Theoretical and practical algorithms for CSP parameterized by the number of variables with value-independent constraints

Gregory Gutin
Royal Holloway, University of London,UK

Bergen, PC&PC, 5th August 2019

Outline

Outline

Workflow Satisfiability Problem (WSP)

WSP is of interest in Access Control (best papers in DBSec and SACMAT 2015 were on WSP). WSP has many variations.

Workflow Satisfiability Problem (WSP)

WSP is of interest in Access Control (best papers in DBSec and SACMAT 2015 were on WSP). WSP has many variations.

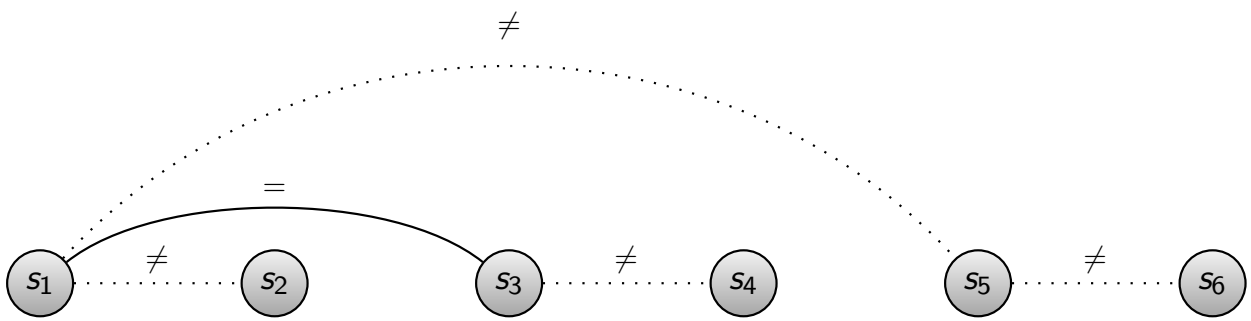
(basic) WSP instance: a set $U = \{u_1, u_2, u_3, u_4\}$ of users and a set S of six steps as follows.

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Workflow Satisfiability Problem (WSP), cont'd

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Constraints and authorizations:

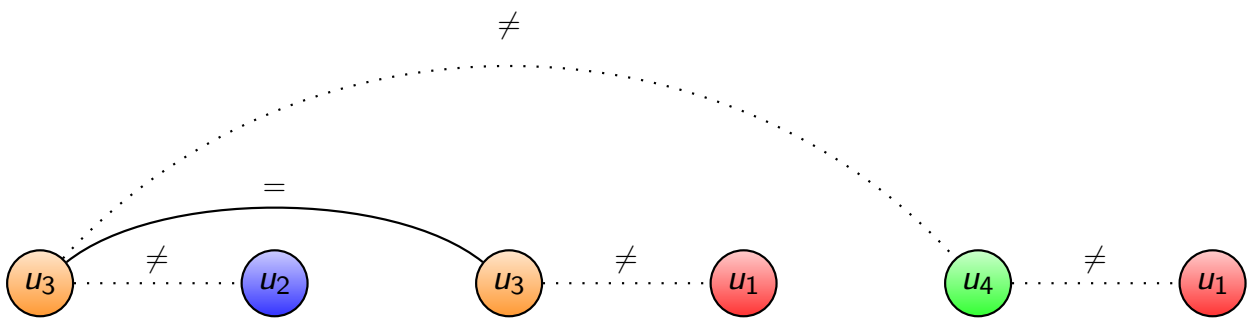


Is there a solution?

Workflow Satisfiability Problem (WSP), cont'd

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Constraints and authorizations:



Authorized Users

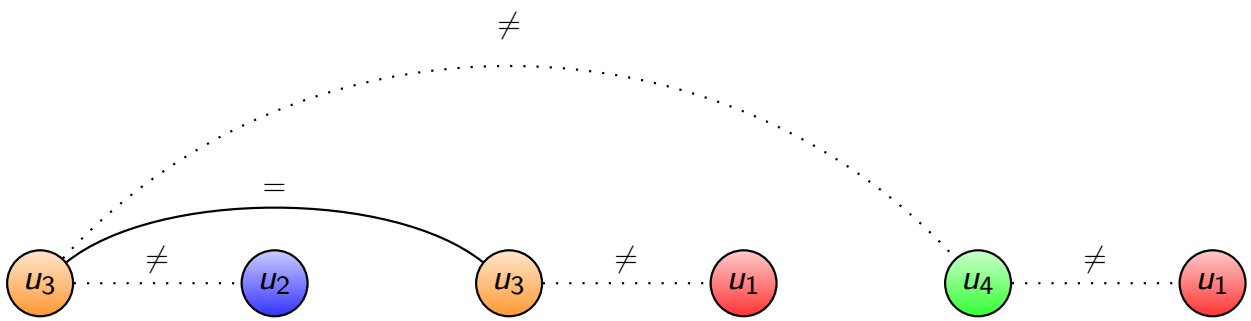
u_3	u_2	u_3	u_1	u_1	u_1
u_1	u_3	u_4	u_3	u_3	u_4
u_2				u_4	

Is there a solution? Yes.

Workflow Satisfiability Problem (WSP), cont'd

s_1 create purchase order	s_2 approve purchase order
s_3 sign goods received note	s_4 countersign goods received note
s_5 create payment	s_6 approve payment

Constraints and authorizations:



Authorized Users

u_3	u_2	u_3	u_1	u_1	u_1
u_1	u_3	u_4	u_3	u_3	u_4
u_2				u_4	

Is there a solution? Yes. It's an instance of List Coloring.

Definition of basic WSP

- Instance $W = (S, U, C)$, where S and U are sets of steps and users, C is a family of non-unary constraints with variables S and domain U. Unary constraints (authorizations): $A(s_i) = U'_i \subseteq U$.

Definition of basic WSP

- Instance $W = (S, U, C)$, where S and U are sets of steps and users, C is a family of non-unary constraints with variables S and domain U. Unary constraints (authorizations): $A(s_i) = U'_i \subseteq U$.
- $\pi : S \to U$ is a plan; π is valid if π satisfies all constraints C. W is satisfiable if there is a valid plan π.

