Constraint Programming with External Worst-Case Traversal Time Analysis

Pierre Talbot

pierre.talbot@uni.lu

16th May 2023

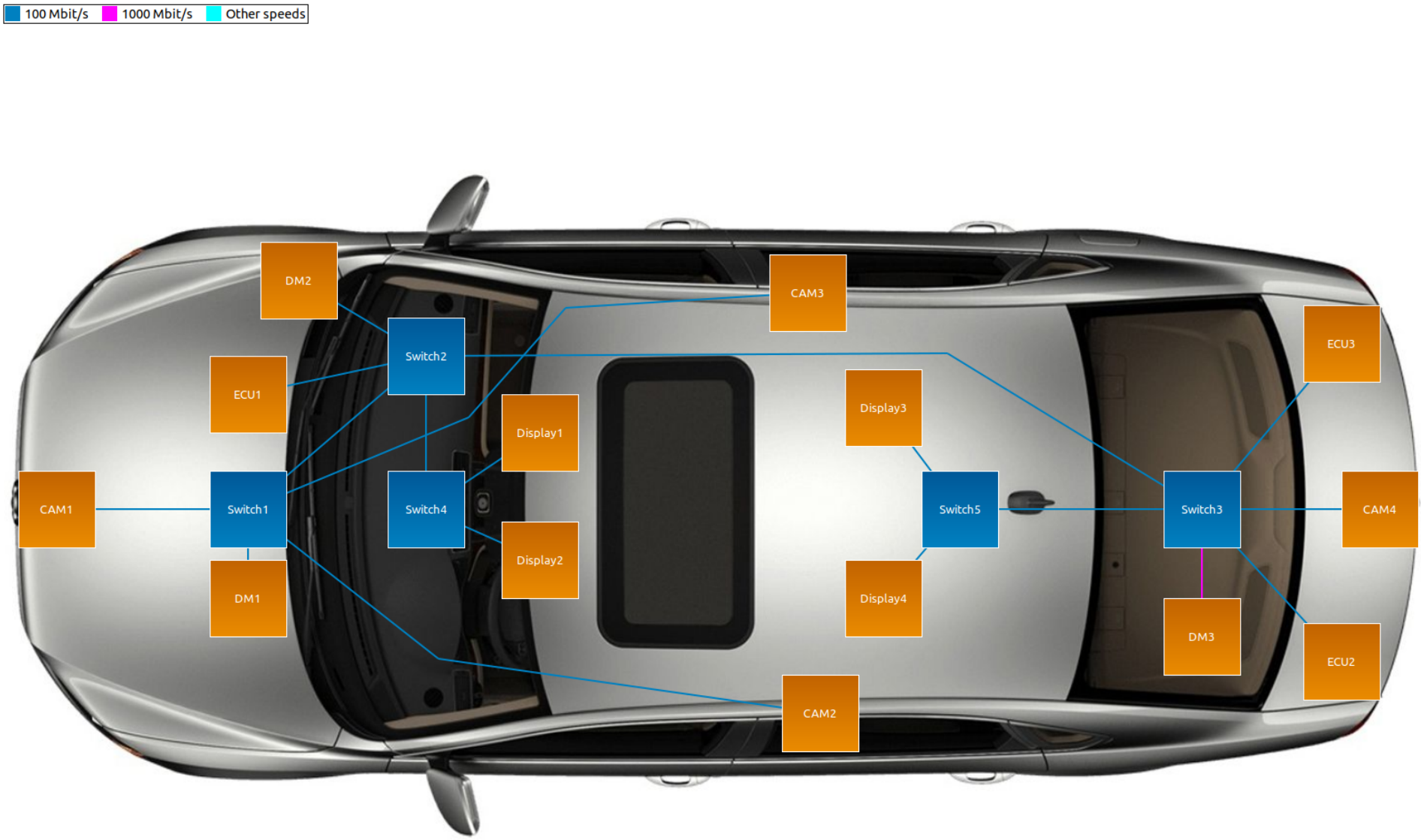
University of Luxembourg

Context of the Work

Collaboration with Nicolas Navet and Tingting Hu.


- In the research group *Critical Real-Time Embedded Systems*.
- Connection with the automotive industry (BMW, Mercedes-Benz, Renault, *etc.*)

Automotive Network



Worst-case Traversal Time (WCTT)

The worst-case traversal time (WCTT) analysis is a formal method guaranteeing the end-to-end delay of network packets.

