Constraint Programming with External Worst-Case Traversal Time Analysis

Pierre Talbot
University of Luxembourg, Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust (SnT), Luxembourg

Tingting Hu
University of Luxembourg, Luxembourg

Nicolas Navet
University of Luxembourg, Luxembourg

Abstract

The allocation of software functions to processors under compute capacity and network links constraints is an important optimization problem in the field of embedded distributed systems. We present a hybrid approach to solve the allocation problem combining a constraint solver and a worst-case traversal time (WCTT) analysis that verifies the network timing constraints. The WCTT analysis is implemented as an industrial black-box program, which makes a tight integration with constraint solving challenging. We contribute to a new multi-objective constraint solving algorithm for integrating external under-approximating functions, such as the WCTT analysis, with constraint solving, and prove its correctness. We apply this new algorithm to the allocation problem in the context of automotive service-oriented architectures based on Ethernet networks, and provide a new dataset of realistic instances to evaluate our approach.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computer systems organization → Real-time systems; Networks → Network performance evaluation

Keywords and phrases Constraint programming, external function, multi-objective optimization, network analysis, worst-case traversal time analysis, abstract interpretation

Digital Object Identifier 10.4230/LIPIcs.CP.2023.34

Supplementary Material *Software*: `https://github.com/ptal/automotive-network-cp/tree/cp2023`, archived at `swh:1:dir:e0fe246fdd3c6654293c9f5183cae7782b540d07`

Funding *Pierre Talbot*: This work is supported by the Luxembourg National Research Fund (FNR) – COMOC Project, ref. C21/IS/16101289.

Acknowledgements We are grateful to the reviewers for their detailed comments.

1 Introduction

The hardware architecture of automobiles consists of dozens of interconnected electronic control units (ECUs). An important optimization problem in the field of distributed embedded system, called the *deployment problem*, is to allocate software functions to the ECUs without overloading their compute capacity and overloading the network communication links. Worst-case traversal time analysis (WCTT) is critical to ensure the communications among software functions meet hard deadlines. In most works on the deployment problem, the network considered is a controller area network (CAN), whose operating principles are relatively simple and for which an exact WCTT analysis is available [7, 39]. Therefore, the constraint model can specify both the allocation problem and the WCTT analysis. However, the newest automotive electrical-electronic (E/E) architectures rely on high-speed switched Ethernet networks. WCTT analysis already exists for Ethernet networks but is much harder to model

29th International Conference on Principles and Practice of Constraint Programming (CP 2023).
Editor: Roland H. C. Yap; Article No. 34; pp. 34:1–34:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as a constraint problem, especially considering the wealth of complicated quality-of-service mechanisms available in the time-sensitive networking standards (TSN, see [22]) that are used on top of standard Ethernet.

Constraint programming is a declarative paradigm for solving combinatorial problems. In general, the solvers are designed to work with a closed-world assumption, that is, they do not interact with external entities during solving. However, in practice, there are often parts of the model that are difficult to express as constraints, and are already programmed in another language. This is the case of the WCTT analysis. Industrial constraint solvers such as IBM ILOG CP Optimizer[1] and Local Solver[2] both propose *black-box expressions* to plug external functions in the model. However, these extensions are "last resort solutions" as they do not come with a semantics, and there is no guarantee on when the function is called, and how it is used within the solver. We contribute to a rigorous approach to this problem when the external function is under-approximating, i.e., it only produces valid solutions but not necessarily all.

In this work, we integrate a constraint solver for the allocation problem and a WCTT analyser for the timing constraints. We propose a general framework, based on abstract interpretation [6], for integrating external under-approximating functions (here, the WCTT analyser) in a constraint solving algorithm, and prove its correctness. The under-approximating external function validates each solution produced by the constraint solver. Moreover, when the external function can explain its failure, we dynamically add a new constraint to the constraint model, approximating the reason of the failure of the external function, to improve the quality of the subsequent solutions. In the following, we call these constraints *conflicts*[3]. Because the WCTT analysis is a black-box function, deriving useful *over-approximating conflicts* – which do not remove solutions from the problem, but might accept non-solutions – can be difficult, or even impossible depending on the information provided by the analyser. Our main contribution is to propose cusolve_mo a multi-objective constraint solving algorithm that is *over-approximating* even if the generated conflicts are not over-approximating. This algorithm extends the well-known multi-objective constraint programming algorithm of Gavanelli [13] which has been frequently used in constraint optimization [19, 31, 14]. Further, our framework can be used on top of any constraint solvers. Finally, we contribute to a new set of benchmarks for the deployment problem and evaluate our solving algorithms on them.

2 Service Deployment Problem

For the sake of conciseness, we present the *service deployment problem* in mathematical notation. In Appendix A, we give the constraint model in the MiniZinc constraint modelling language [25]. The MiniZinc model is very close from the mathematical definition given here and does not contain any particular modelling trick.

Let $\langle H, L, hc, lc \rangle$ be a weighted graph where H is a set of hardware units connected by communication links $L \subseteq H \times H$. Moreover, each unit $h \in H$ has a compute capacity $hc(h)$ and each link $\ell \in L$ has a link capacity $lc(\ell)$. This graph represents a network of connected heterogeneous hardware units such as processors and switches.

[1] `https://www.ibm.com/docs/en/icos/20.1.0?topic=2010-cp-optimizer-black-box-expressions`

[2] `https://www.localsolver.com/docs/last/modelingfeatures/externalfunctions.html`

[3] We avoid using the terminology of *nogood* because as we will see later, these conflicting constraints might not always preserve all solutions of the problem (over-approximating).

Let $\langle S, Com, sc, cc\rangle$ be a weighted graph where S is a set of software functions that we call *services* and $Com \subseteq S \times S$ is the set of communications between the services. Each service $s \in S$ consumes a certain amount of computational power $sc(s)$ and for any communication $c \in Com$, $cc(c)$ represents the network utilization due to this communication.

The core of the *service deployment problem* is to find a deployment function $d : S \to H$ allocating each service on a processor. We illustrate this problem in Figure 1 where the software graph (Figure 1b) must be deployed on the hardware graph (Figure 1a) while satisfying a number of constraints – several solutions to this particular instance are shown on Figure 4. The first constraint is on the compute capacity of each processor:

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

It guarantees that the sum of the computational power required by all services allocated on processor h does not exceed the compute capacity of h.