Definition of basic WSP

- Instance $W = (S, U, C)$, where S and U are sets of steps and users, C is a family of non-unary constraints with variables S and domain U. Unary constraints (authorizations): $A(s_i) = U'_i \subseteq U$.
- $\pi : S \to U$ is a plan; π is valid if π satisfies all constraints C. W is satisfiable if there is a valid plan π.
- Assumption: we can check, in poly time, whether a plan satisfies a constraint.

Definition of basic WSP

- Instance $W = (S, U, C)$, where S and U are sets of steps and users, C is a family of non-unary constraints with variables S and domain U. Unary constraints (authorizations): $A(s_i) = U'_i \subseteq U$.
- $\pi : S \to U$ is a plan; π is valid if π satisfies all constraints C. W is satisfiable if there is a valid plan π.
- Assumption: we can check, in poly time, whether a plan satisfies a constraint.
- basic WSP = CSP, where CSP variables = WSP steps, CSP values = WSP users. We'll talk about WSP.

Parameterized WSP

- In practice, $k = |S| \ll n = |U|$ and k is relatively small.

Parameterized WSP

- In practice, $k = |S| \ll n = |U|$ and k is relatively small.
- Wang and Li (ACM Trans. Inf. Syst. Secur., 2010): parameterized WSP by k (k-WSP) and proved k-WSP is W[1]-hard and in W[2]. Unknown yet if k-WSP is W[1]-complete or W[2]-complete.

Parameterized WSP

- In practice, $k = |S| \ll n = |U|$ and k is relatively small.
- Wang and Li (ACM Trans. Inf. Syst. Secur., 2010): parameterized WSP by k (k-WSP) and proved k-WSP is W[1]-hard and in W[2]. Unknown yet if k-WSP is W[1]-complete or W[2]-complete.
- Wang and Li (2010) considered constraints $(s_i, T, =)$ and $(s_i, T, \neq)$, where $T \subseteq S$. Restricted to these non-unary constraints, k-WSP is FPT. Unary constraints are arbitrary.

Parameterized WSP

- In practice, $k = |S| \ll n = |U|$ and k is relatively small.
- Wang and Li (ACM Trans. Inf. Syst. Secur., 2010): parameterized WSP by k (k-WSP) and proved k-WSP is W[1]-hard and in W[2]. Unknown yet if k-WSP is W[1]-complete or W[2]-complete.
- Wang and Li (2010) considered constraints $(s_i, T, =)$ and $(s_i, T, \neq)$, where $T \subseteq S$. Restricted to these non-unary constraints, k-WSP is FPT. Unary constraints are arbitrary.
- Improved and generalized by Crampton, GG and Yeo (ACM Trans. Inf. Syst. Secur., 2013) to regular (undefined here) non-unary constraints with $O^*(2^k)$-algorithm. (Incl.Excl.)

Outline

User-Independent Constraints

- Cohen, Crampton, Gagarin, GG and Jones (J. AI Res., 2014): constraint c is user-independent (UI) if for every plan $\pi : S \to U$ satisfying c and permutation θ of U, plan $\pi' : S \to U$ also satisfies c, where for each $s \in S$, $\pi'(s) = \theta(\pi(s))$.

User-Independent Constraints

- Cohen, Crampton, Gagarin, GG and Jones (J. AI Res., 2014): constraint c is user-independent (UI) if for every plan $\pi : S \to U$ satisfying c and permutation θ of U, plan $\pi' : S \to U$ also satisfies c, where for each $s \in S$, $\pi'(s) = \theta(\pi(s))$.
- Counting constraints $(r, Q, \leq)$ and $(r, Q, \geq)$ are UI. $(r, Q, \leq)$ is satisfied by π if $|\pi(Q)| \leq r$ (confidentiality) and $(r, Q, \geq)$ is satisfied by π if $|\pi(Q)| \geq r$ (diversity). Example: AllDiff(Q)=$(|Q|, Q, \geq)$.

User-Independent Constraints

- Cohen, Crampton, Gagarin, GG and Jones (J. AI Res., 2014): constraint c is user-independent (UI) if for every plan $\pi : S \to U$ satisfying c and permutation θ of U, plan $\pi' : S \to U$ also satisfies c, where for each $s \in S$, $\pi'(s) = \theta(\pi(s))$.
- Counting constraints $(r, Q, \leq)$ and $(r, Q, \geq)$ are UI. $(r, Q, \leq)$ is satisfied by π if $|\pi(Q)| \leq r$ (confidentiality) and $(r, Q, \geq)$ is satisfied by π if $|\pi(Q)| \geq r$ (diversity). Example: AllDiff(Q)=$(|Q|, Q, \geq)$.
- All the non-unary constraints defined in the 2004 American National Standards Institute Role Based Access Control standard are UI.

Results for Non-unary User-Independent Constraints (Unary Constraints are Arbitrary)

Cohen, Crampton, Gagarin, GG and Jones (JAIR 2014):

- k-WSP with UI non-unary constraints can be solved in time $O^*(2^{k \log k})$.

Results for Non-unary User-Independent Constraints (Unary Constraints are Arbitrary)

Cohen, Crampton, Gagarin, GG and Jones (JAIR 2014):

- k-WSP with UI non-unary constraints can be solved in time $O^*(2^{k \log k})$.
- Unless ETH fails, no $O^*(2^{o(k \log k)})$-time algorithm.

Results for Non-unary User-Independent Constraints (Unary Constraints are Arbitrary)

Cohen, Crampton, Gagarin, GG and Jones (JAIR 2014):

- k-WSP with UI non-unary constraints can be solved in time $O^*(2^{k \log k})$.
- Unless ETH fails, no $O^*(2^{o(k \log k)})$-time algorithm.
- BFS-like user-iterative algorithm of exponential space.

Results for Non-unary User-Independent Constraints (Unary Constraints are Arbitrary), con'd

- Karapetyan, Gagarin and GG (FAW 2015): a backtracking $O^*(2^{k \log k})$-time poly-space algorithm for k-WSP with UI non-unary constraints, much faster in practice. Further improved by Karapetyan, Parks, GG and Gagarin (JAIR 2019, ta). [arXiv:1604.05636; Will be discussed in detail.]

