Local search and conflict learning: an application to the Orienteering Problem with Time Windows (OPTW)

Trong-Hieu Tran[1, 2] Cédric Pralet[2] Hélène Fargier[1]

[1]IRIT-CNRS, Université de Toulouse, France

[2]ONERA/DTIS, Université de Toulouse, France

ANITI@ARDM Workshop, June 2022

Orienteering Problem with Time Windows (OPTW)

- N set of clients
- $\forall i \in N$: reward R_i, service time d_i, $TW_i = [opening_i, closing_i]$
- Find $S \subseteq N$ s.t. max. $\Sigma_{i \in S} R_i$ & $\exists \sigma(S)$ a feasible sequence over S

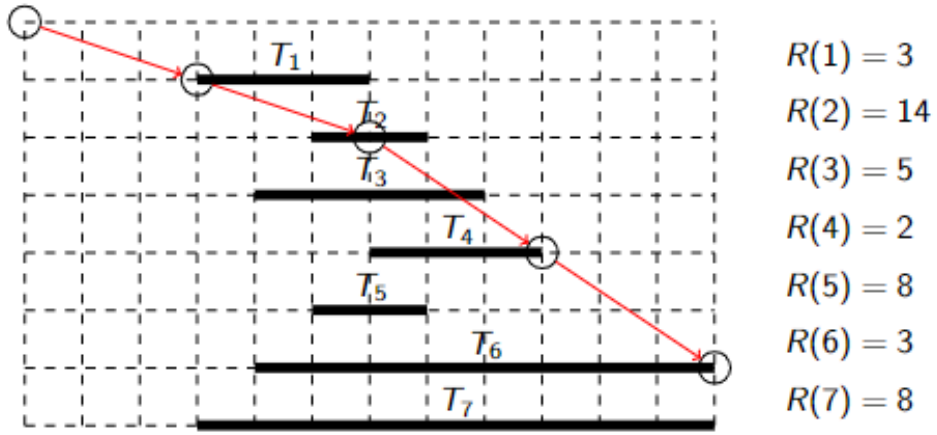


Figure: OPTW example involving 7 customers, and representation of a valid sequence of visits

Orienteering Problem with Time Windows (OPTW)

- N set of clients
- $\forall i \in N$: reward R_i, service time d_i, $TW_i = [opening_i, closing_i]$
- Find $S \subseteq N$ s.t. max. $\Sigma_{i \in S} R_i$ & $\exists \sigma(S)$ a feasible sequence over S

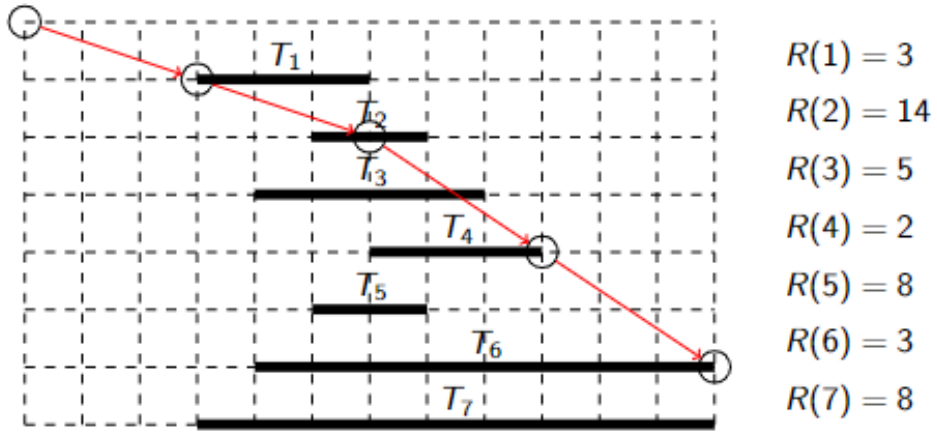


Figure: OPTW example involving 7 customers, and representation of a valid sequence of visits

OPTW = client selection + sequencing

Target application: Earth Observation Satellite scheduling

Incomplete search techniques for OPTW

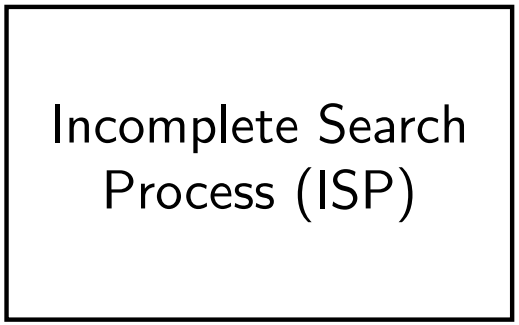
Reference	Algorithm	Percentage of best known solutions m=1	m=2	m=3	m=4	Average
Labadie et al. (2011)	GRASP-ELS	50.0	21.1	32.9	46.1	37.5
Lin and Yu (2012)	SSA	51.3	34.2	39.5	56.6	45.4
Labadie et al. (2012)	GVNS	36.8	30.3	40.8	44.7	38.2
Souffriau et al. (2012)	GRILS	51.3	15.8	22.4	39.5	32.3
Hu and Lim (2014)	I3CH	43.4	34.2	57.9	55.3	47.7
Cura (2014)	ABC	48.7	36.8	46.1	48.7	45.1
Gunawan et al. (2015b)	ILS	68.4	51.3	56.6	55.3	57.9
Gunawan et al. (2015a)	SAILS	67.1	50.0	57.9	53.9	57.2
Schmid & Ehmke (2016)	LNS	82.9	71.1	65.8	60.1	70.0

Figure: State-of-the-art solution approaches to Team OPTW[1]

[1] [1] V. Schmid and J. F. Ehmke (2017) "An effective large neighborhood search for the TOPTW".