The second constraint is on the communication network:

$$\forall \ell \in L, \sum_{c \in Com} com(c, \ell) \leq lc(\ell)$$

where the function $com(c, \ell)$ returns the cost on the link ℓ of communication c, and is defined by:

$$com(c, \ell) = \begin{cases} cc(c) & \text{iff } \ell \in path(d(x), d(y)),\ c = (x, y) \\ 0 & \text{otherwise} \end{cases}$$

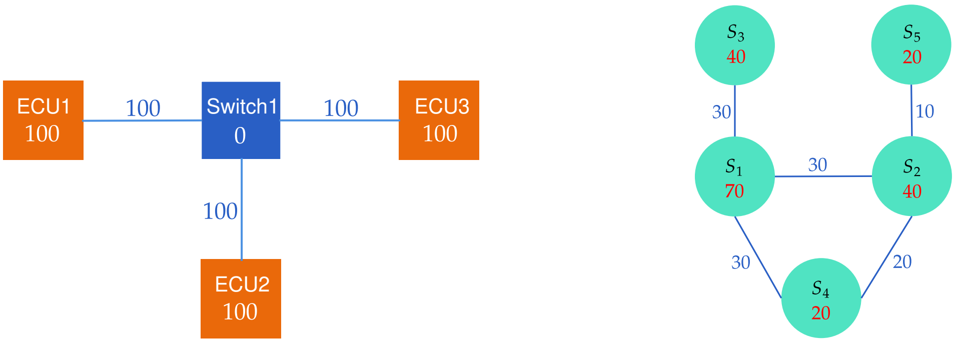
This constraint guarantees that the maximal capacity of a network link ℓ is never exceeded by all communications $c \in Com$ deployed on processors communicating through this link ℓ. The function $path(h_1, h_2)$ returns a path in the hardware graph between two hardware units h_1 and h_2. In our implementation, this function represents the routing table and is given by the user as a parameter of the model. Shortest path is the standard routing strategy in automotive networks, that we use in our experiments. Follow-up work may consider the routing table as a decision variable of the model.

Finally, we note that additional constraints may be considered in similar deployment problems [15, 11]. For instance, a service might need to be allocated on a specific processor (locality constraint) or on the same processor than another service (co-location constraint). For brevity and because these constraints can be taken into account in standard ways, we choose to focus on the core problem presented above.

2.1 Multi-Objective Optimization

In automotive applications, we usually want to find a deployment function d optimizing various objectives such as reliability [21], extensibility and cost reduction [11, 17]. We focus on two new extensibility objectives and a well-known cost reduction objective. Typically, once the services are deployed on the processors, they cannot be moved to other processors. This poses challenges when updating the system with new services. Therefore, an important goal is that the deployment function favours extensibility, that is the ability to add further services over the lifetime of the vehicle. This requirement is captured by two extensibility objectives as follows:

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



(a) Hardware graph with 3 ECUs and 1 switch. The values of hc are displayed in white and those of lc in blue.

(b) Software graph with 5 services and 5 communications. The values of sc are displayed in red and those of cc in blue.

Figure 1 An example of the software deployment problem.

which minimizes the maximum utilization rate of a processor, and

$$\min \max_{\ell \in L}\Big(\sum_{c \in Com} com(c, \ell)\Big)/lc(\ell)$$

which minimizes the maximum utilization rate of a network link. The maximum value is considered as the corresponding hardware resource will be the bottleneck of the system. Indeed, we want to avoid a processor to be fully occupied in case a new service requires to be placed on this processor in a system update, e.g., due to the proximity of a sensor.

At the same time, we want to minimize the number of processors in order to reduce the costs:

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

Of course, this objective directly conflicts with the first one. The tradeoffs between extensibility and cost of the architecture are exposed to the system expert through a Pareto front of solutions. The final decision will involve many factors that are external to the model, such as re-use of existing processors and safety concerns.

3 Constraint Programming with External Function

3.1 Constraint Programming

A constraint satisfaction problem (CSP) is a tuple (X, D, C) where X is a set of variables, $D = D_1 \times \ldots \times D_n$ the sets of values taken by each variable $x_i \in X$, and C a set of relations over variables, called *constraints*. An assignment is a function $asn(x_i) = v_i$ from variable $x_i \in X$ to values where $v_i \in D_i$. Let asn be an assignment and $c \in C$ a constraint defined on the variables $x_1, \ldots, x_n$. Then the constraint c is *satisfied* when $c(asn(x_1), \ldots, asn(x_n))$ holds, or for short $c(asn)$. An assignment asn is a *solution* when each constraint is satisfied. We write $\textsc{asn}$ the set of all assignments. We call the *concrete domain* the powerset lattice $D^\flat = \langle \mathcal{P}(\textsc{asn}), \supseteq \rangle$ where the least element is the set of all assignments $\textsc{asn}$ and the greatest element is the empty set (no solution). The set of solutions of a CSP $P = (X, D, C)$ is an element of $D^\flat$ computed by the following function:

$$sol(P) := \{asn \in \textsc{asn} \mid \forall c \in C,\ c(asn)\}$$

A multi-objective constraint optimization problem $M = (X, D, C, \preceq)$ extends the previous definition with a partial order relation $\preceq$: $\textsc{asn}|_{\textsc{obj}} \times \textsc{asn}|_{\textsc{obj}}$ where $\textsc{asn}|_{\textsc{obj}}$ is the restriction of the assignments[4] to a set of objective variables $\textsc{obj} \subseteq X$. In contrast to a single-objective optimization problem, there can be several solutions such that none is better than the others, which is why we need a partial order. Let $a, b \in \textsc{asn}|_{\textsc{obj}}$. In the case of maximization over integer variables, we can define $a \preceq b \Leftrightarrow \forall x \in \textsc{obj},\ a(x) \leq b(x)$, that is, all the objectives of b are greater or equal to the ones of a. We say that b dominates a when $a \preceq b$. We write $a \succ b$ for $a \neq b \wedge a \succeq b$ and we have $a \succeq b \Leftrightarrow b \preceq a$ and $a \succ b \Leftrightarrow b \prec a$. Importantly, due to the partial order, the fact that a does not dominate b ($a \not\succeq b$) does not imply that a is dominated by b ($a \prec b$).

For the sake of clarity, we overload $\preceq$ to work on arbitrary assignments. For any $a, b \in \textsc{asn}$, we have $a \preceq b \Leftrightarrow a|_{\textsc{obj}} \preceq b|_{\textsc{obj}}$, and similarly for $\prec$, $\succeq$ and $\succ$. In that case, $\preceq$ is a preorder since two assignments with the same objective values might not be equal: $\preceq$ lacks antisymmetry. This is not an issue since antisymmetry can be recovered by considering classes of equivalent assignments, but we do not need this construction here. The solution function for a multi-objective constraint optimization problem is then defined as:

$$\mathit{sol}(X, D, C, \preceq) := \{a \in \mathit{sol}(X, D, C) \mid \forall b \in \mathit{sol}(X, D, C),\ b \not\succ a\}$$

Multi-objective optimization in the context of constraint programming is described in greater length in, e.g., [13, 31, 14].

In the following, we will also need to merge and generate constraints from the Pareto front. The type of a Pareto front is a set of assignments $\mathrm{PF} := \mathcal{P}(\mathrm{ASN})$. We define the operator $\sqcup : \mathrm{PF} \times \mathrm{PF} \to \mathrm{PF}$ merging two Pareto fronts as $A \sqcup B := \{c \in A \cup B \mid \forall d \in A \cup B,\ d \not\succ c\}$. An equivalent definition of the solutions set is possible using $\sqcup$:

$$\mathit{sol}(X, D, C, \preceq) := \bigsqcup \{\{a\} \mid a \in \mathit{sol}(X, D, C)\}$$

where $\bigsqcup\{s_1, \ldots, s_n\} := s_1 \sqcup \ldots \sqcup s_n$.