Results for Non-unary User-Independent Constraints (Unary Constraints are Arbitrary), con'd

- Karapetyan, Gagarin and GG (FAW 2015): a backtracking $O^*(2^{k \log k})$-time poly-space algorithm for k-WSP with UI non-unary constraints, much faster in practice. Further improved by Karapetyan, Parks, GG and Gagarin (JAIR 2019, ta). [arXiv:1604.05636; Will be discussed in detail.]

- Gutin and Wahlström (IPL 2016): k-WSP with UI non-unary constraints cannot be solved in time $O^*(c^{k \log k})$ for any $c < 2$ unless SETH fails.

Outline

Basic Algorithm and its Improved Version

- Basic algorithm operates by generating complete patterns (defined later) instead of plans to significantly reduce search space (to an FPT size space instead of XP exponential).

Basic Algorithm and its Improved Version

- Basic algorithm operates by generating complete patterns (defined later) instead of plans to significantly reduce search space (to an FPT size space instead of XP exponential).
- Complete patterns are representatives of plans.

Basic Algorithm and its Improved Version

- Basic algorithm operates by generating complete patterns (defined later) instead of plans to significantly reduce search space (to an FPT size space instead of XP exponential).
- Complete patterns are representatives of plans.
- Basic Algorithm has two parts: (1) checking UI non-unary constraints, (2) checking unary constraints.

Basic Algorithm and its Improved Version

- Basic algorithm operates by generating complete patterns (defined later) instead of plans to significantly reduce search space (to an FPT size space instead of XP exponential).
- Complete patterns are representatives of plans.
- Basic Algorithm has two parts: (1) checking UI non-unary constraints, (2) checking unary constraints.
- Improved Algorithm operates by generating partial patterns (corresponding to functions from subsets of S to U) gradually leading to complete patters unless some constraint is unsatisfied in which case backtracking.

Basic Algorithm: Part 1

- Since non-unary constraints are UI and up to k users can be used for steps $s_1, \ldots, s_k$, introduce complete patterns: tuples $(p_1, \ldots, p_k)$, where $p_i \in [k]$, if $p_i = p_j$ then s_i, s_j must be assigned the same user, if $p_i \neq p_j$, then different users.

Basic Algorithm: Part 1

- Since non-unary constraints are UI and up to k users can be used for steps $s_1, \ldots, s_k$, introduce complete patterns: tuples $(p_1, \ldots, p_k)$, where $p_i \in [k]$, if $p_i = p_j$ then s_i, s_j must be assigned the same user, if $p_i \neq p_j$, then different users.
- Look for complete patterns satisfying all non-unary constraints.

Basic Algorithm: Part 1

- Since non-unary constraints are UI and up to k users can be used for steps $s_1, \ldots, s_k$, introduce complete patterns: tuples $(p_1, \ldots, p_k)$, where $p_i \in [k]$, if $p_i = p_j$ then s_i, s_j must be assigned the same user, if $p_i \neq p_j$, then different users.
- Look for complete patterns satisfying all non-unary constraints.
- Generate all non-equivalent complete patterns by enumerating only minimum ones. Natural order, i.e., $(1, 2, 1) < (2, 1, 2)$.

Basic Algorithm: Part 1

- Since non-unary constraints are UI and up to k users can be used for steps $s_1, \ldots, s_k$, introduce complete patterns: tuples $(p_1, \ldots, p_k)$, where $p_i \in [k]$, if $p_i = p_j$ then s_i, s_j must be assigned the same user, if $p_i \neq p_j$, then different users.
- Look for complete patterns satisfying all non-unary constraints.
- Generate all non-equivalent complete patterns by enumerating only minimum ones. Natural order, i.e., $(1, 2, 1) < (2, 1, 2)$.
- Min patterns can be viewed as partitions of $\{1, \ldots, k\}$, where each block corresponds to a different user. The number of partitions of $\{1, \ldots, k\}$ is k'th Bell number $B_k \leq k!$.

Basic Algorithm: Part 1

- Since non-unary constraints are UI and up to k users can be used for steps $s_1, \ldots, s_k$, introduce complete patterns: tuples $(p_1, \ldots, p_k)$, where $p_i \in [k]$, if $p_i = p_j$ then s_i, s_j must be assigned the same user, if $p_i \neq p_j$, then different users.
- Look for complete patterns satisfying all non-unary constraints.
- Generate all non-equivalent complete patterns by enumerating only minimum ones. Natural order, i.e., $(1, 2, 1) < (2, 1, 2)$.
- Min patterns can be viewed as partitions of $\{1, \ldots, k\}$, where each block corresponds to a different user. The number of partitions of $\{1, \ldots, k\}$ is k'th Bell number $B_k \leq k!$.
- Bell numbers have been studied by mathematicians since the 19th century, but their roots go back to medieval Japan.

Basic Algorithm: Part 2

- For each complete pattern p (as partition) $S_1 \uplus \cdots \uplus S_t = S$ construct a bipartite graph B with bipartition $(\{S_1, \ldots, S_t\}, U)$ such that $S_i u \in E(B)$ if u is authorised for all steps in S_i.

Basic Algorithm: Part 2

- For each complete pattern p (as partition) $S_1 \uplus \cdots \uplus S_t = S$ construct a bipartite graph B with bipartition $(\{S_1, \ldots, S_t\}, U)$ such that $S_i u \in E(B)$ if u is authorised for all steps in S_i.
- Pattern p satisfies all unary constraints iff B has a matching with t edges.

Basic Algorithm: Part 2

- For each complete pattern p (as partition) $S_1 \uplus \cdots \uplus S_t = S$ construct a bipartite graph B with bipartition $(\{S_1, \ldots, S_t\}, U)$ such that $S_i u \in E(B)$ if u is authorised for all steps in S_i.
- Pattern p satisfies all unary constraints iff B has a matching with t edges.
- Runtime of Basic Algorithm: $O^*(B_k) = O^*(2^{k \log k})$.

Basic Algorithm: Part 2

- For each complete pattern p (as partition) $S_1 \uplus \cdots \uplus S_t = S$ construct a bipartite graph B with bipartition $(\{S_1, \ldots, S_t\}, U)$ such that $S_i u \in E(B)$ if u is authorised for all steps in S_i.
- Pattern p satisfies all unary constraints iff B has a matching with t edges.
- Runtime of Basic Algorithm: $O^*(B_k) = O^*(2^{k \log k})$.
- Poly-space if we check all constraints just after each pattern is generated.