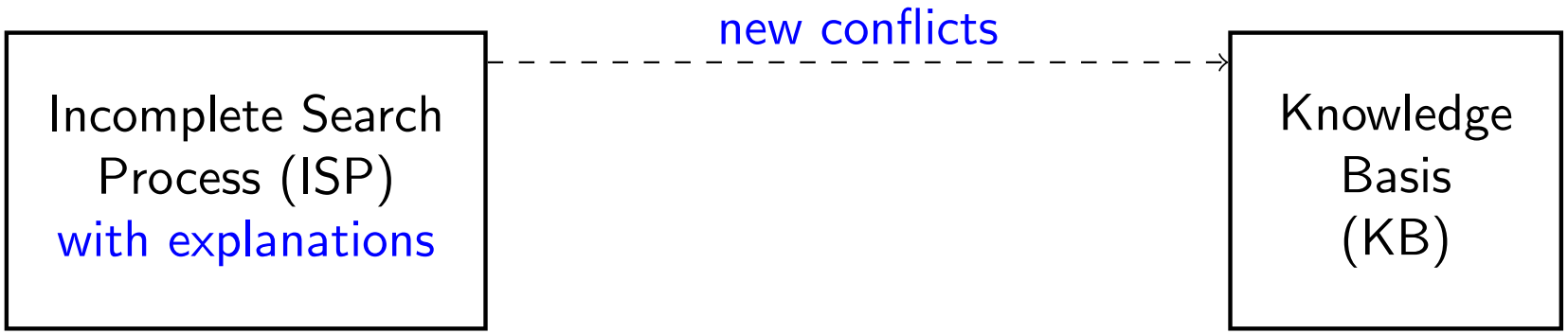
Motivation and general schema

- **Goal:** boosting the construction procedure by integrating a long-term memory



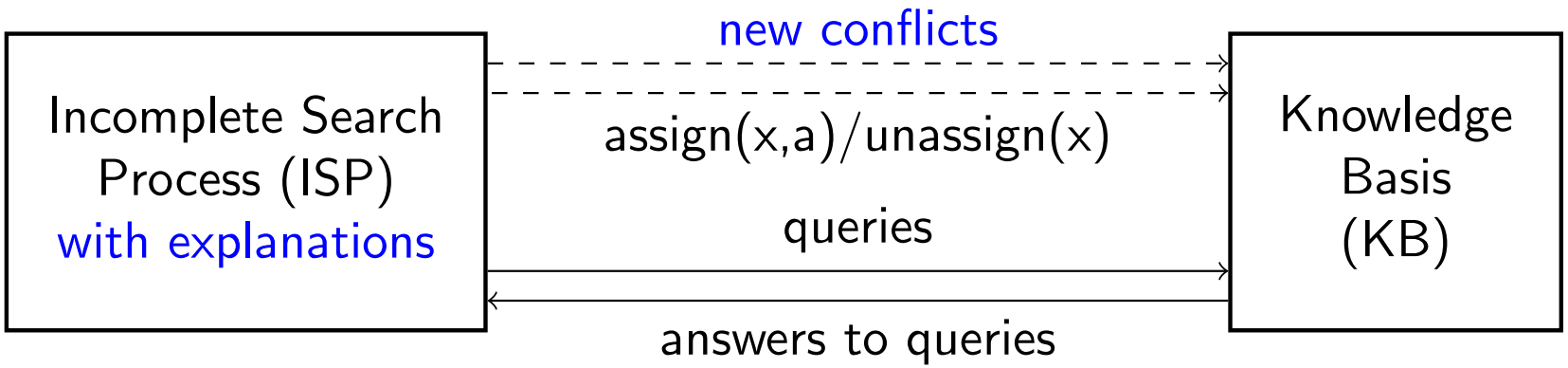
Motivation and general schema

- **Goal:** boosting the construction procedure by integrating a long-term memory



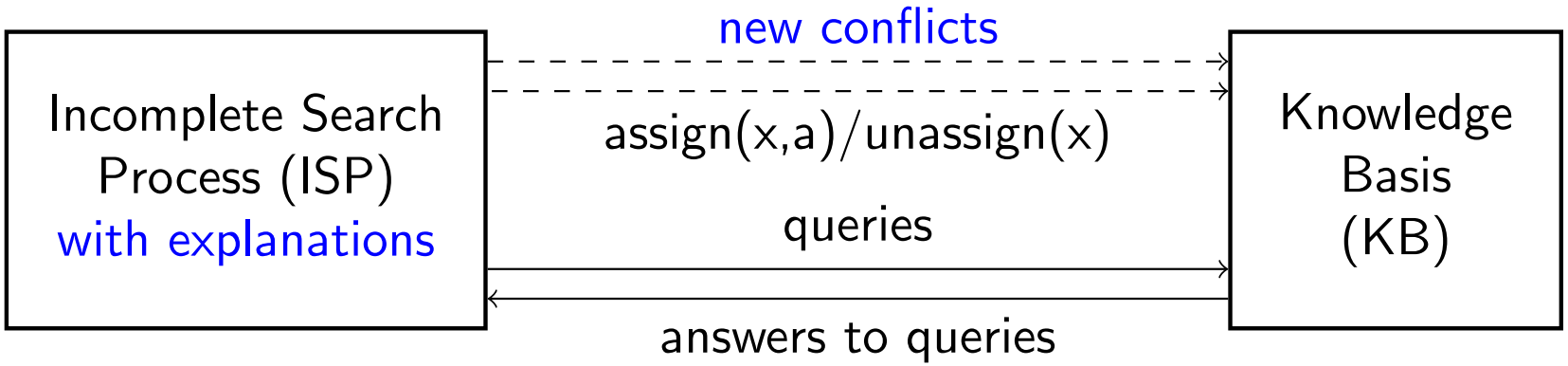
Motivation and general schema

- **Goal:** boosting the construction procedure by integrating a long-term memory