Finally, we define the function $\mathit{opt} : \textsc{asn} \to C$ which returns a constraint ensuring that for all solutions $a \in \mathit{sol}(X, D, C)$, no solution in $b \in \mathit{sol}(X, D, C \wedge \mathit{opt}(a))$ is dominated by a, i.e. $a \not\succeq b$. This function is defined by:

$$\mathit{opt}(a) := \bigvee_{x \in \textsc{obj}} x < a(x)$$

It generates a constraint requiring at least one of the objective variables to be strictly better than the one obtained in a. This approach to multi-objective optimization was pioneered by Gavanelli [13].

3.2 Abstract Constraint Programming

In general, the set $\mathit{sol}(P)$ might not be efficiently computed on the concrete domain. In constraint reasoning by abstract interpretation [10, 27, 33, 35], they design an abstract solving function $\mathit{sol}^\sharp_o(P)$ which *over-approximates* the solution set, i.e., $\mathit{sol}^\sharp_o(P) \supseteq \mathit{sol}(P)$. Dually, we can also design an *under-approximating* solving function such that $\mathit{sol}^\sharp_u(P) \subseteq \mathit{sol}(P)$. Over-approximation contains all solutions but might contains non-solution assignments as well, while under-approximation only contains solutions but not necessarily all solutions. For

[4] Formally, $\textsc{asn}|_{\textsc{obj}} := \{\mathit{asn}|_{\textsc{obj}} \mid \mathit{asn} \in \textsc{asn}\}$ where $\mathit{asn}|_{\textsc{obj}}(x) = \mathit{asn}(x)$ for all $x \in \textsc{obj}$.

instance, discrete constraint programming solvers are both under- and over-approximating [10, 35], and continuous constraint programming solvers are over-approximating [27]. Incomplete discrete solvers, such as those based on local search, can be viewed as under-approximating solving functions.

3.3 Abstract Constraint Model

We can also use the abstraction framework at the level of the constraint model. In industry, some elements of a constraint model might already be available and tested, and it is usually not practical to spend time redeveloping those parts as a constraint problem. Sometimes, the problem is just too difficult to be expressed as a constraint model in a reasonable amount of time; this is the case of the WCTT analysis for instance. In these cases, the problem P is never explicitly written as a constraint model. Instead, we can rely on an over-approximating model O of P, such that $sol(O) \supseteq sol(P)$. We often have an idea of some constraints that must be satisfied in any solution of the model but we do not necessarily know them all. This model O can be solved by an over-approximating function $sol^\sharp_o(O) \supseteq sol(O)$ – although O simplifies P, it might still not be efficiently computable. If O is unsatisfiable ($sol^\sharp_o(O) = \{\}$), then the problem P is unsatisfiable as well since only $sol(P) = \{\}$ satisfies $sol^\sharp_o(O) \supseteq sol(P)$. In the following, we denote $\textsc{osolve}(O) \in sol^\sharp_o(O)$, the solving algorithm computing a single solution of O and returning $\{\}$ if O is unsatisfiable. Its definition in terms of abstract interpretation can be found in [1, 10, 27]. Dually, we can also propose an under-approximating model U of P and its solving function $sol^\sharp_u(U)$ such that $sol^\sharp_u(U) \subseteq sol(U) \subseteq sol(P)$. If U is satisfiable, then the problem P is satisfiable as well. An under-approximation makes additional assumptions about the reality, and therefore might discard solutions of P. Therefore, the real problem P is framed between an over-approximating model O and an under-approximating model U, which is summarized by $sol^\sharp_o(O) \supseteq sol(O) \supseteq sol(P) \supseteq sol(U) \supseteq sol^\sharp_u(U)$.

3.4 Under-Approximating External Function

We must go one step further for our abstract framework to be useful in practice. If we cannot explicitly list the constraints of the concrete problem P, it seems unlikely that we could list *more constraints* in an under-approximating model U. Nevertheless, when given a solution to O, it can often be validated by existing code developed by domain experts. For instance, worst-case analysis such as WCTT and feasibility tests fall in this category. They conservatively analyse the network architecture, and discard some solutions that would be valid but could not be proven valid by the analysis. In practice, worst-case analysis are not directly working with constraints and domains, and thus do not explicitly define an under-approximating constraint model U, but they work on assignments.

We formally define the analysis as an under-approximating function $uf : \textsc{asn} \rightarrow C$. The function uf returns a conflict constraint when the assignment generated by $\textsc{osolve}$ is not in $sol(U)$, or *true* if it is in $sol(U)$. Actually, we define the solutions of the under-approximating model U as the set of all solutions accepted by uf:

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

A general conflict automatically available to all functions uf, is the logical negation of the assignment (**NA**): $\neg asn := \neg(x_1 = asn(x_1) \wedge \ldots \wedge x_n = asn(x_n)) \Leftrightarrow x_1 \neq asn(x_1) \vee \ldots \vee x_n \neq asn(x_n)$. However, it is a weak conflict since it only prevents $\textsc{osolve}$ from returning to this assignment, without providing additional pruning. A conflict is *over-approximating* if it does not remove valid solutions: for all assignments asn, we have $sol(U) \subseteq sol(O \wedge uf(asn))$.

```
function USOLVE($O$, $ufo$)
    $S \leftarrow \{\}$
    $asn \leftarrow$ OSOLVE($O$)
    while $asn \neq \{\}$ do
        if $ufo(asn) =$ true then
            $S \leftarrow S \cup \{asn\}$
            $O \leftarrow O \wedge \neg asn$
        else
            $O \leftarrow O \wedge ufo(asn)$
        end if
        $asn \leftarrow$ OSOLVE($O$)
    end while
    return $S$
end function
```

(a) Find all satisfiable solutions.

```
function USOLVE_MO($O$, $ufo$, $\sqcup$, $opt$)
    $F \leftarrow \{\}$
    $asn \leftarrow$ OSOLVE($O$)
    while $asn \neq \{\}$ do
        if $ufo(asn) =$ true then
            $F \leftarrow F \sqcup \{asn\}$
            $O \leftarrow O \wedge opt(asn)$
        else
            $O \leftarrow O \wedge ufo(asn)$
        end if
        $asn \leftarrow$ OSOLVE($O$)
    end while
    return $F$
end function
```

(b) Multi-objective version of USOLVE.

Figure 2 Constraint solving with external under-approximating function producing over-approximating conflicts.

We say that a conflict c is *sound* if it implies **NA**; in other terms, it excludes the current assignment: $asn \notin sol(c)$. **NA** is a sound over-approximating conflict. We denote by *ufo* an under-approximating external function returning sound over-approximating conflicts.

Let O be an over-approximating model and *ufo* a sound under-approximating external function. The function USOLVE presented in Algorithm 2a constructs the solutions set S of the under-approximating model U. Constraint programming helps us navigating in the under-approximated solution space of the external function more efficiently. Without it, we would need to call *ufo* on many more unsatisfiable assignments, since those would not be removed by a constraint solver. The next proposition shows that USOLVE computes an under-approximation of P.