Improved Algorithm

- Partial pattern: vector $p = (p_1, \ldots, p_k)$, where $0 \leq p_i \leq k$, if $p_i = 0$ then s_i has not been considered yet, if $p_i = p_j > 0$ then s_i, s_j must be assigned the same user, if $0 \neq p_i \neq p_j \neq 0$, then different users. The scope of p is $\{s_i : \ p_i \neq 0\}$.

Improved Algorithm

- Partial pattern: vector $p = (p_1, \ldots, p_k)$, where $0 \leq p_i \leq k$, if $p_i = 0$ then s_i has not been considered yet, if $p_i = p_j > 0$ then s_i, s_j must be assigned the same user, if $0 \neq p_i \neq p_j \neq 0$, then different users. The scope of p is $\{s_i : \ p_i \neq 0\}$.
- For a partial pattern p, we can check all constraints c for which scope$(c) \subseteq$ scope(p).

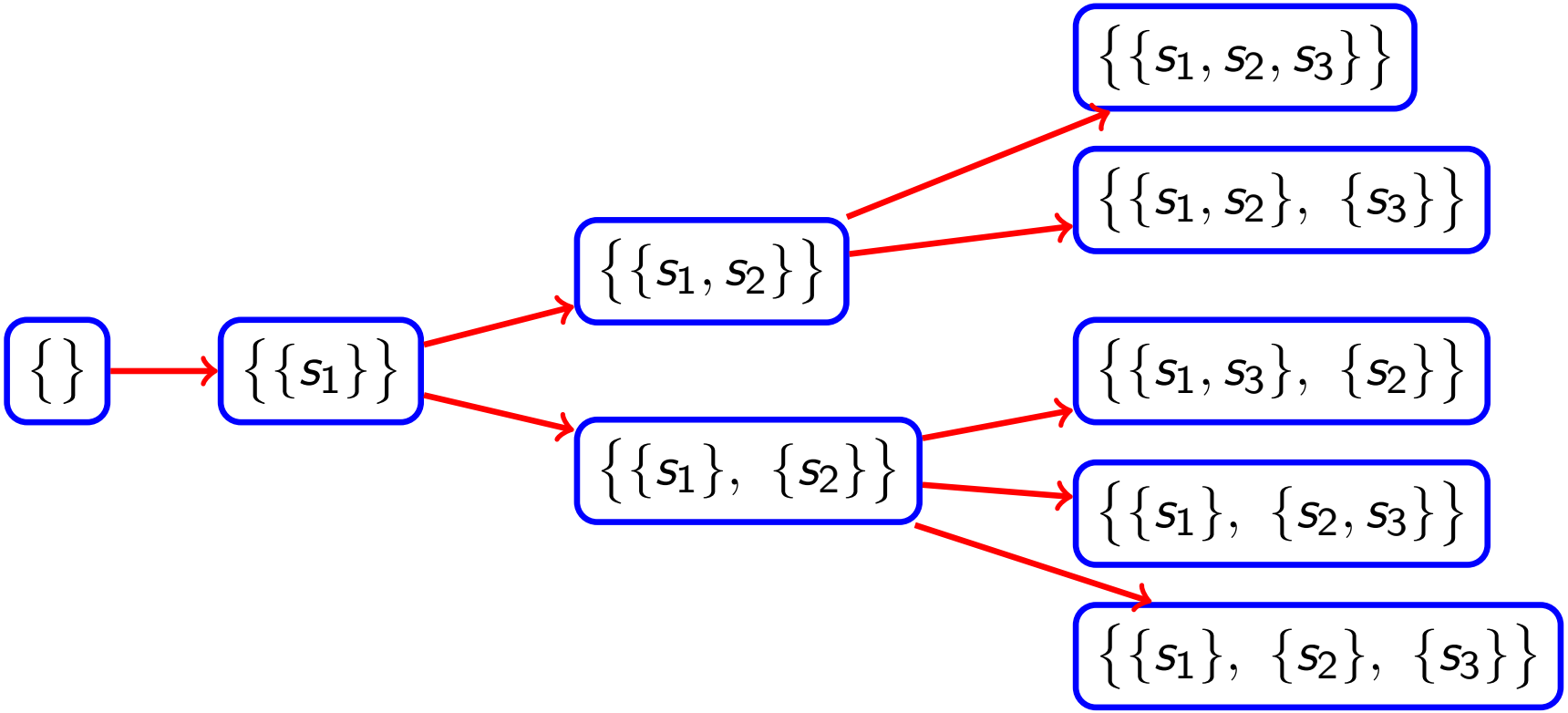
Improved Algorithm

- Partial pattern: vector $p = (p_1, \ldots, p_k)$, where $0 \leq p_i \leq k$, if $p_i = 0$ then s_i has not been considered yet, if $p_i = p_j > 0$ then s_i, s_j must be assigned the same user, if $0 \neq p_i \neq p_j \neq 0$, then different users. The scope of p is $\{s_i : \ p_i \neq 0\}$.
- For a partial pattern p, we can check all constraints c for which scope$(c) \subseteq$ scope(p).
- p can be viewed as a partition of scope(p). When a new step s_j is added to scope, we add s_i to one of the blocks or create a new block. (See the next slide.)

Improved Algorithm

- Partial pattern: vector $p = (p_1, \ldots, p_k)$, where $0 \leq p_i \leq k$, if $p_i = 0$ then s_i has not been considered yet, if $p_i = p_j > 0$ then s_i, s_j must be assigned the same user, if $0 \neq p_i \neq p_j \neq 0$, then different users. The scope of p is $\{s_i : \ p_i \neq 0\}$.
- For a partial pattern p, we can check all constraints c for which scope$(c) \subseteq$ scope(p).
- p can be viewed as a partition of scope(p). When a new step s_j is added to scope, we add s_i to one of the blocks or create a new block. (See the next slide.)
- A special procedure to decide which step to add depending on non-unary constraints.