Motivation and general schema

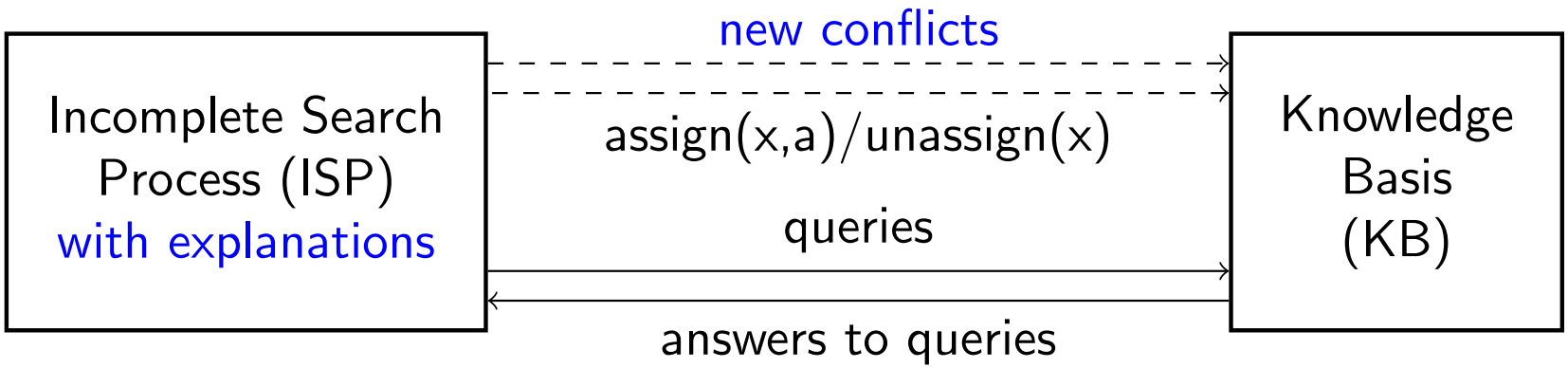
- **Goal:** boosting the construction procedure by integrating a long-term memory



- **Related works:**
  - Long-term memory - Tabu search
  - Combine search and learning - CDCL in SAT or LCG in CP

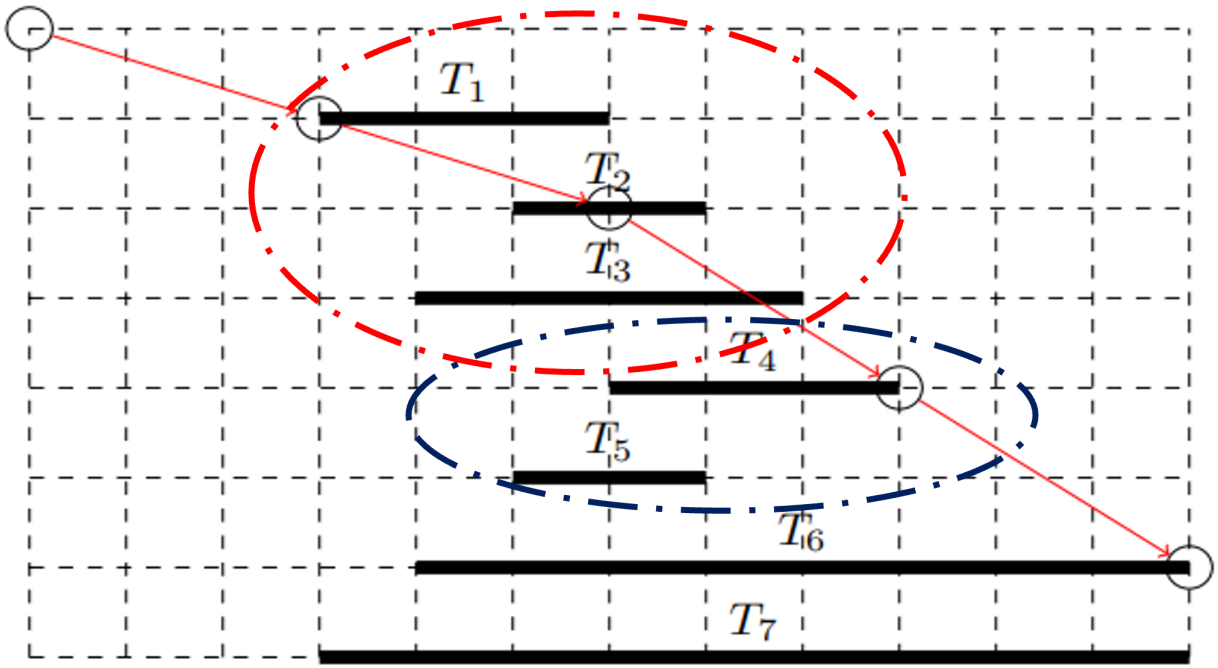
Motivation and general schema

- **Goal:** boosting the construction procedure by integrating a long-term memory



- **Related works:**
  - Long-term memory - Tabu search
  - Combine search and learning - CDCL in SAT or LCG in CP
- **Problem statements:**
  - Explanation module: Conflict extraction procedure ?
  - KB module: KB data structures ? queries on KB ?

An example of conflicts



$$R(1) = 3,\ R(2) = 14,\ R(3) = 5,\ R(4) = 2,\ R(5) = 8,\ R(6) = 3,\ R(7) = 8$$

Figure: An example of two conflicts

Conflict extraction: a Dynamic Programming approach

TW-conflict

Given N is the set of all clients, a subset $S \subset N$ is a TW-conflict if there is no visit ordering that satisfies TW constraints of all clients $k \in S$

Conflict extraction: a Dynamic Programming approach

TW-conflict

Given N is the set of all clients, a subset $S \subset N$ is a TW-conflict if there is no visit ordering that satisfies TW constraints of all clients $k \in S$

- Checking if S is a TW-conflict is also NP-hard $\sim$ solving TSPTW

Conflict extraction: a Dynamic Programming approach

TW-conflict

Given N is the set of all clients, a subset $S \subset N$ is a TW-conflict if there is no visit ordering that satisfies TW constraints of all clients $k \in S$

- Checking if S is a TW-conflict is also NP-hard $\sim$ solving TSPTW
- Goal: searching *the smallest* TW-conflicts with DP approach

Algorithm 1: QUICKXPLAIN(S)

```
1 size ← 2
2 while size < k do
3     if ∃ set of TW-conflicts C_S ⊂ S s.t. ∀c ∈ C_S : |c| = k then
4         return C_S
5     size++
6 return ∅
```

Conflict extraction: a Dynamic Programming approach

TW-conflict

Given N is the set of all clients, a subset $S \subset N$ is a TW-conflict if there is no visit ordering that satisfies TW constraints of all clients $k \in S$

- Checking if S is a TW-conflict is also NP-hard $\sim$ solving TSPTW
- Goal: searching *the smallest* TW-conflicts with DP approach

Algorithm 2: QUICKXPLAIN(S)

```
1 size ← 2
2 while size < k do
3     if ∃ set of TW-conflicts C_S ⊂ S s.t. ∀c ∈ C_S : |c| = k then
4         return C_S
5     size++
6 return ∅
```

- Searching minimal conflicts in S is *very fast* if k is small enough

Different types of clauses

Generating clauses at three modules:

- **DP-based Xplainer**: TW conflicts

$$(\neg x_1 \vee \ldots \vee \neg x_k)$$

- **Search engine**: clauses for avoiding visited local optima

$$(x_1 \vee \ldots \vee x_m)$$

- **KB**: clause learning - lazy clause generation

$$x_1 \implies UP(.)$$

$$\neg x_3 \implies UP(.)$$

$$x_5 \implies UP(.) \implies \bot$$

Learnt clause:

$$\neg x_1 \vee x_3 \vee \neg x_5$$

KB data structures: requisites

An ideal KB should ...