▶ **Proposition 1.** *USOLVE is a sound under-approximating function, that is,* USOLVE$(O, ufo) = ufo^{-1}(true) \subseteq sol(P)$.

Proof. Let O_i and S_i be the variables O and S at the ith iteration of the loop where $O_0 = O$ and $S_0 = \{\}$. We must show that at the final iteration n, we have $ufo^{-1}(true) = S_n$. We proceed inductively by defining O_{i+1} and S_{i+1} as follows:

1. If asn is a solution to ufo: $O_{i+1} = O_i \wedge \neg asn$ and $S_{i+1} = S \cup \{asn\}$. In that case we have $sol(O_i) \cup S_i = sol(O_{i+1}) \cup S_{i+1}$.
2. If asn is not a solution to ufo: $O_{i+1} = O_i \wedge ufo(asn)$ and $S_{i+1} = S$. In that case we have $sol(O_i) \cup S_i \supseteq sol(O_{i+1}) \cup S_{i+1}$. Since the conflict must be over-approximating, no assignment removed from O_i is in $ufo^{-1}(true)$ and therefore $ufo^{-1}(true) \subseteq sol(O_{i+1}) \cup S_{i+1}$.

The final iteration is necessarily with $O_n = \{\}$, hence we must have $ufo^{-1}(true) \subseteq S_n$. Since we only add in S_n the assignments asn such that $ufo(asn) = true$, we also have $ufo^{-1}(true) \supseteq S_n$. ◀

The extension to multi-objective optimization is a small modification of USOLVE. The set F represents the Pareto front of the problem. To compute the Pareto front, we introduce the algorithm USOLVE_MO in Figure 2b, and we highlight the differences with USOLVE in green.

▶ **Proposition 2.** *USOLVE_MO is a sound under-approximating function. Moreover, we have:*

1. $\textsc{usolve\_mo}(O, \mathit{ufo}, \sqcup, \mathit{opt}) = \mathit{sol}(U, \preceq)$,
2. $\textsc{usolve\_mo}(O, \mathit{ufo}, \sqcup, \mathit{opt}) \subseteq \textsc{usolve}(O, \mathit{ufo})$, *and*
3. $\textsc{usolve\_mo}(O, \mathit{ufo}, \sqcup, \mathit{opt}) = \{\} \Leftrightarrow \textsc{usolve}(O, \mathit{ufo}) = \{\}$.

Proof. We prove each statement in turn:

1. Each time we reach a solution asn, we add the constraint $opt(asn)$ to O. By definition, all solutions removed by this constraint are dominated by asn, and therefore it cannot remove solutions from $sol(U, \preceq)$.
2. Notice that $opt(asn) \Rightarrow \neg asn$, and therefore it can only prune more solutions in comparison to USOLVE.
3. Before reaching the first solution, the algorithm behaves in the same way than USOLVE. ◀

The risk when navigating in an under-approximating solution space is to find no solution at all. Therefore, it is useful to notice that the multi-objective algorithm returns an empty set of solution only if USOLVE does as well (by Proposition 2(3)).

4 Worst-Case Traversal Time Analysis

We focus on the worst-case traversal time (WCTT) analysis, which verifies if the network communications among the deployed services meet timing constraints, i.e., deadline constraints in this work. In an automotive network, we must ensure the deadlines of network packets are met, which is crucial for safety reasons (e.g., a message sent to an airbag arrives on time) and other non-functional requirements (e.g., the speakers must be synchronized when playing music). WCTT analysis is a formal method which provides upper bounds on the worst-case delay of every packet sent in the network. It is therefore an under-approximating external function because all deployments passing this analysis will also fulfill the real-time constraints in reality. But some deployments, that in fact meets all timing constraints, will not pass the WCTT analysis because it only gives an upper-bound on the delay: it is a sufficient but not necessary condition.

The WCTT analysis for Ethernet networks, based on *network calculus* [18], is mathematically complicated (see for instance [28]). We think it would take tremendous efforts to model WCTT analysis as a constraint problem if the goal is to develop an implementation that is sufficiently accurate to be used on real-world problems, given the complexity of the WCTT analysis after 30 years of research. As an illustration, the network calculus engine from the company RTaW we use in the paper has been developed for 15 years and implements state-of-the-art techniques such as [34, 3]. We take a more pragmatic approach where we reuse an existing WCTT analyser, and integrate it in our framework as an under-approximating external function.

Let us first describe the output of a WCTT analysis. For each communication $(x, y) \in Com$, the WCTT analysis outputs the worst-case end-to-end delay of a packet traversing the network from $d(x)$ to $d(y)$. A negative delay means the deadline for that communication cannot be met and thus the deployment d is unsatisfiable from the point of view of the analysis. From an unsatisfiable assignment, we can think of various conflicts such as forcing the network load of the problematic link to be smaller, or forbidding to allocate the services x or y on their current processors. Unfortunately, conflicts that are intuitive are often not over-approximating. To complicate the finding of over-approximating conflicts, the WCTT analysis is non-monotonic w.r.t. the network load. Indeed, it can happen that increasing the load of an unsatisfiable network turns it into a satisfiable network according to the WCTT analysis. This is a well-known phenomenon referred to as *timing anomaly* (see, e.g., [23]), that may occur with non-preemptive scheduling as in communication networks.

```
function CUSOLVE_MO($F$, $O$, $C$, $uf$, $\sqcup$, $opt$)
    $asn \leftarrow$ OSOLVE($O \wedge C$)
    if $asn \neq \{\}$ then
        $co \leftarrow uf(asn)$
        if $co =$ true then
            $F \leftarrow F \sqcup \{asn\}$
            $O \leftarrow O \wedge opt(asn)$
            CUSOLVE_MO($F$, $O$, $C$, $uf$, $\sqcup$, $opt$)
        else
            CUSOLVE_MO($F$, $O$, $C \wedge co$, $uf$, $\sqcup$, $opt$)
            CUSOLVE_MO($F$, $O$, $C \wedge \neg co \wedge \neg asn$, $uf$, $\sqcup$, $opt$)
        end if
    end if
    return $F$
end function
```

Figure 3 Multi-objective constraint solving with under-approximating external function returning conflicts that are not over-approximating.

Although the conflicts mentioned above are not over-approximating, they can nevertheless be useful as heuristics to find a solution faster. In Figure 3, we extend USOLVE_MO in the case where uf is returning sound conflicts that are not over-approximating. The intuition is that the conflict is viewed as a branching decision, and thus backtracked when the sub-problem has been fully explored. Doing so, we do not lose the completeness of our solving algorithm. We provide an example unrolling this algorithm in Section 4.2.

We note that if the conflict co is over-approximating, then $O \wedge \neg co$ is necessarily unsatisfiable. In that case, CUSOLVE_MO remains correct but the second recursive call CUSOLVE_MO($F, O, C \wedge \neg co, uf, \sqcup, opt$) is unnecessary.