- Let *asn* be a deployment of services on processors.
- WCTT is a function $f(asn)$ returning true or false if the analysis passes.
- We seek a deployment minimizing cost, maximizing extensibility, *etc.*

Problem: exploring all assignments is too time-consuming.

Constraint Programming

There are several constraints that prune the search space:

- The capacity of each CPU is not exceeded.
- The capacity of each network link is not exceeded.

But this is not enough! We still need to call the WCTT analysis.

How to integrate constraint programming with the WCTT analysis?

Related Work

Controller Area Network (CAN)

- WCTT over CAN network is simple and actually exact.
- Various approaches to tackle this problem such as with genetic algorithm, MIP, constraint programming.
- Both the constraint part and the analysis part are represented in the model.

Only recently Kugele et al. (2021) considered the deployment over Ethernet networks with SMT solvers—but small network (3 CPUs), generate-and-test algorithm, and no constraint model provided.

Contributions

- *In distributed embedded system:* A multi-objective constraint model of the deployment problem over Ethernet network.
- *In constraint programming:* A new multi-objective optimization algorithm CUSOLVE_MO integrating external function calls during solving.
- Proofs of correctness of the algorithms.

Under-Approximating External Function

WCTT External Function

- An assignment $asn \in \text{ASN}$ is a function from services to CPUs.
- The WCTT external function accepts or rejects assignments if they pass the analysis:

$$uf : \text{ASN} \to \{true, false\}$$

- The solutions induced by the external function is:

$$sol(U) := uf^{-1}(true) = \{asn \in \text{ASN} \mid uf(asn) = true\}$$

  where U is the constraint model implicitely represented by uf.

The external function can be slow (1.5s for the WCTT), therefore searching in the whole assignments space is not practical.

Sandwiching the Problem

Modelling the traversal time analysis as a (hypothetical) constraint problem P is too difficult, so we "sandwich" the problem between an under-approximating problem U and an over-approximatin problem O.

- On the one hand, *uf* is *under-approximating* because it discards possible solutions:

$$sol(U) \subseteq sol(P)$$

- On the other hand, we can *over-approximate* P by a constraint model O, accepting assignments that are not solutions:

$$sol(P) \subseteq sol(O)$$

Sandwiching the Problem

Modelling the traversal time analysis as a (hypothetical) constraint problem P is too difficult, so we "sandwich" the problem between an under-approximating problem U and an over-approximatin problem O.

- On the one hand, *uf* is *under-approximating* because it discards possible solutions:

$$sol(U) \subseteq sol(P)$$

- On the other hand, we can *over-approximate* P by a constraint model O, accepting assignments that are not solutions:

$$sol(P) \subseteq sol(O)$$

$$sol(U) \subseteq sol(P) \subseteq sol(O)$$

Over-Approximating Constraint Model

Constraint Programming in a Nutshell

Constraint satisfaction problem (CSP)

A CSP is a pair $\langle d, C \rangle$, example:

$$\langle \{x_1 \mapsto \{1,2,3,4\}, x_2 \mapsto \{2,3,4\}\}, \{x_1 \geq x_2, x_1 \neq 4\} \rangle$$

A solution is $\{x_1 \mapsto 2, x_2 \mapsto 2\}$.

How Does a Constraint Solver Work?

A constraint solving algorithm: propagate and search

- **Propagate**: Remove inconsistent values from the variables' domain.

$$
\begin{array}{ll}
x_1 \geq x_2 & \{x_1 \mapsto \{1,2,3,4\}, x_2 \mapsto \{2,3,4\}\} \\
x_1 \neq 4 & \{x_1 \mapsto \{2,3,4\}, x_2 \mapsto \{2,3,4\}\} \\
x_1 \geq x_2 & \{x_1 \mapsto \{2,3\}, x_2 \mapsto \{2,3,4\}\} \\
x_1 \neq 4 & \{x_1 \mapsto \{2,3\}, x_2 \mapsto \{2,3\}\} \\
x_1 \geq x_2 & \{x_1 \mapsto \{2,3\}, x_2 \mapsto \{2,3\}\}
\end{array}
$$