Generating Complete Patterns



Outline

Instance Generation

- Intel Xeon CPU E5-2630 v2 (2.6 GHz) with 32 GB RAM. $n = 10k$, $10 \leq k \leq 60$.

Instance Generation

- Intel Xeon CPU E5-2630 v2 (2.6 GHz) with 32 GB RAM. $n = 10k$, $10 \leq k \leq 60$.
- Unary constraints: for each $u \in U$, first choose $b_u = |A(u)|$ randomly and uniformly from $\{1, \ldots, \lfloor k/2 \rfloor\}$, then $A(u)$ from all subsets of S of size b_u.

Instance Generation

- Intel Xeon CPU E5-2630 v2 (2.6 GHz) with 32 GB RAM. $n = 10k$, $10 \leq k \leq 60$.
- Unary constraints: for each $u \in U$, first choose $b_u = |A(u)|$ randomly and uniformly from $\{1, \ldots, \lfloor k/2 \rfloor\}$, then $A(u)$ from all subsets of S of size b_u.
- Non-unary constraints:
  - k at-least-3-out-of-5 $c = (T, \geq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \geq 3$.

Instance Generation

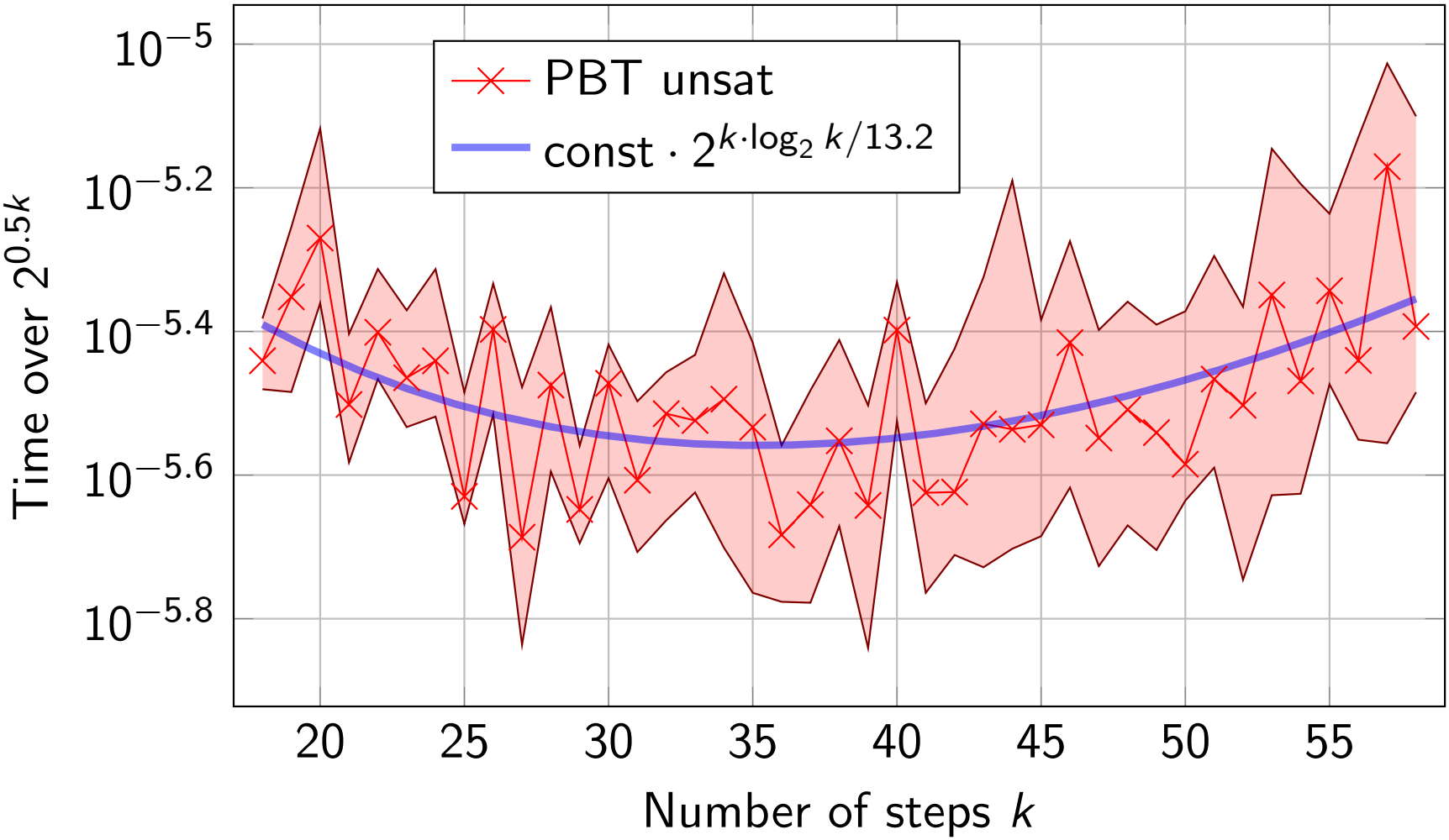
- Intel Xeon CPU E5-2630 v2 (2.6 GHz) with 32 GB RAM. $n = 10k$, $10 \leq k \leq 60$.
- Unary constraints: for each $u \in U$, first choose $b_u = |A(u)|$ randomly and uniformly from $\{1, \ldots, \lfloor k/2 \rfloor\}$, then $A(u)$ from all subsets of S of size b_u.
- Non-unary constraints:
  - k at-least-3-out-of-5 $c = (T, \geq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \geq 3$.
  - k at-most-3-out-of-5 $c = (T, \leq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \leq 3$.

Instance Generation

- Intel Xeon CPU E5-2630 v2 (2.6 GHz) with 32 GB RAM. $n = 10k$, $10 \leq k \leq 60$.
- Unary constraints: for each $u \in U$, first choose $b_u = |A(u)|$ randomly and uniformly from $\{1, \ldots, \lfloor k/2 \rfloor\}$, then $A(u)$ from all subsets of S of size b_u.
- Non-unary constraints:
  - k at-least-3-out-of-5 $c = (T, \geq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \geq 3$.
  - k at-most-3-out-of-5 $c = (T, \leq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \leq 3$.
  - e non-equals $c = (s_i, s_j, \neq)$: plan π satisfies c if $\pi(s_i) \neq \pi(s_j)$.