▶ **Proposition 3.** *CUSOLVE_MO*$(\{\}, O, \{\}, uf, \sqcup, opt) = sol(U, \preceq)$

Proof. The difference with Proposition 2, is that uf does not necessarily produce over-approximating conflicts. However, given any constraint problem O and any constraint co, we always have $sol(O \wedge co) \cup sol(O \wedge \neg co) = sol(O)$. Therefore, the same solution space is eventually explored, and the same Pareto front is found. However, the conflicting assignment asn might be reexplored in the right branch. Indeed, $sol(\neg co) = \text{ASN} \setminus sol(co)$ and therefore $asn \in sol(\neg co)$. By forbiding revisiting this assignment in both the left and right branches, we guarantee progress and thus termination of the algorithm. ◀

4.1 Conflicts for WCTT

Let asn be the current assignment for which WCTT detected that the communication $(x, y) \in Com$ does not meet its deadline. Among the possible conflicts, we experiment with the following ones in the next section:

- Forbid the source service x to be allocated on its current hardware unit or the one of y:

$$d(x) \notin \{asn(x), asn(y)\} \qquad \textbf{(FS)}$$

- Forbid the target service y to be allocated on its current hardware unit or the one of x:

$$d(y) \notin \{asn(x), asn(y)\} \qquad \textbf{(FT)}$$

- Decrease the number of hops in the network between x and y:

$$|path(d(x), d(y))| < |path(asn(x), asn(y))| \qquad \textbf{(DH)}$$

We also test two conflicts that are global to the network, thus do not consider a specific communication. It is based on the observation that a communication can fail to meet its deadline because of *other* communications. We let the variable $load_\ell = \sum_{c \in Com} com(c, \ell)$ to be the network load of the link ℓ in the current assignment asn.

- Decrease the load of at least one network link:

$$\bigvee_{\ell \in L} \sum_{c \in Com} com(c, \ell) \leq load_\ell \qquad \textbf{(D1L)}$$

- Decrease the load of the most occupied network link, with $m = maxarg_{\ell \in L}(load_\ell / lc(\ell))$:

$$\sum_{c \in Com} com(c, m) \leq load_m \qquad \textbf{(DML)}$$

Taking the conjunction of any two conflicts will further prune the search tree, while taking their disjunction will create a weaker conflict. In the experiments, we test the conjunction of **FS** and **FT** that we name **FST**. In our investigations, we found that, in general, the disjunction of conflicts did not help to reach a (better) solution faster.

4.2 An Example of the Algorithm cusolve_mo

We unroll the algorithm CUSOLVE_MO with the **DH** conflict strategy on the software deployment problem given in Figure 1. Initially, the Pareto front F and the set of conflicts C are empty. The first call to OSOLVE returns a solution to the problem as depicted in the orange box labelled A. Because one of the network link is used at 100% capacity, we suppose the communication between S_1 and S_4 fails to meet its deadline, hence the WCTT analysis fails on this solution. The conflict $|path(d(S_1), d(S_4))| < 1$, reducing the number of hops between S_1 and S_4, is added to the conflicts set C. In this case, this conflict forces both services to be allocated on the same processor.

The model is solved again with this new conflict and OSOLVE returns a solution as depicted in the box B. This time the WCTT analysis succeeds, and the solution is added to the Pareto front F. The objectives are $(100, 80, 2)$ where 100 is the maximum utilization rate among all processors, 80 is the maximum utilization among all network links and 2 is the number of cores used.

As long as the WCTT analysis succeeds, the OSOLVE procedure is iteratively called with the updated Pareto front. In the box C, we have a solution $(90, 90, 3)$ incomparable to $(100, 80, 2)$, hence the Pareto front now contains both. In the box D, we find the solution $(90, 80, 3)$ which dominates the previous one. Afterwards, the OSOLVE procedure finds the problem unsatisfiable, which means there is no solution better than the ones found previously. However, we have previously added a conflict which was not over-approximating, and therefore we might have missed solutions of the problem. Therefore, we need to backtrack and explore the problem with the negation of the conflict. In box E, the model is solved again with the latest Pareto front, and a new non-dominated solution $(80, 90, 3)$ is found. This is repeated and a better solution $(70, 90, 3)$ is found in box F. As long as the WCTT analysis succeeds, the solving procedure continues, and when it fails we branch as we did in the first node.

This example demonstrates that conflicts are heuristics which are used in a way that do not prevent to find all solutions. This is also why the non-monotonicity of the WCTT analysis is not an issue: the entire solution space is eventually explored.