- **Search**: Divide the problem into (complementary) subproblems explored using *backtracking*.
  - Subproblem 1: $\langle\{x_1 \mapsto \{2\}, x_2 \mapsto \{2,3\}\}, \{x_1 \geq x_2, x_1 \neq 4\}\rangle$
  - Subproblem 2: $\langle\{x_1 \mapsto \{3\}, x_2 \mapsto \{2,3\}\}, \{x_1 \geq x_2, x_1 \neq 4\}\rangle$

Constraint solver: propagate and search

A classic solver in constraint programming:

function SOLVE($\langle d, C \rangle$)
    $\langle d', C \rangle \leftarrow \texttt{propagate}(\langle d, C \rangle)$
    if d' is an assignment **then**
        return $\{d'\}$
    else if d' has an empty domain **then**
        return $\{\}$
    else
        $\langle d_1, \ldots, d_n \rangle \leftarrow \texttt{branch}(d')$
        return $\bigcup_{i=0}^{n} \texttt{solve}(\langle d_i, C \rangle)$
    end if
end function

Constraint Model of the Deployment Problem

We define a tuple $\langle d, C \rangle$ where $d : Var \to D$ is a function mapping variable to domains, and C is a set of constraints.

- **Constants**: Set $S = \{s_1, \ldots, s_n\}$ of services and $H = \{h_1, \ldots, h_m\}$ of CPUs.
- **Variables**: $d(s_i) = H$, initially each service s_i can be allocated on any CPU.
- **Constraints**: Ensure the utilization rate of the the processor (function hc) is not exceeded:

$$\forall h \in H, \sum_{s \in d^{-1}(h)} sc(s) \leq hc(h)$$

  where $sc : S \to \mathbb{Z}$ is the CPU utilization of the services.
- A **solution** is a function $d : S \to H$, where each service is allocated on one CPU.

Constraint Model of the Deployment Problem

Multi-objective Optimization Problem

- **Extensibility**: Minimize the maximum utilization rate of a processor:

$$\min \max_{h \in H} \sum_{s \in d^{-1}(h)} sc(s)$$

- **Extensibility**: Same with network link.
- **Cost**: Minimize the number of processors used:

$$\min |d(S)|$$

Algorithms

Two Phase Algorithm: osolve_mo_then_uf

Algorithm

- Compute the Pareto front of the over-approximating model O.
- Filter the solutions of O passing the WCTT analysis.

But OSOLVE_MO_THEN_UF can prune solutions from $sol(U)$, so the Pareto front is not necessarily optimal.

Integrated Algorithm: usolve_mo

function USOLVE_MO(O, uf, $\sqcup$, opt)
  $U \leftarrow \{\}$
  $asn \leftarrow \text{SOLVE}(O)$
  while $asn \neq \{\}$ **do**
    if $uf(asn) = \texttt{true}$ **then**
      $U \leftarrow U \sqcup \{asn\}$
      $O \leftarrow O \wedge opt(asn)$
    else
      $O \leftarrow O \wedge \neg asn$
    end if
    $asn \leftarrow \text{SOLVE}(O)$
  end while
  return U
end function

Conflicts from *uf*

The WCTT analysis returns which communications fail to pass the analysis.

$\Rightarrow$ Generate new constraints based on this information.

$\Rightarrow$ $uf : \text{ASN} \to C$.

Examples of Conflicts

Suppose the communication between the services x and y fail:

- Forbid source (FS): $d(x) \notin \{asn(x), asn(y)\}$
- Forbid target (FT): $d(y) \notin \{asn(x), asn(y)\}$
- Decreasing hops (DH): $|path(d(x), d(y))| < |path(asn(x), asn(y))|$

(In the paper `cusolve_mo` to take into account conflicts that are not over-approximating).

Experiments

Description of the Experiments

Setting

- AMD Epyc ROME 7H12 processor (64 cores, 280W).
- SOLVE implemented by GeCode 6.3.0 in parallel mode with 8 cores (16 threads), timeout 30 minutes.