- **Quickly add new conflicts**
- **Compactly represent the set of conflicts acquired during the search**
- **Quickly detect invalid selections (query)**

Other advanced queries can be considered:

- Extract the *optimal* assignment (i.e. max total rewards, for intensification)
- Extract the *farthest* assignment (for diversification)

Multiple candidate data structures

Conflicts $S = \{i_1, \ldots, i_k\}$ is expressed as a set of clauses $\neg x_{i_1} \vee \ldots \vee \neg x_{i_k}$

- Two watched literals: simple implementation, not very compact in size
- Incremental SAT: ready-to-use existing SAT solvers, not very compact in size

Multiple candidate data structures

Conflicts $S = \{i_1, \ldots, i_k\}$ is expressed as a set of clauses $\neg x_{i_1} \vee \ldots \vee \neg x_{i_k}$

- Two watched literals: simple implementation, not very compact in size
- Incremental SAT: ready-to-use existing SAT solvers, not very compact in size

Conflicts $S = \{i_1, \ldots, i_k\}$ is compiled into an Ordered Binary Decision Diagram (OBDD)

- OBDD: polynomial query answering, challenging in case of on-the-fly compilation

Multiple candidate data structures

Conflicts $S = \{i_1, \ldots, i_k\}$ is expressed as a set of clauses $\neg x_{i_1} \vee \ldots \vee \neg x_{i_k}$

- Two watched literals: simple implementation, not very compact in size
- Incremental SAT: ready-to-use existing SAT solvers, not very compact in size

Conflicts $S = \{i_1, \ldots, i_k\}$ is compiled into an Ordered Binary Decision Diagram (OBDD)

- OBDD: polynomial query answering, challenging in case of on-the-fly compilation

Conflict $S = \{i_1, \ldots, i_k\}$ is expressed as Pseudo-Boolean constraints $x_{i_1} + \ldots + x_{i_k} \leq k - 1$

- Relaxed LP model with $x_i \in [0, 1]$

A simple but effective KB-2WL

- Each clause is watched by at most 2 literals
- Unit clause $\neg x_i$ implies that client i is *non-selectable*
- Support backtracking in random order

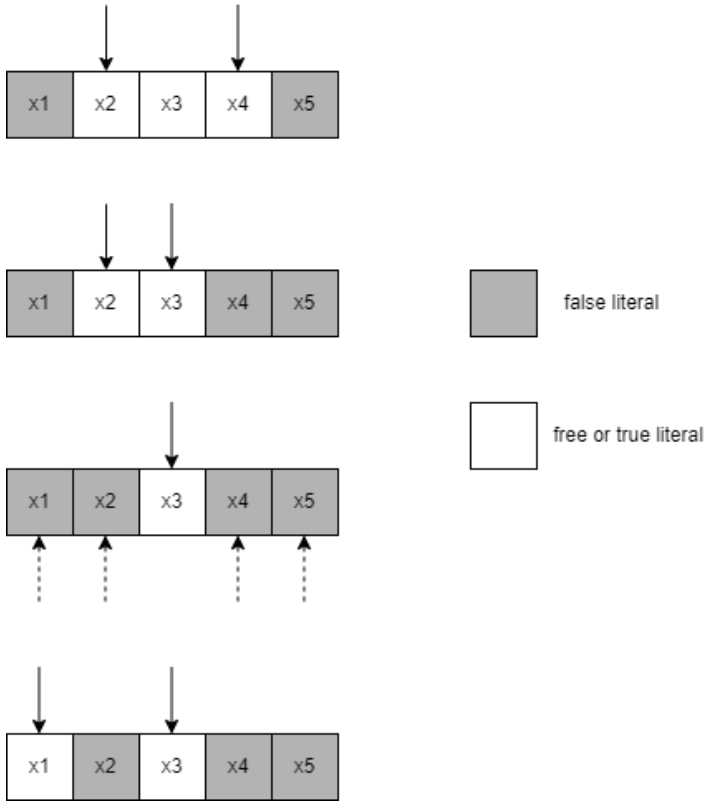


Figure: An example of KB with 2-watched-literals technique that supports random backtracking

Experimental settings

- Benchmark instances:

Instance Set	#instances	#customers	TW remark
Solomon 1	29	100	
Solomon 2	27	100	wider TW
Cordeau 1	10	48-288	
Cordeau 2	10	48-288	wider TW

- Simple integration of KB into LNS

Algorithm 3: KBLNS

```
1  S ← CONSTRUCT()
2  while ite++ < maxIte do
3      XPLAIN(S, KB)          // generate TW-conflicts at local optimum ∀c ∉ S*
4      S ← DESTROY(S)         // remove k nodes from the current visit set
5      S ← REPAIR(S, KB)      // check selection with KB before checking insertion
```

Impact of Time Windows in conflict generation

Remark: wider TWs $\rightarrow$ more visits $\rightarrow$ larger conflicts $\rightarrow$ longer explanation time

