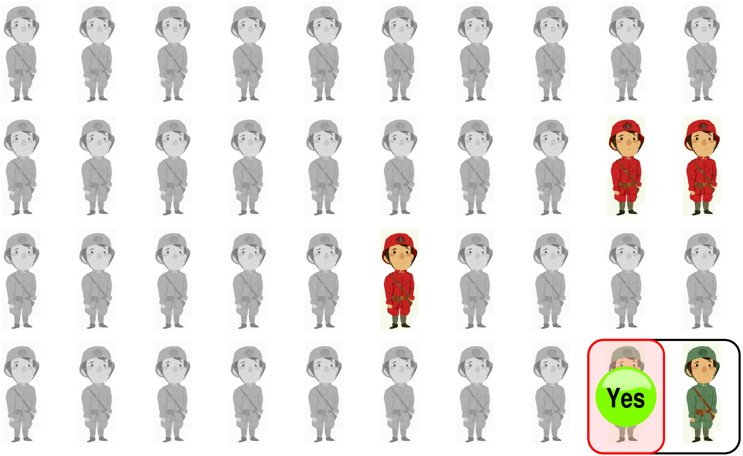
number of queries $= 14$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



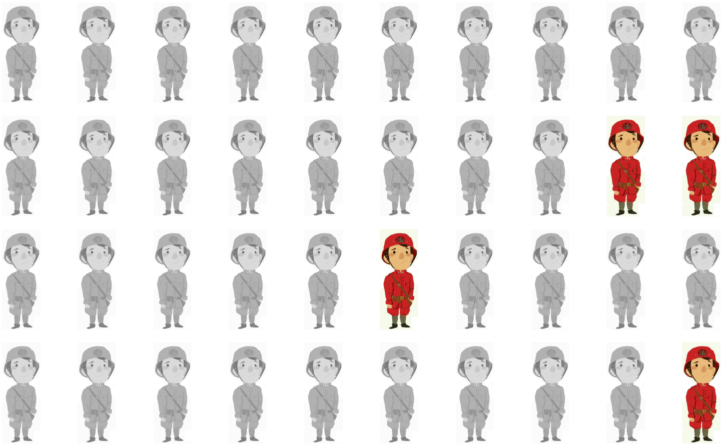
number of queries $= 15$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



number of queries $= 15$

Bisecting algorithm with CanDiscard oracle ($n = 40$)



number of queries $= 15$

Number of queries (think: n is big, k is small)

Theorem (Hwang, 70's)

Bisecting algorithm finds an unknown k-element set in n-element universe using $O(k\log(n/k))$ queries.

Theorem (Folklore)

Every deterministic (or randomized Las Vegas) algorithm performs $\Omega(k\log(n/k))$ queries in the worst case (in expectation).

Back to algebraic FPT algorithms

Abstract problem

Problem

Given a set U,
find **any member** of an unknown **family** of subsets $\mathcal{S} \subseteq 2^U$,
using oracle $\textsc{Includes}$, where for any $A \subseteq U$,

$$\textsc{Includes}(A) = \texttt{true} \text{ iff exists } S \in \mathcal{S} \text{ such that } S \subseteq A.$$

Abstract problem

Problem

Given a set U,
find **any member** of an unknown **family** of subsets $\mathcal{S} \subseteq 2^U$,
using oracle $\textsc{Includes}$, where for any $A \subseteq U$,

$$\textsc{Includes}(A) = \texttt{true} \text{ iff exists } S \in \mathcal{S} \text{ such that } S \subseteq A.$$

Observation

$\textsc{CanDiscard}(A) = \textsc{Includes}(U \setminus A)$.

Abstract problem

Problem

Given a set U,
find **any member** of an unknown **family** of subsets $\mathcal{S} \subseteq 2^U$,
using oracle $\textsc{Includes}$, where for any $A \subseteq U$,

$$\textsc{Includes}(A) = \texttt{true} \text{ iff exists } S \in \mathcal{S} \text{ such that } S \subseteq A.$$

Observation

$\textsc{CanDiscard}(A) = \textsc{Includes}(U \setminus A)$.