Instance Generation

- Intel Xeon CPU E5-2630 v2 (2.6 GHz) with 32 GB RAM. $n = 10k$, $10 \leq k \leq 60$.
- Unary constraints: for each $u \in U$, first choose $b_u = |A(u)|$ randomly and uniformly from $\{1, \ldots, \lfloor k/2 \rfloor\}$, then $A(u)$ from all subsets of S of size b_u.
- Non-unary constraints:
  - k at-least-3-out-of-5 $c = (T, \geq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \geq 3$.
  - k at-most-3-out-of-5 $c = (T, \leq)$, where $T \subseteq S$ of size 5. A plan π satisfies c if $|\pi(T)| \leq 3$.
  - e non-equals $c = (s_i, s_j, \neq)$: plan π satisfies c if $\pi(s_i) \neq \pi(s_j)$.
  - e is chosen s.t. $\mathbb{P}[(S, U, C)$ being satisfiable] is approx 50%. They are hardest instances.


PBT unsat
const · $2^{k \cdot \log_2 k/13.2}$
Time over $2^{0.5k}$
10^{-5}
$10^{-5.2}$
$10^{-5.4}$
$10^{-5.6}$
$10^{-5.8}$
20
25
30
35
40
45
50
55
Number of steps k

Contestants

PBT Improved Algorithm aka Pattern Back Tracking (in C#)

Contestants

PBT Improved Algorithm aka Pattern Back Tracking (in C#)

PUI the initial FPT algorithm by Cohen et al. (in C++)

Contestants

PBT Improved Algorithm aka Pattern Back Tracking (in C#)

PUI the initial FPT algorithm by Cohen et al. (in C++)

UDBP (Res) SAT4J using 'ordinary' pseudo-Boolean formulation of WSP in the resolution proof system mode. (cutting plane mode was much worse).

Contestants

PBT Improved Algorithm aka Pattern Back Tracking (in C#)

PUI the initial FPT algorithm by Cohen et al. (in C++)

UDBP (Res) SAT4J using 'ordinary' pseudo-Boolean formulation of WSP in the resolution proof system mode. (cutting plane mode was much worse).

PBPB (Res) SAT4J using 'FPT' formulation pseudo-Boolean formulation of WSP in the resolution proof system mode.

Contestants

PBT Improved Algorithm aka Pattern Back Tracking (in C#)

PUI the initial FPT algorithm by Cohen et al. (in C++)

UDBP (Res) SAT4J using 'ordinary' pseudo-Boolean formulation of WSP in the resolution proof system mode. (cutting plane mode was much worse).

PBPB (Res) SAT4J using 'FPT' formulation pseudo-Boolean formulation of WSP in the resolution proof system mode.

PBPB (CutP) SAT4J using 'FPT' formulation pseudo-Boolean formulation of WSP in the cutting plane mode.

Contestants

PBT Improved Algorithm aka Pattern Back Tracking (in C#)

PUI the initial FPT algorithm by Cohen et al. (in C++)

UDBP (Res) SAT4J using 'ordinary' pseudo-Boolean formulation of WSP in the resolution proof system mode. (cutting plane mode was much worse).

PBPB (Res) SAT4J using 'FPT' formulation pseudo-Boolean formulation of WSP in the resolution proof system mode.

PBPB (CutP) SAT4J using 'FPT' formulation pseudo-Boolean formulation of WSP in the cutting plane mode.

CP-SAT CP-SAT from OR-tools (Google) using 'ordinary' CSP formulation of WSP

Pseudo-Boolean formulations of WSP

- $x_{s,u} = 1$ iff u is assigned to s.

Pseudo-Boolean formulations of WSP

- $x_{s,u} = 1$ iff u is assigned to s.
- 'Ordinary' formulation (only not-equals):

$$\begin{aligned} \textstyle\sum_{u \in U} x_{s,u} = 1 \quad & \forall s \in S, \\ x_{s,u} = 0 \quad & \forall s \in S \text{ and } \forall u \in U \setminus A(s), \\ x_{s_1,u} + x_{s_2,u} \leq 1 \quad & \forall \text{ not-equals with scope } \{s_1, s_2\} \text{ and } \forall u \in U. \end{aligned}$$

- 'FPT' formulation uses $x_{s,u}$'s and 'FPT variables' $M_{s,s'}$. $M_{s,s'} = 1$ if s, s' are assigned the same user and $M_{s,s'} = 0$, otherwise. Much longer formulation.