Experiments

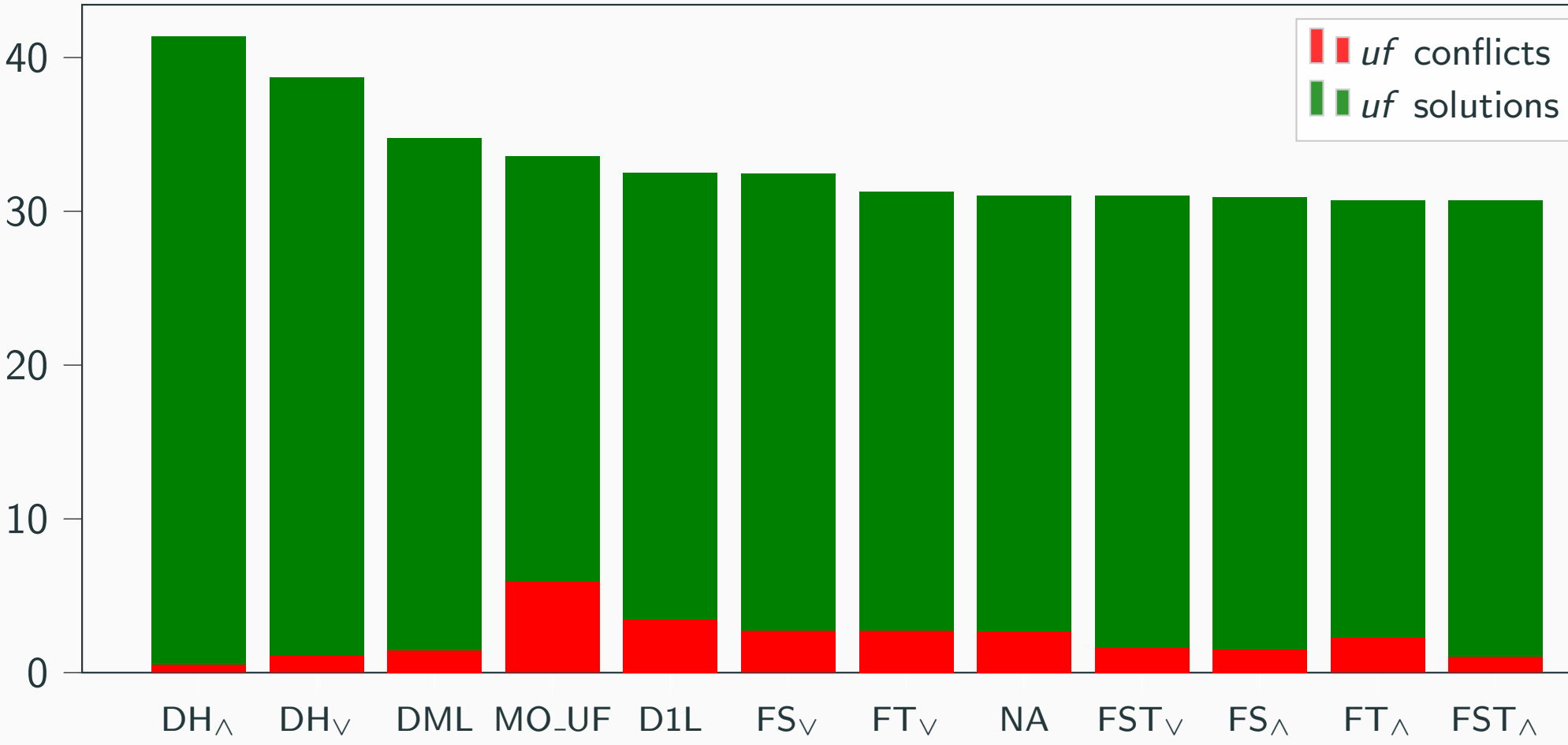
- Instances derived from a realistic automotive Ethernet network consisting of 19 network devices (14 ECUs and 5 switches).
- 5 instances of 50 services, 5 instances of 75 services and 8 instances of 100 services.
- For each of the 18 instances, we generated 10 versions where the sum of all computational requirements is 20%, 40%, 60%, 80% and 90% of the total computational capacity of all ECUs with a uniform distribution among services.