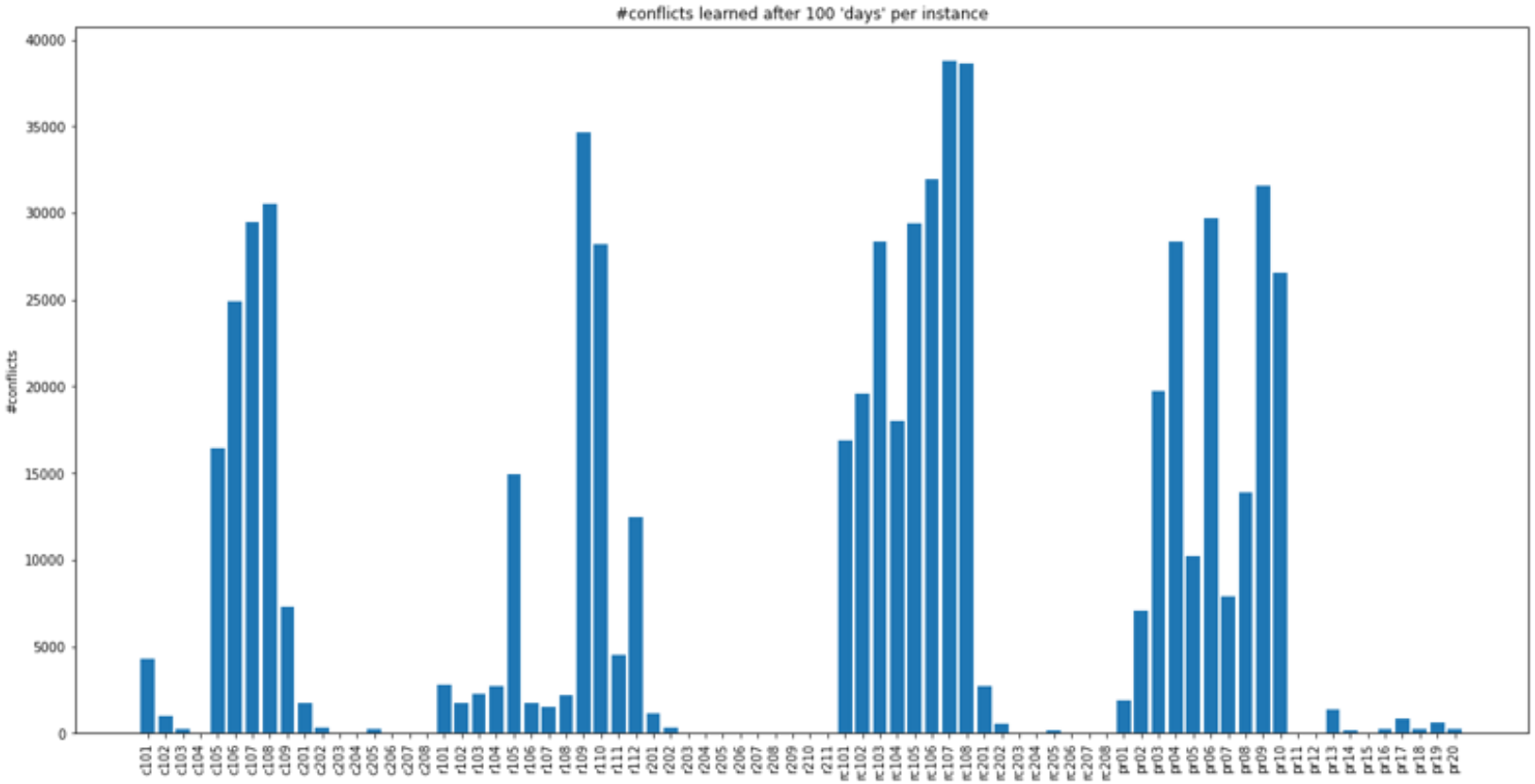


Figure: Number of conflicts extracted after 100 random calls of conflict extraction

Experimental results: Does KB speed up the search?

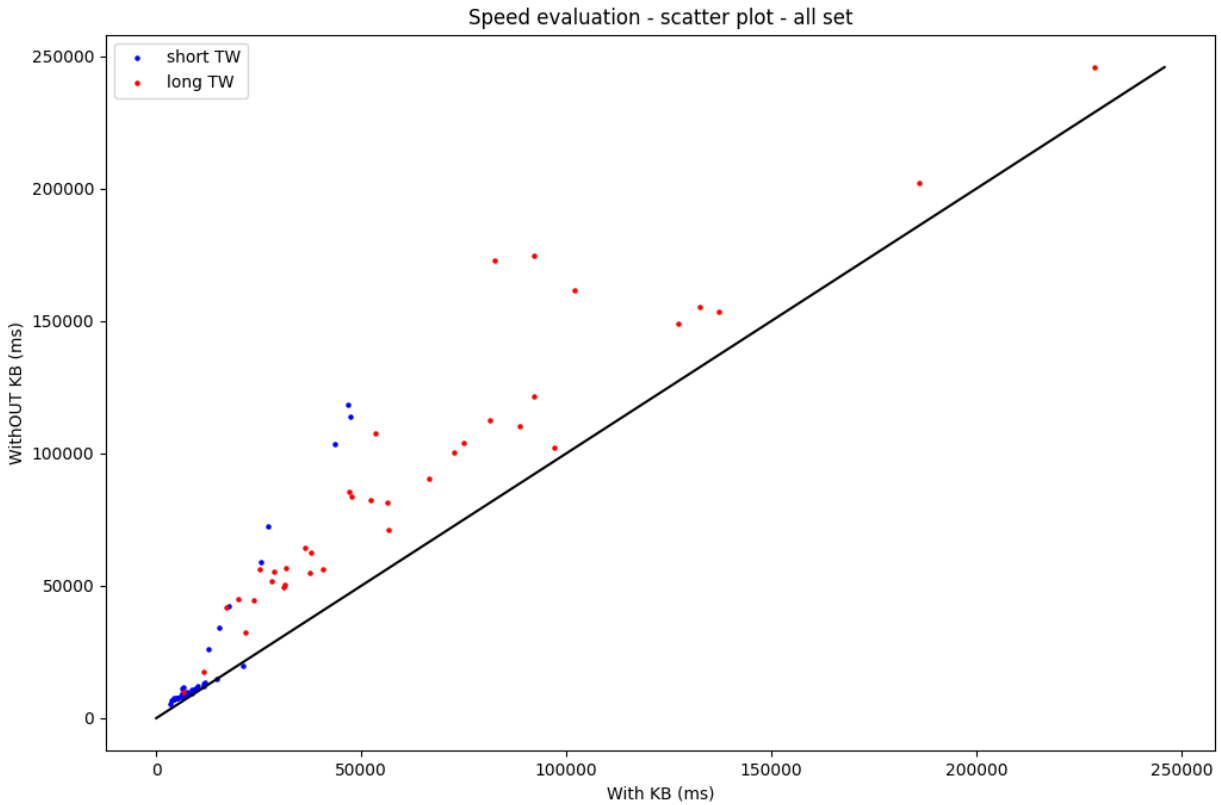
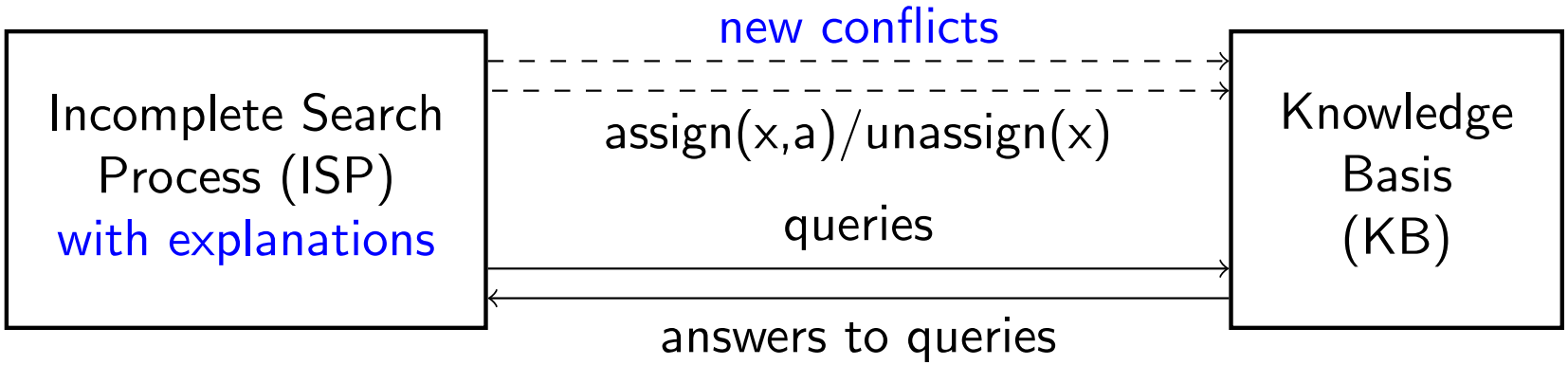