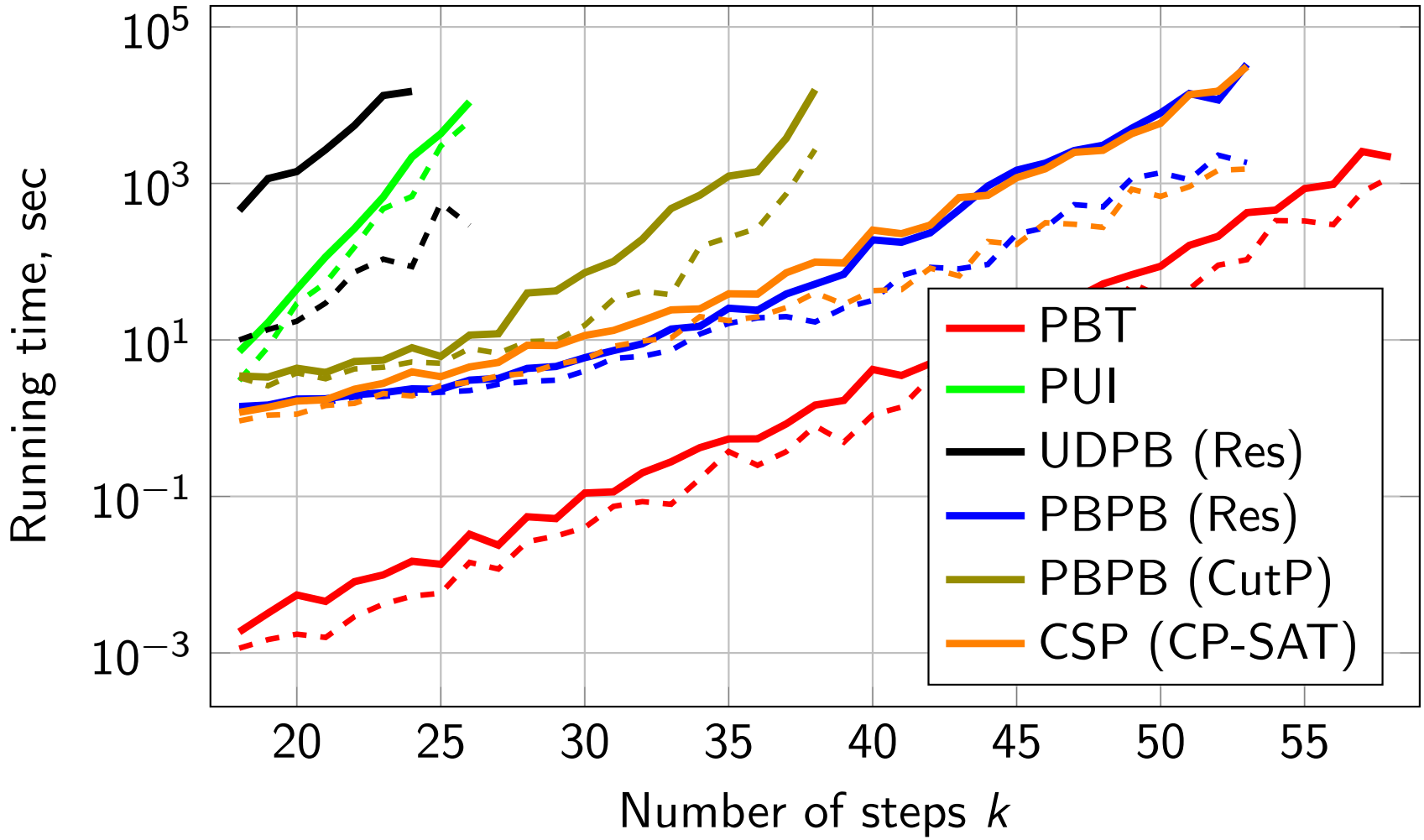
Brief Report of Experimental Results

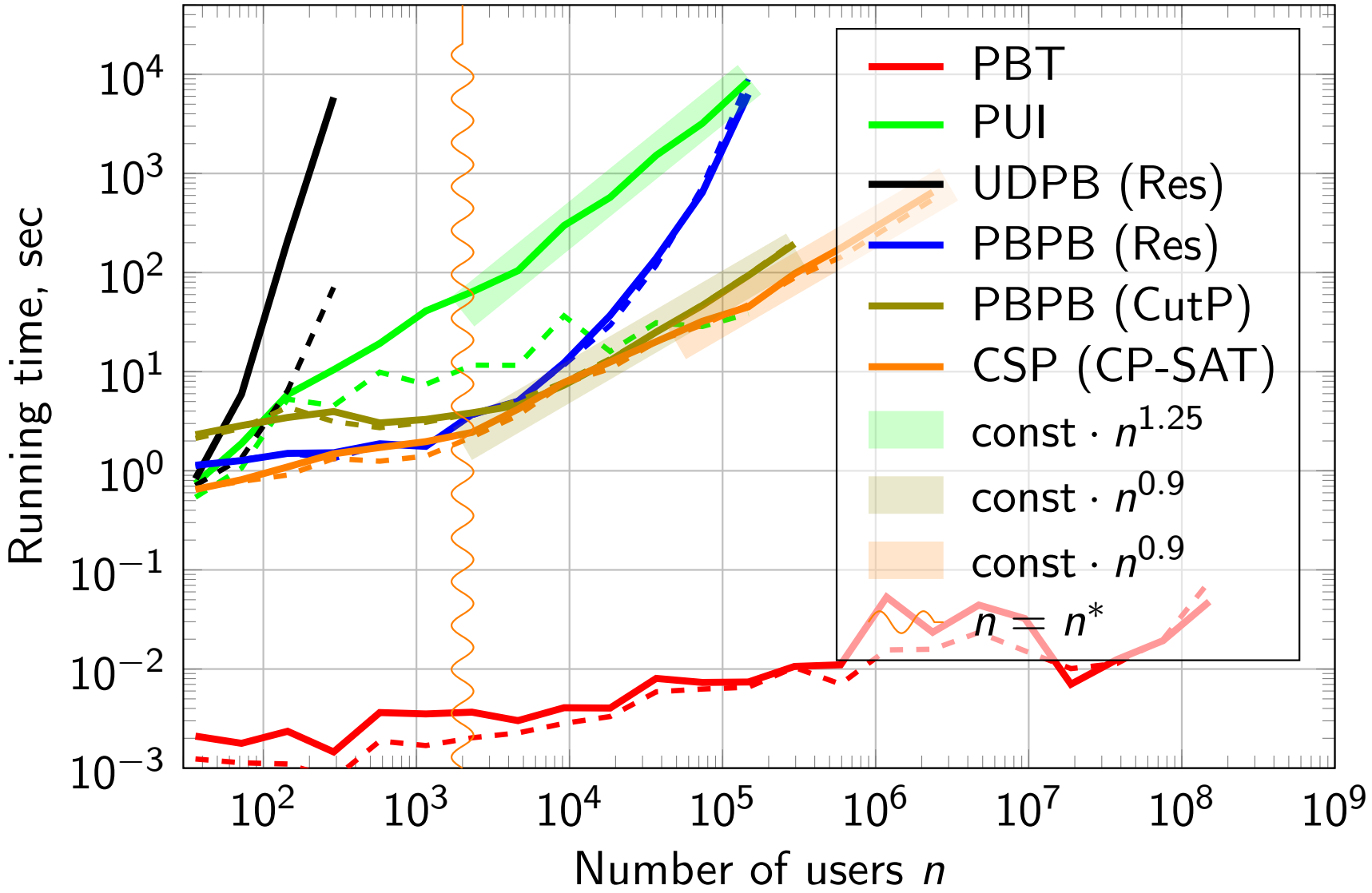
Theorem

Every UI constraint can be encoded using $M_{s,s'}$'s.

Proof.