osolve_mo_then_uf

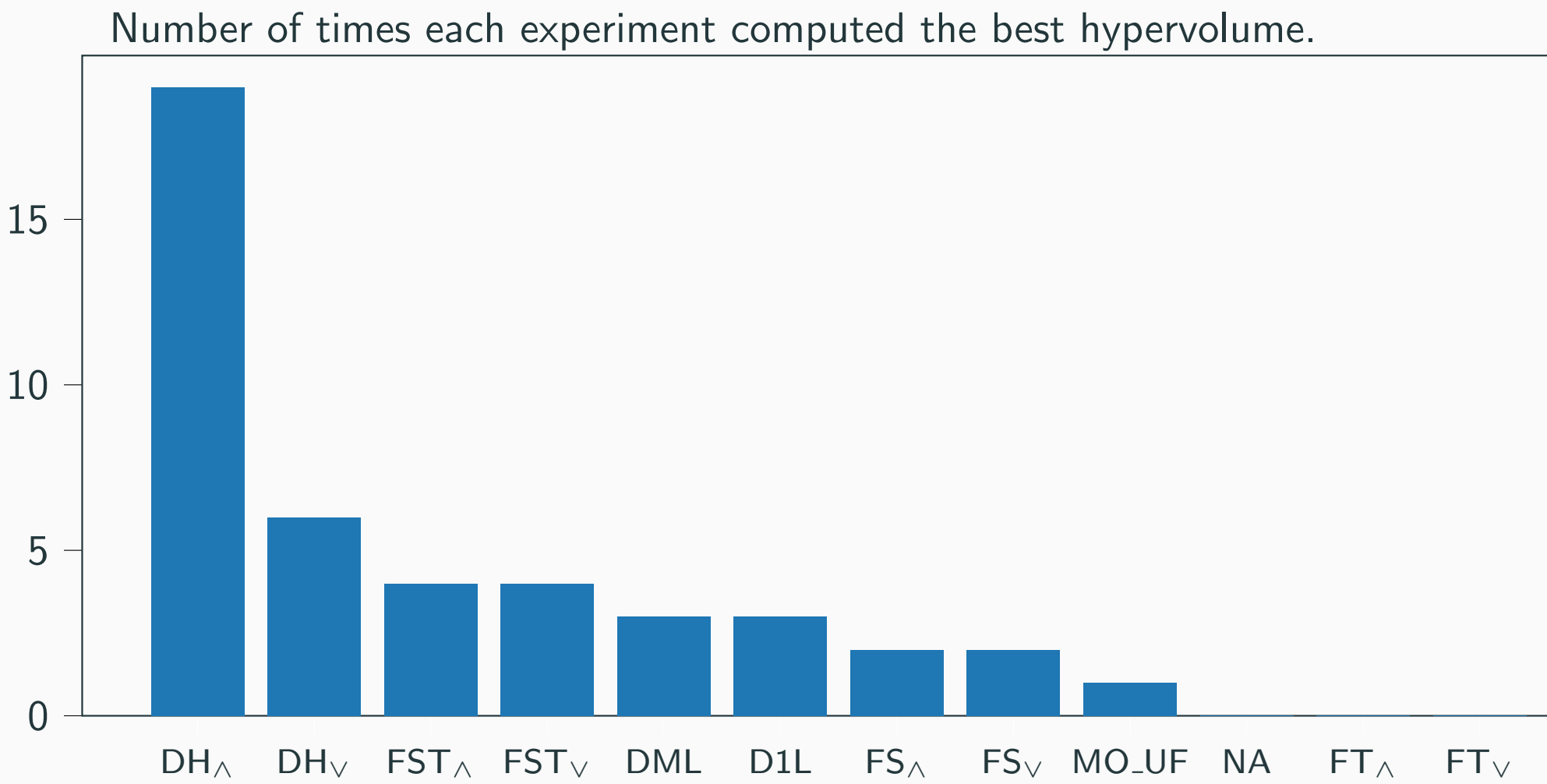
- Low number of services (50): 14/15 instances have the same hypervolume before and after filtering.
- For 75 and 100 services: 24/30 instances with an hypervolume within 3% of the unfiltered hypervolume.
- Still, some instances with filtered hypervolume below 75% of the unfiltered hypervolume.

Cumulated Hypervolume Score

Cumulated hypervolume score for each experiment over all instances.



Best Hypervolume



Conclusion

- Problems from the industry are often unpure.
- Need to reuse existing code, blackbox functions.
- We propose an approach integrating under-approximating blackbox functions in constraint programming.

More in the paper

- Multi-objective algorithm when external function generates conflicts.
- Proofs of correctness of all algorithms.