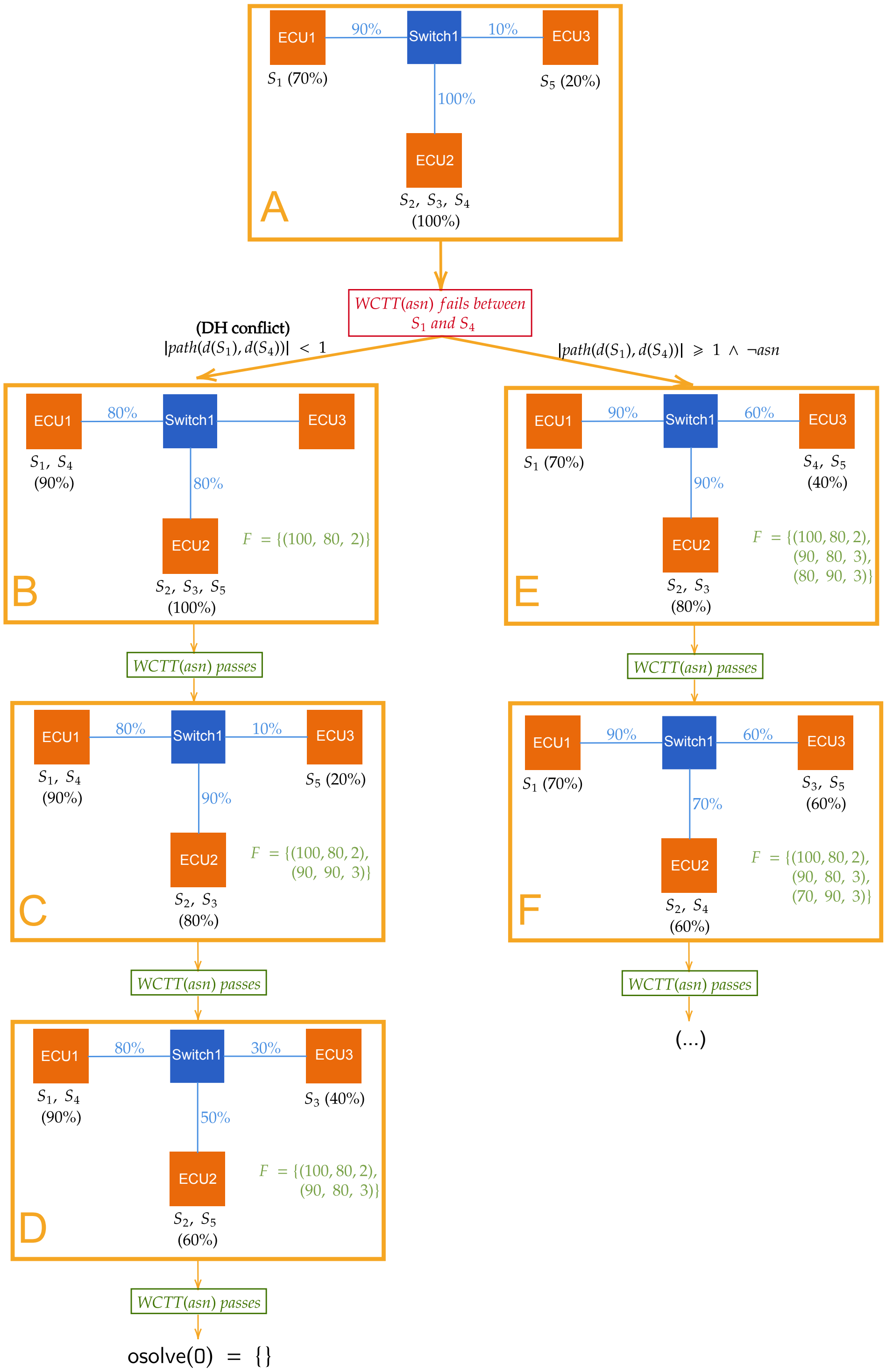


Figure 4 An example unrolling CUSOLVE_MO on a small network. The orange boxes represent the solutions found by OSOLVE. F is the current Pareto front and is automatically added to the model before solving; note that the Pareto front is preserved on backtracking.

5 Implementation and Experiments

The code of the MiniZinc model, the data of the instances and the implementation of the algorithms used can be found online at `https://github.com/ptal/automotive-network-cp/tree/cp2023`.

5.1 Experimental Setting

We run all the experiments on an AMD Epyc ROME 7H12 processor (64 cores, 280W). The constraint programming solver implementing the osolve solving function is Gecode 6.3.0 [32] in parallel mode with 8 cores and 16 threads. We use a first-fail variable selection strategy (the variable with the smallest domain is chosen first) and a random value selection, as it shows better performance than the free search strategy of Gecode. We also tried Chuffed 0.10.4 [26], a hybrid solver between SAT and constraint programming, but it did not outperform Gecode on our problem. Due to the lack of open-source alternative, the WCTT analysis is performed by the proprietary software RTaW-Pegase-4.3.7 [29], which implements state-of-the-art network calculus algorithms [18, 2, 4]. The WCTT analysis takes on average 1.5 seconds to run, and this time remains stable across instances. The algorithms presented in this paper – usolve, usolve_mo and cusolve_mo – are implemented in Python using MiniZinc Python 0.9.0 [9]. In addition we provide osolve_mo which implements the multi-objective optimization solving procedure of [13] – it is the same than usolve_mo but without the external function filtering. Although multi-objective optimization is very important in practice, it is not natively available in every constraint programming solver (for instance in Gecode, Chuffed or ortools). Similarly to [14], our approach does not require to modify the constraint solver, but the solver state is lost between two calls to osolve which might be less efficient – although the issue is mitigated since we use a random search strategy. It can be seen as a restart strategy triggered on every solution. It is also similar to what is done in MiniSearch [30] to design search strategies generically across solvers.

5.2 Dataset Description

The instances are derived from a realistic automotive Ethernet network, shown in Figure 5, consisting of 19 network devices (14 ECUs and 5 switches) provided by the company RealTime-at-Work[5]. The experiments consider 5 problem instances of 50 services, 5 instances of 75 services and 8 instances of 100 services. The numbers of communications vary among the instances, but are between 125% and 135% of the number of services. An information missing in the network description is the CPU usage for each service. 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. To summarize, an instance named `I5_75-14-u60` has 75 services allocated on 14 processors and using 60% of the total computational power, and `I5` denotes the fifth instance with 75 services. In total, we have a new dataset of 90 MiniZinc instances for the deployment problem. In the following, we present experimental results for a subset of these instances (`I{1,2,5}_{50,75,100}-14-u{20,40,60,80,90}`), totalizing 45 instances. We set a timeout on the constraint solver of 30 minutes for each instance, and unlimited time for the WCTT analysis.

[5] `https://www.realtimeatwork.com/`

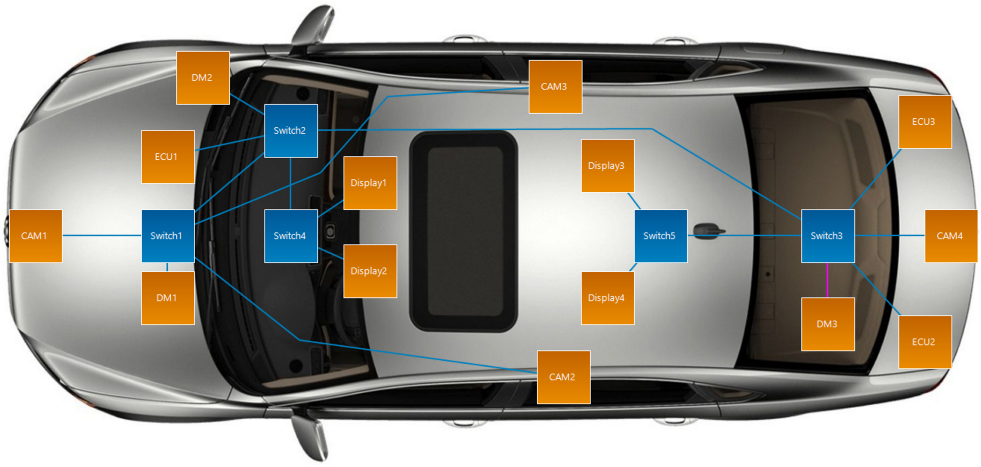


Figure 5 Realistic automotive Ethernet network used in the experiments.

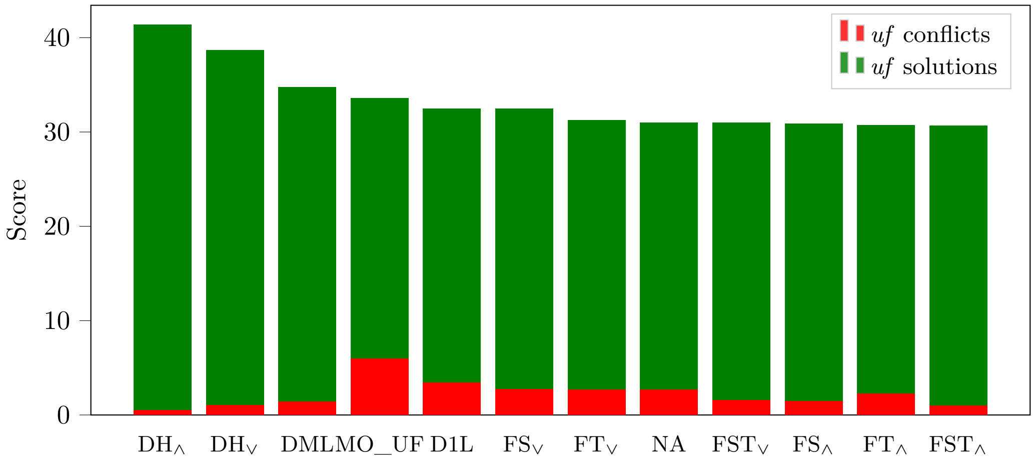


Figure 6 Cumulated hypervolume score for each experiment over all instances.