Figure: Scatter plot - *maxIte* = 10000 (x-axis: execution time without KB, y-axis: execution time with KB, blue: instances of set 1, red: instances of set 2)

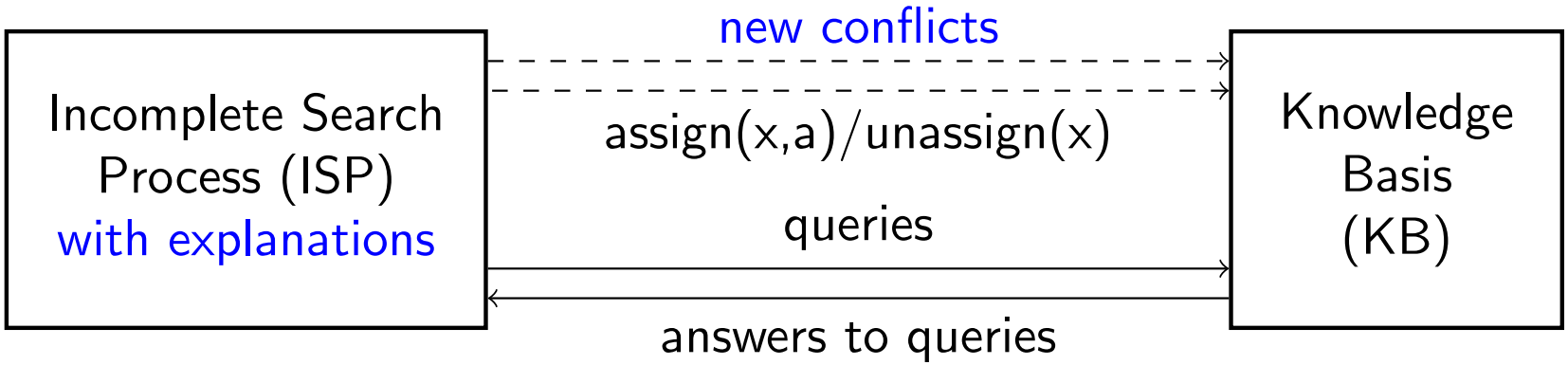
Main contributions

- An exact explanation procedure for generating TW-conflicts



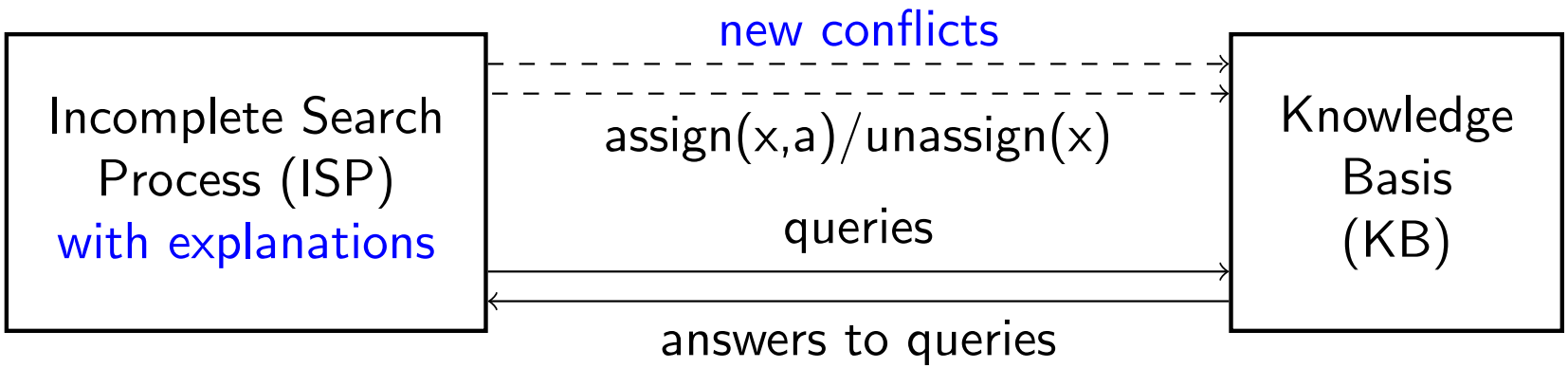
Main contributions

- An exact explanation procedure for generating TW-conflicts
- A simple but effective KB for storing TW-conflicts



Main contributions

- An exact explanation procedure for generating TW-conflicts
- A simple but effective KB for storing TW-conflicts
- A simple use of KB for neighborhood pruning helps speed up the search process



Future perspectives

- Extend to multi-vehicle problem (TOPTW)
  boolean variables $\rightarrow$ *integer variables*
- Explore other KBs and advanced queries
  KB for pruning $\rightarrow$ *KB for guiding the search*
- Generate other types of conflicts
  exact TW-conflicts $\rightarrow$ *approximate conflicts, etc.*

Thank you for your attention !