Corollary

Bisecting algorithm works here!
And finds a witness in $O(k \log(n/k))$ queries.

Abstract problem

Problem

Given a set U,
find **any member** of an unknown **family** of subsets $\mathcal{S} \subseteq 2^U$,
using oracle $\textsc{Includes}$, where for any $A \subseteq U$,

$$\textsc{Includes}(A) = \texttt{true} \text{ iff exists } S \in \mathcal{S} \text{ such that } S \subseteq A.$$

Observation

$\textsc{CanDiscard}(A) = \textsc{Includes}(U \setminus A)$.

Corollary

Bisecting algorithm works here!
And finds a witness in $O(k \log(n/k))$ queries.

Theorem (folklore)

Every algorithm needs $\Omega(k \log(n/k))$ queries.

Question

Does it work for our k-path problem?

Question

Does it work for our k-path problem?

Almost, we need to take care about false negatives.

Abstract problem: **randomized setting**

Problem

Given a set U,
find **any member** of an unknown **family** of subsets $\mathcal{S} \subseteq 2^U$,
using oracle $\textsc{Includes}$, where for any $A \subseteq U$,

- If $\textsc{Includes}(A) = \texttt{true}$ then exists $S \in \mathcal{S}$ such that $S \subseteq A$.
- If $\textsc{Includes}(A) = \texttt{false}$ then with probability at least $1/2$ there is no $S \in \mathcal{S}$ such that $S \subseteq A$.

Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Witness extraction in randomized setting

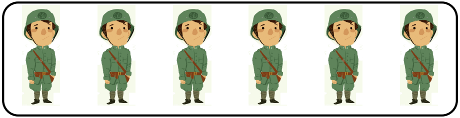
Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



Witness extraction in randomized setting

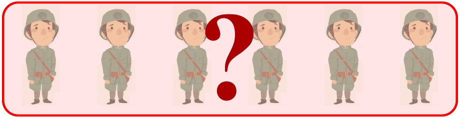
Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



Witness extraction in randomized setting

Implementing CanDiscard(A)

Return Includes($U \setminus A$).

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:

Witness extraction in randomized setting

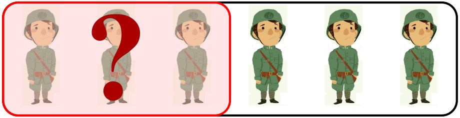
Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



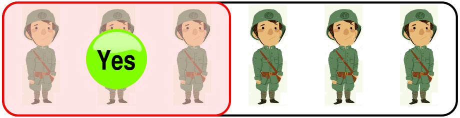
Witness extraction in randomized setting

Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!
Errors can be fixed in future:



Witness extraction in randomized setting

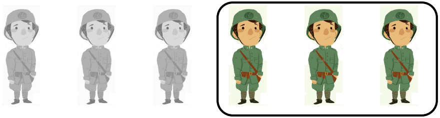
Implementing CanDiscard(A)

Return Includes($U \setminus A$).

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



Witness extraction in randomized setting

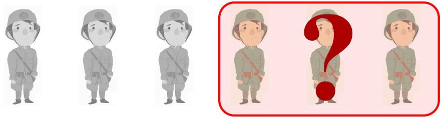
Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



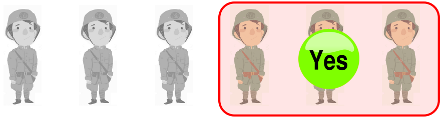
Implementing $\textsc{CanDiscard}(A)$

Return $\textsc{Includes}(U \setminus A)$.

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



Witness extraction in randomized setting

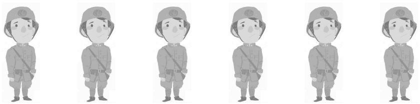
Implementing CanDiscard(A)

Return Includes($U \setminus A$).

- If `true`, **for sure** we can discard A.
- If `false`, keep A (error probability at most $1/2$).

Note: we never discard elements erroneously!