5.3 Evaluation of cusolve_mo

We evaluate the algorithm cusolve_mo on **NA** and the seven strategies presented in Section 4.1. For a single assignment, it is possible that several communications cannot meet their deadlines, and thus several conflicts are generated. We write **FS**$_\vee$ when these conflicts are combined disjunctively and **FS**$_\wedge$ when they are combined conjunctively. It only impacts the conflicts that are local to a communication (**FS**, **FT**, **DH** and **FST**), thus we have 11 conflicts in total.

Our main comparison metrics is the hypervolume of the Pareto front which is standard in multi-objective optimization. For all 45 instances, none of the algorithms tested could find the optimum within the time limit. We give a general picture of the situation in Figures 6 and 7. Overall, the decreasing hops strategy is the best, and finds the best hypervolumes on 19 of the 45 instances. We also witness a smaller number of conflicts, which means that **DH** is effective to search the state-space of $sol(U)$. When considering the score, forbidding the

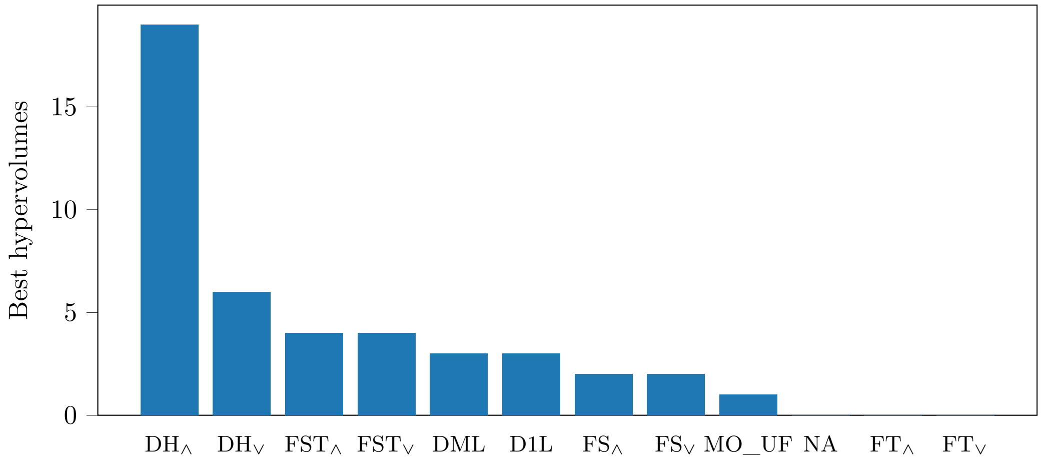


Figure 7 Number of times each experiment computed the best hypervolume.

source and target services on a particular ECU are usually not better than simply using the **NA** conflict. These strategies are still superior regarding the number of times they find the best hypervolume.

In addition, we evaluate CUSOLVE_MO against a more straightforward *two-steps* algorithm OSOLVE_MO_THEN_UF (denoted by **MO_UF**) where the Pareto front is first fully generated, and then filtered by the uf function. To improve this method, we keep all intermediate solutions when building the Pareto front in a set S. During the filtering step, if a solution a is discarded by the function uf, we remove a from the Pareto front and reconstruct it with $\bigsqcup\{\{b\} \mid b \in S \setminus \{a\}\}$. It does not make this algorithm over-approximating as it can still discard assignments accepted by uf, but it improves the filtered Pareto front.

As shown in Figure 6, **MO_UF** ranks fourth, and therefore is a good approach to solve the deployment problem when we do not seek (or cannot find) the true optimal solution. Interestingly, for 18 instances over the 45, the hypervolume before and after filtering is the same, which means that all solutions of the Pareto front were valid w.r.t. uf. This is particularly true with 50 services where 14/15 instances have the same hypervolume before and after filtering. This result is explained by noticing that adding more services has a higher impact on the network load, and thus the WCTT analysis fails more often. Over the 30 instances with 75 and 100 services, there are 24 instances that have a filtered hypervolume within 3% of the unfiltered hypervolume. It is not always the case as for the instances `I1_100-14-u60` and `I1_100-14-u80`, the filtered hypervolume is respectively 57% and 73% of the unfiltered hypervolume. An advantage of the offline filtering proposed by **MO_UF** is to call uf an order of magnitude less than with **DH**. Over all instances, **MO_UF** calls uf 1180 times while **DH** calls uf 12245 times. Therefore, depending on the time taken by the external function and the number of conflicts, **MO_UF** can be better – especially for low number of conflicts and long evaluation time.

From an implementation perspective, using MINIZINC PYTHON allowed us to implement an algorithm generic across solvers, but it incurs a cost. Besides losing the state of the solver between calls, we must call the MINIZINC (source of the model) to FLATZINC (simpler format supported by solvers) translator, and it takes on average around 40% of the total solving time. An improvement to MINIZINC PYTHON would be to directly add FLATZINC constraints to avoid recompiling the MINIZINC model each time.

6 Related Work

6.1 CAN Networks

The deployment problem has been extensively studied due to its importance in distributed real-time embedded systems. It was pioneered in [38] for tasks allocation on *controller area network* (CAN). In CAN network, the hardware units are all connected on a broadcast bus, and therefore the hardware network is fully connected. The main difference with our work is that we consider a more general switch-based network. In the context of real-time and critical systems, such as those found in the automotive industry, it is crucial to ensure the network will not be overloaded by communication, and when required, that the network packet deadlines are met. The WCTT analysis on switch-based networks is more complicated and under-approximating (it does not give an exact upper bound), while it is an exact analysis for CAN network [37, 7, 39].

Due to its simpler nature, schedulability analysis, such as WCTT, over CAN networks has been directly incorporated in the constraint model before. The work of Hladik et al. [15] is the first to model the deployment problem over CAN network using constraint programming. They model the schedulability analysis as a global constraint. Alternatively, they also use a method inspired by logic-based Benders decomposition (LBBD) [16] to separate the allocation problem solved using constraint programming and the schedulability analysis solved by an ad-hoc algorithm. It differs from our approach mainly because there is no notion of approximation, and the conjunction of both parts models the problem exactly. Moreover, they consider only the satisfiability of the problem, and they do not seek to optimize one or more objectives.

Other techniques were proposed to solve the deployment over CAN networks with multi-objective optimization, for instance, evolutionary optimization [21], ant colony system with constraint propagation and without searching [36], mixed integer linear programming (MIP) [24] and satisfiability modulo theories (SMT) [11]. These methods are either incomplete (no proof of optimality or unsatisfiability) and thus under-approximating, or they are complete (MIP and SMT) which is only possible because they model a simpler problem (CAN network).