Details about pseudocode of KBLNS

Algorithm 4: KBLNS

```
1 CONSTRUCT()
2 while ite++ < maxIte do
3     XPLAIN()
4     DESTROY()
5     KBREPAIR()
```

Algorithm 5: KBREPAIR

```
1 F ← KBUPDATE(N')
2 while F ≠ ∅ do
3     (n*, p*) ← SELECT(F)
4     N' ← N' \ {n*}
5     F ← KBUPDATE(N')
```

Algorithm 6: KBUPDATE

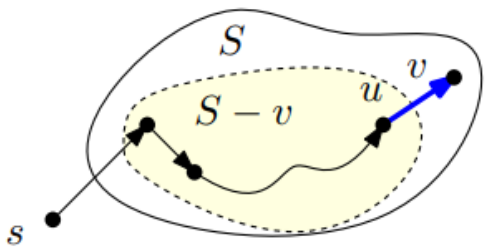
```
1 F ← ∅
2 for all node n ∈ N' do
3     if CHECK-SELECTION(n, m, KB) then
4         CHECK-INSERTION(n, m, F)
5 return F
```

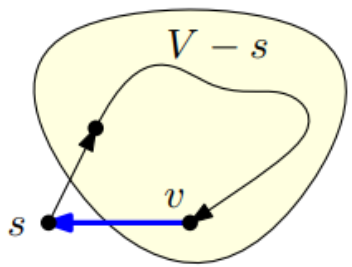
Conflict extraction: a Dynamic Programming approach

DP state

Given a node $v \in S$, $cost(S, v)$ is the travel time of the shortest path that traverses all nodes in $S \setminus \{v\}$ and end at v.

Remark: in case of late arrival at v, $cost(S, v) = \infty$





TW-conflict

S is a TW-conflict iff $\forall v \in S$,
$cost(S, v) = \infty$ or $max(cost(S, v), openingDate(v)) + d(v, s) > T_{max}$

Pseudocode of conflict extraction

Algorithm 7: XPLAIN

```
1  conflictSet ← ∅
2  for size ∈ {2, ..., n − 1} do
3      for S ⊆ V − s with |S| = size do
4          conflictCount ← 0
5          for v ∈ S do
6              cost(S, v) ← min{arrival_{u≺v} | u ∈ S − v}
7              if cost(S, v) = ∞ or
                 max(cost(S, v), openingDate(v)) + d(v, s) > T_max then
8                  conflictCount + +
9          if conflictCount == size then
10             conflictSet ← conflictSet ∪ S
11     if |conflictSet| > 0 then
12         return (True, conflictSet)
13 return (False, ∅)
```

Analysis of explanation module

To reduce the overhead of conflict extraction procedure, we propose to:

- Determine heuristically S_c the scope of conflict with $|S_c| = conflictMaxSize$
- Set $xplainTimeLimitPerIteration$
- Set *early stopping* for the learning phase

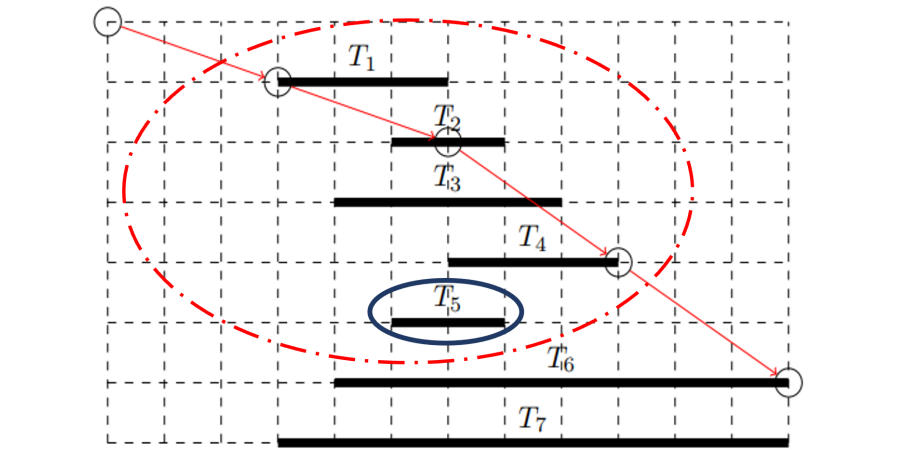


Figure: An example of selecting heuristically 5 clients to explain why $x_5 \notin S^*$