Errors can be fixed in future:



Witness extraction in randomized setting

Algorithm

1. Run Bisecting Algorithm (with no change!)
2. While the remaining set is not a witness, run the naive algorithm.

Theorem (our result)

If there is a witness, it is (always) found within $O(k \log n)$ queries in expectation.

Conjecture

This is optimal. (Best lower bound: $\Omega(k \log(n/k))$.)

Implementation of finding k-paths

How should we implement $GF(2^q)$ arithmetic?

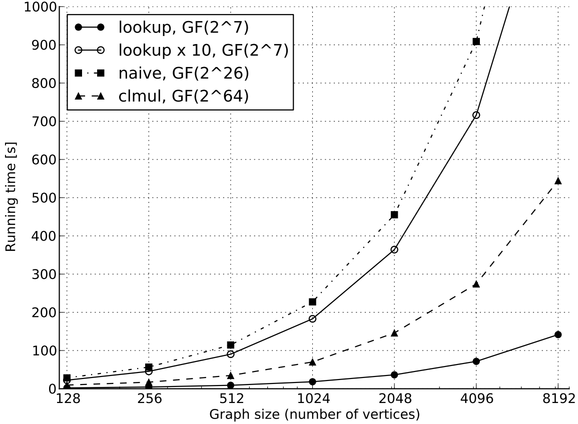
- For correctness, we need $q \geq 2 + \lceil \log_2 k \rceil$.
- We can assume $k \leq 30$ (otherwise too slow) $\Rightarrow q \geq 7$.
- Large $q \;\Rightarrow\;$ slow arithmetic, low error probability.
- Small $q \;\Rightarrow\;$ fast arithmetic, big error probability.

Possible implementations:

- **naive**: do it yourself — slow.
- **clmul**: using carry-less multiplication PCLMULQDQ (new Intel, AMD processors) — quite fast, $q = 64$ (very low error probability).
- **lookup**: whole multiplication table stored in cache — extremely fast, $q = 7$ (quite high error probability).

Comparision of $GF(2^q)$ implementations

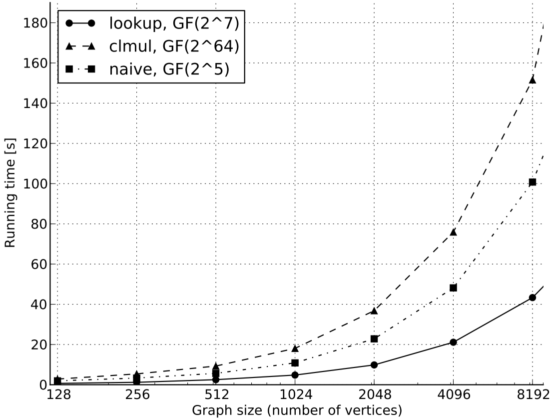
For **decision** algorithm (**single** query): PCLMULQDQ wins.



Path size $k = 16$. No solutions in the instance.

Comparision of $GF(2^q)$ implementations

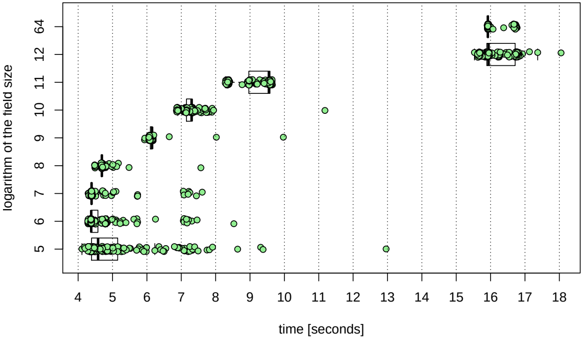
For **extraction** algorithm (**multiple** queries): lookup table wins.



Path size $k = 12$. No solutions in the instance.

Comparision of $GF(2^q)$ implementations

Optimizing the implementation and the size of the field:



Statistics for 200 runs of the extraction algorithm ($n = 1000$, $k = 12$) using lookup implementation for $q = 5, \ldots, 12$ and clmul implementation ($q = 64$).