Every UI constraint corresponds to a set of patterns $\mathcal{P}$.
Constraints: $M_{s,s'} = M_{s',s}$ for all $s < s'$, $M_{s,s} = 1$ for all $s \in S$, and $\sum_{B \in p} \sum_{s' < s'' \in B} (1 - M_{s',s''}) +$
$\sum_{B' \neq B'' \in p} \sum_{s' \in B'} \sum_{s'' \in B'', s' < s''} M_{s',s''} \geq 1$ for all $p \in \overline{\mathcal{P}}$. □


Running time, sec
10^5
10^3
10^1
10^{-1}
10^{-3}
20
25
30
35
40
45
50
55
Number of steps k
PBT
PUI
UDPB (Res)
PBPB (Res)
PBPB (CutP)
CSP (CP-SAT)


Running time, sec
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Number of users n
PBT
PUI
UDPB (Res)
PBPB (Res)
PBPB (CutP)
CSP (CP-SAT)
const · $n^{1.25}$
const · $n^{0.9}$
const · $n^{0.9}$
$n = n^*$

Report of Experimental Results

- Winners: PBT, Runners-up: CP-SAT and the best of PBPB (Res) and PBPB (CutP).

Report of Experimental Results

- Winners: PBT, Runners-up: CP-SAT and the best of PBPB (Res) and PBPB (CutP).
- Comparison: For $k = 50$, PBT takes median time 100 sec. for UNSAT instances. CP-SAT is 2-3 orders of magnitude slower than PBT and uses much more memory.

Report of Experimental Results

- Winners: PBT, Runners-up: CP-SAT and the best of PBPB (Res) and PBPB (CutP).
- Comparison: For $k = 50$, PBT takes median time 100 sec. for UNSAT instances. CP-SAT is 2-3 orders of magnitude slower than PBT and uses much more memory.
- Remarkable: CP-SAT uses 'ordinary' CSP formulation of WSP, not 'FPT' (we did not know how to get one), nevertheless displays an FPT-like behaviour.

Report of Experimental Results

- Winners: PBT, Runners-up: CP-SAT and the best of PBPB (Res) and PBPB (CutP).
- Comparison: For $k = 50$, PBT takes median time 100 sec. for UNSAT instances. CP-SAT is 2-3 orders of magnitude slower than PBT and uses much more memory.
- Remarkable: CP-SAT uses 'ordinary' CSP formulation of WSP, not 'FPT' (we did not know how to get one), nevertheless displays an FPT-like behaviour.
- Likely explanation: CP-SAT formulation is reduced to a SAT formulation. 'FPT variables' appear in the SAT formulation.

Outline

Valued WSP Theory

- Crampton, GG, Karapetyan (SACMAT 2015), best paper award, journal version in J. Comput. Security 2017.

Valued WSP Theory

- Crampton, GG, Karapetyan (SACMAT 2015), best paper award, journal version in J. Comput. Security 2017.
- Sometimes WSP instance cannot be satisfied, but it will be OK to falsify some 'soft' constraints especially if they are not falsified 'too much', e.g. for at-most-3-out-of-5 "only" 4 users, not 5, are assigned.

Valued WSP Theory

- Crampton, GG, Karapetyan (SACMAT 2015), best paper award, journal version in J. Comput. Security 2017.
- Sometimes WSP instance cannot be satisfied, but it will be OK to falsify some 'soft' constraints especially if they are not falsified 'too much', e.g. for at-most-3-out-of-5 "only" 4 users, not 5, are assigned.
- Formalisation: For each constraint c and plan π, if π satisfies c then $w_c(\pi) = 0$, otherwise $w_c(\pi) > 0$. We wish to find $\text{argmin}_\pi \sum_{c \in C} w_c(\pi)$.

Valued WSP Theory

- Crampton, GG, Karapetyan (SACMAT 2015), best paper award, journal version in J. Comput. Security 2017.
- Sometimes WSP instance cannot be satisfied, but it will be OK to falsify some 'soft' constraints especially if they are not falsified 'too much', e.g. for at-most-3-out-of-5 "only" 4 users, not 5, are assigned.
- Formalisation: For each constraint c and plan π, if π satisfies c then $w_c(\pi) = 0$, otherwise $w_c(\pi) > 0$. We wish to find $\text{argmin}_\pi \sum_{c \in C} w_c(\pi)$.
- A constraint c is UI (for Valued WSP) if $w_c(\pi) = w_c(\pi')$, where π and π' are equivalent, i.e. the same up to permutation of users.

Valued WSP Theory

- Crampton, GG, Karapetyan (SACMAT 2015), best paper award, journal version in J. Comput. Security 2017.
- Sometimes WSP instance cannot be satisfied, but it will be OK to falsify some 'soft' constraints especially if they are not falsified 'too much', e.g. for at-most-3-out-of-5 "only" 4 users, not 5, are assigned.
- Formalisation: For each constraint c and plan π, if π satisfies c then $w_c(\pi) = 0$, otherwise $w_c(\pi) > 0$. We wish to find $\text{argmin}_\pi \sum_{c \in C} w_c(\pi)$.
- A constraint c is UI (for Valued WSP) if $w_c(\pi) = w_c(\pi')$, where π and π' are equivalent, i.e. the same up to permutation of users.
- If all non-unary constraints are UI, we have an $O^*(2^{k \log k})$-time algorithm for Valued WSP.

Experimental Results

- Contestants: PBnB vs CPLEX 12.6.

Experimental Results

- Contestants: PBnB vs CPLEX 12.6.
- $k = 20, 25, 30, 35$, $n = 10k$. 1 hour per instance.

Experimental Results

- Contestants: PBnB vs CPLEX 12.6.
- $k = 20, 25, 30, 35$, $n = 10k$. 1 hour per instance.
- PBnB solved every instance for $k \leq 30$ and a large majority for $k = 35$.

Experimental Results

- Contestants: PBnB vs CPLEX 12.6.
- $k = 20, 25, 30, 35$, $n = 10k$. 1 hour per instance.
- PBnB solved every instance for $k \leq 30$ and a large majority for $k = 35$.
- CPLEX 12.6 solved all instances only for $k = 20$, for $k = 35$ less than 43%.

Thank you! Takk skal du ha!

Any questions?