Detail about pr01

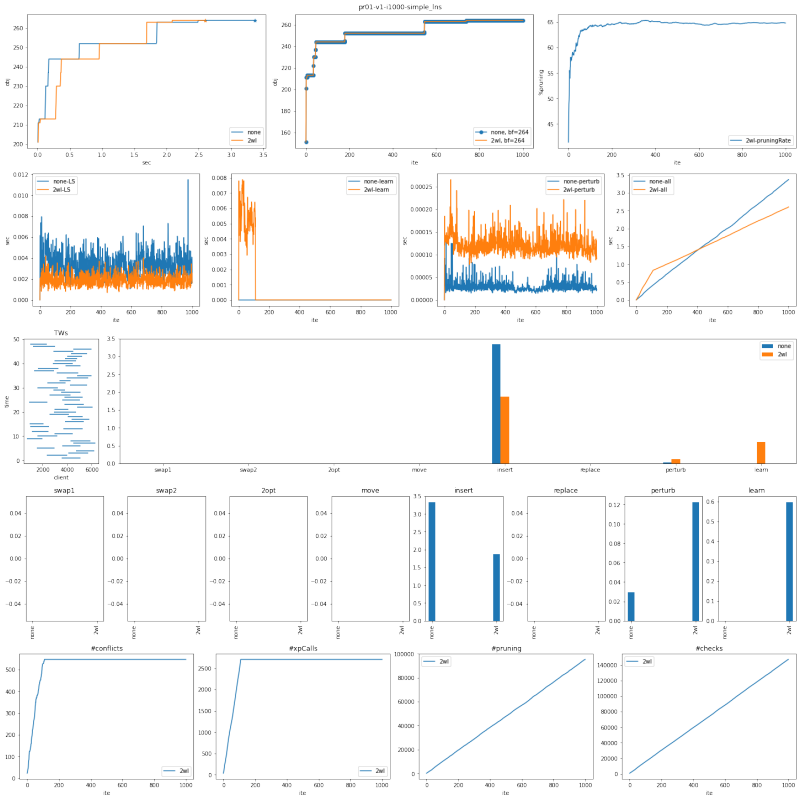


Figure: pr01-48 clients

Detail about pr06

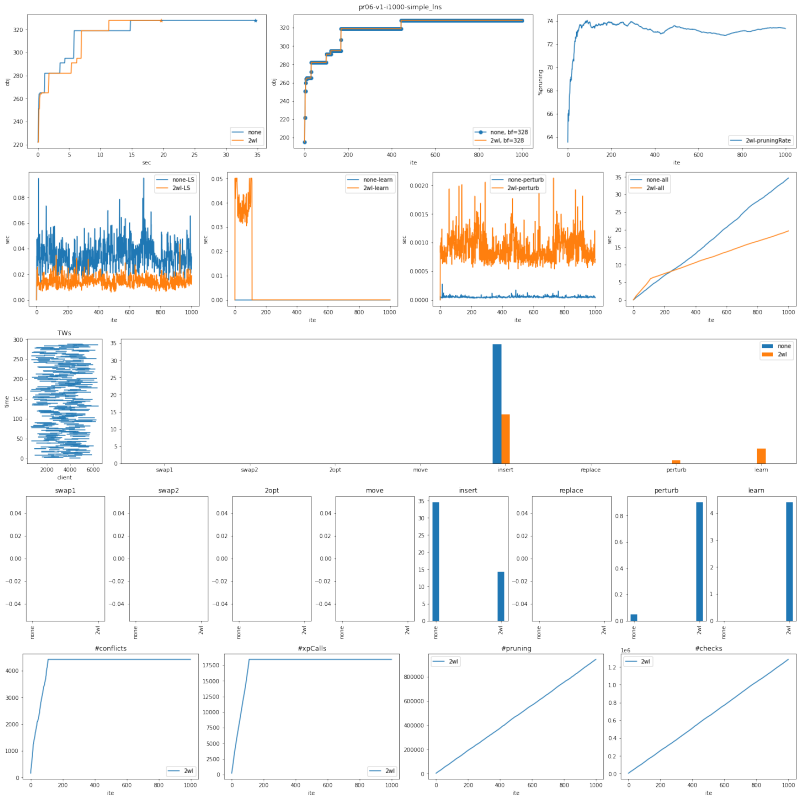


Figure: pr06-288 clients

Experimental results: Does KB speed up the search?

For each instance, we compute:

$$r = \log_2 \frac{no_KB_CPU_time}{with_KB_CPU_time}$$

$r < 0 \implies$ faster without KB
$r > 0 \implies$ faster with KB,
$r = 1$ means 2 x faster with KB

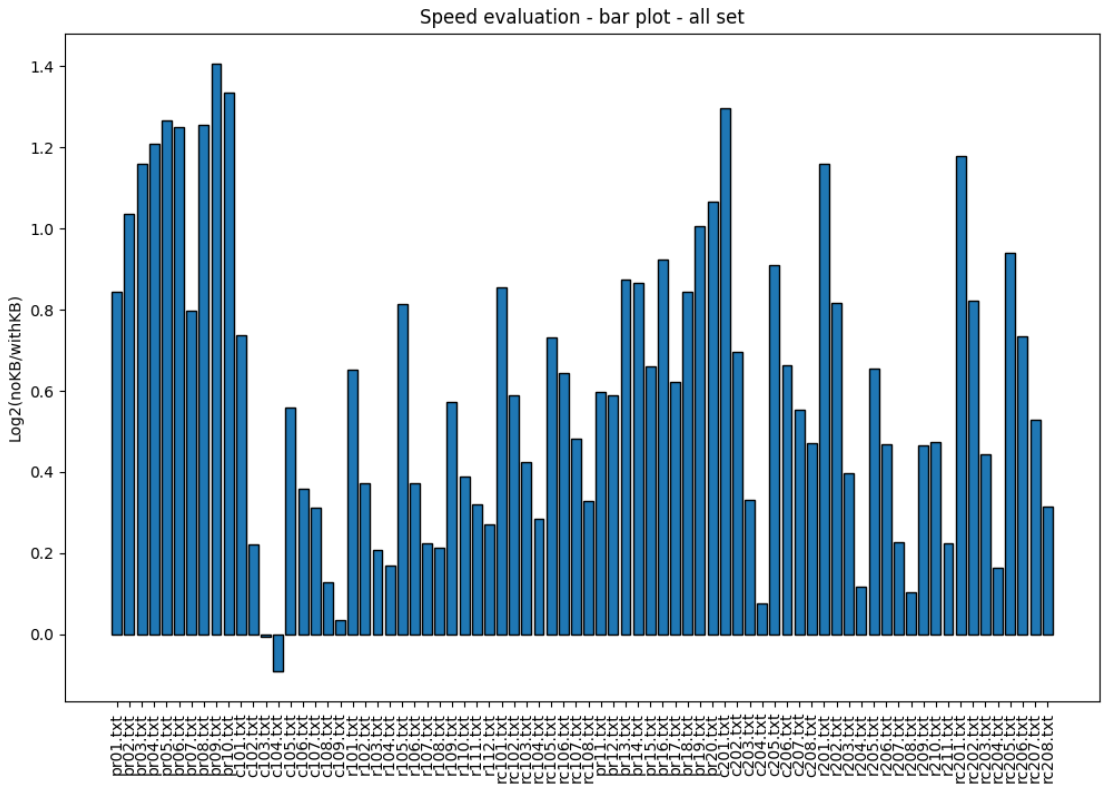


Figure: Bar plot