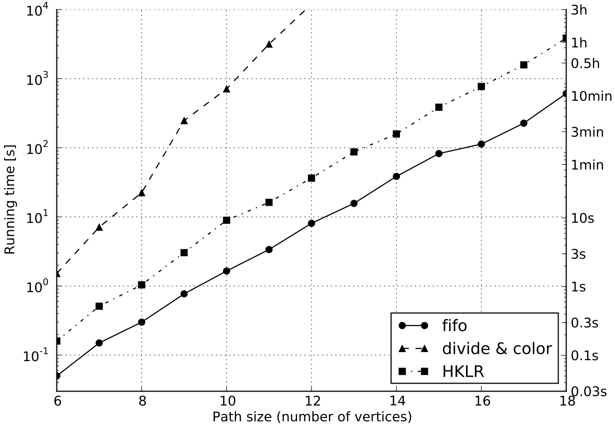
Comparision of algorithms

We have implemented:

- **divide-and-color** algorithm for finding k-path
- an $O(2^k k|E|)$-time decision algorithm D for k-path (based on Bjorklund et al paper),
- witness extraction using our modified bisection and D (denoted **fifo**)
- Independently, Hassidim, Keller, Lewenstein, and Roditty (WADS'13) found an extraction algorithm based on similar ideas.

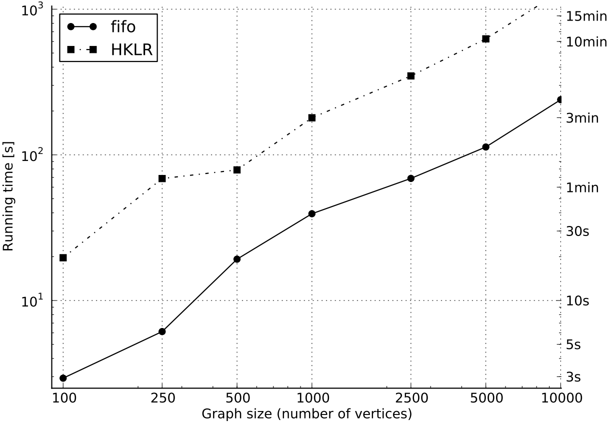
  we implemented witness extraction using their method and D (denoted **HKLR**)

Comparision of algorithms (2.53-GHz Intel Xeon CPU)



$n = 1000$, size of the path $k = 6, \ldots, 18$, exactly one witness.

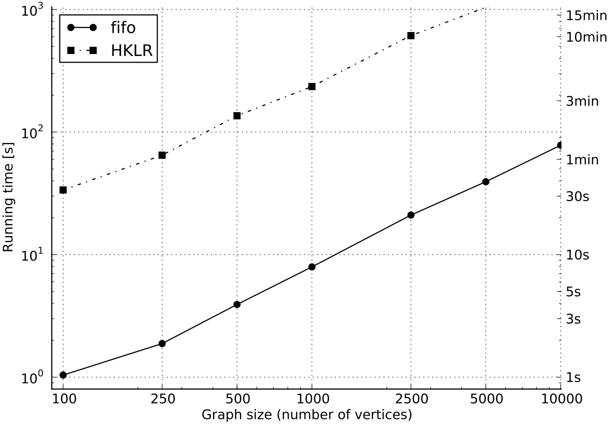
Comparision of algorithms (2.53-GHz Intel Xeon CPU)



$k = 14$, size of the path $n = 100, \ldots, 10000$, exactly one witness.

Comparision of algorithms (2.53-GHz Intel Xeon CPU)

If many witnesses, they are easier to find – fifo exploits it nicely.



$k = 15$, size of the path $n = 100, \ldots, 10000$, $O(n^2)$ witnesses.

Conclusions

- A fast algorithm for extrating witnesses
- Multiplication in $GF(2^q)$ should be implemented using lookup table
- We can find 14-vertex paths in 10 000-vertex graphs below 5 minutes.

Thank you!

The presentation contains some figures of Felix Reidl from a book

Cygan, Fomin, Marx, Kowalik, Lokshtanov, Pilipczuk, Pilipczuk, Saurabh
Parameterized Algorithms

(to appear in early 2015)