6.2 Switched Networks

To the best of our knowledge, Kugele et al. [17] are the first to configure and analyse an application distributed over a switched network using a SMT solver. Similarly to OSOLVE_MO, they generate a Pareto front and then verify the produced solutions. However, the verification is performed using simulation, which does not give a formal worst-case guarantee. Therefore, their solving method is over-approximating and the obtained solutions are not guaranteed to be valid. Besides, they do not provide the constraint model and only tested their algorithm on a small network of 3 ECUs and 25 services.

6.3 Other Applications

Campeanu et al. [5] study the deployment problem in heterogeneous architectures (CPU, GPU and FGPA), but with communication still happening over a CAN network. *Satisfiability Modulo Discrete Event Simulation* [20] combines a SAT solver with discrete event simulation (DES) for a railway construction planning problem. The combination of both techniques share similarities with CUSOLVE_MO since the DES simulator is encapsulated as a theory and provide conflict to the SAT solver – but, like us, only on full assignments. However, simulation is an over-approximating technique and therefore the global method remains over-approximating. Moreover, the algorithms are specialized to the railway construction problem and no general algorithm or correctness proof is given.

6.4 Online and Dynamic Constraint Programming

Our work is related to online constraint programming [12] – also called dynamic constraint programming [8] – as in both approaches the model is incrementally refined. A difference is that online constraint programming is primarily designed when the solutions generated are used in real-time, and variables impacting the past decisions cannot be modified in the subsequent solving steps. We do not have such real-time requirements since the new data are obtained from an offline analysis. Moreover, in [12], they propose to internalize the dynamic part of the problem inside the model. Here, we purposely delegated a part of the model to an external function, which would have been prohibitively complicated to model otherwise.

7 Conclusion

We study the deployment problem, an important problem in the field of distributed real-time embedded systems, and more specifically in the automotive industry. This problem has a task allocation part, which is efficiently solved by constraint programming, and a scheduling network analysis part, which is efficiently solved by a WCTT analysis. The integration of both techniques is difficult since both parts are black-box functions. We propose the algorithm CUSOLVE_MO which combines both parts in a loosely coupled manner, thus making our framework reusable on other similar problems. Our approach is based on abstract interpretation, a formal method allowing us to prove properties of our algorithms. Finally, we evaluated our approach on a new dataset for the deployment problem – since none existed before – and conclude that the cooperation scheme proposed by CUSOLVE_MO works better than solving each part in sequence.

References

1 Krzysztof R. Apt. The essence of constraint propagation. *Theoretical computer science*, 221(1-2):179–210, 1999. doi:10.1016/S0304-3975(99)00032-8.

2 Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008. doi:10.1007/s10626-007-0028-x.

3 Marc Boyer and Hugo Daigmorte. Improved service curve for element with known transmission rate. *IEEE Networking Letters*, 5(1):46–49, 2023. doi:10.1109/LNET.2022.3150649.

4 Marc Boyer, Jörn Migge, and Nicolas Navet. An efficient and simple class of functions to model arrival curve of packetised flows. In *Proceedings of the 1st International Workshop on Worst-Case Traversal Time*, WCTT '11, pages 43–50, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/2071589.2071595.

5 Gabriel Campeanu, Jan Carlson, and Severine Sentilles. Component Allocation Optimization for Heterogeneous CPU-GPU Embedded Systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 229–236, Verona, Italy, 2014. IEEE. doi:10.1109/SEAA.2014.29.

6 Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. Association for Computing Machinery. doi:10.1145/512950.512973.

7 Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 35(3):239–272, 2007. doi:10.1007/s11241-007-9012-7.

8 Rina Dechter and Avi Dechter. Belief Maintenance in Dynamic Constraint Networks. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, pages 37–42. AAAI Press, 1988.

9 Jip J. Dekker. MiniZinc Python, 2023. URL: https://github.com/MiniZinc/minizinc-python.

10 Vijay D'Silva, Leopold Haller, and Daniel Kroening. Abstract satisfaction. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '14*, pages 139–150, San Diego, California, USA, 2014. ACM Press. doi:10.1145/2535838.2535868.

11 Johannes Eder, Sebastian Voss, Andreas Bayha, Alexandru Ipatiov, and Maged Khalil. Hardware architecture exploration: automatic exploration of distributed automotive hardware architectures. *Software and Systems Modeling*, 19(4):911–934, 2020. doi:10.1007/s10270-020-00786-6.

12 Alexander Ek, Maria Garcia de la Banda, Andreas Schutt, Peter J. Stuckey, and Guido Tack. Modelling and Solving Online Optimisation Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1477–1485, 2020. doi:10.1609/aaai.v34i02.5506.

13 Marco Gavanelli. An algorithm for multi-criteria optimization in CSPs. In *ECAI 2002: 15th European Conference on Artificial Intelligence, July 21-26, 2002, Lyon France: Including Prestigious Applications of Intelligent Systems (PAIS 2002): Proceedings*, volume 77, page 136. IOS Press, 2002.

14 Tias Guns, Peter J. Stuckey, and Guido Tack. Solution Dominance over Constraint Satisfaction Problems, 2018. arXiv:1812.09207 [cs]. URL: http://arxiv.org/abs/1812.09207.

15 Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. Solving a real-time allocation problem with constraint programming. *Journal of Systems and Software*, 81(1):132–149, 2008. doi:10.1016/j.jss.2007.02.032.

16 J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003. doi:10.1007/s10107-003-0375-9.

17 Stefan Kugele, Philipp Obergfell, and Eric Sax. Model-based resource analysis and synthesis of service-oriented automotive software architectures. *Software and Systems Modeling*, 20(6):1945–1975, December 2021. doi:10.1007/s10270-021-00896-9.

18 Jean-Yves Le Boudec and Patrick Thiran. Network Calculus. In *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, pages 3–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/3-540-45318-0_1.

19 Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Solving Multi-objective Pseudo-Boolean Problems. In *Theory and Applications of Satisfiability Testing – SAT 2007*, pages 56–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-72788-0_9.

20 Bjørnar Luteberget, Koen Claessen, Christian Johansen, and Martin Steffen. SAT modulo discrete event simulation applied to railway design capacity analysis. *Formal Methods in System Design*, 57(2):211–245, August 2021. doi:10.1007/s10703-021-00368-2.

21 Irene Moser and Sanaz Mostaghim. The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation. In *IEEE Congress on Evolutionary Computation*, pages 1–8, Barcelona, Spain, July 2010. IEEE. doi:10.1109/CEC.2010.5585991.

22 Ahmed Nasrallah, Akhilesh S. Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham ElBakoury. Ultra-Low Latency (ULL) Networks: The IEEE TSN and IETF DetNet Standards and Related 5G ULL Research. *IEEE Communications Surveys & Tutorials*, 21(1):88–145, 2019. doi:10.1109/COMST.2018.2869350.

23 Mitra Nasri, Sanjoy Baruah, Gerhard Fohler, and Mehdi Kargahi. On the Optimality of RM and EDF for Non-Preemptive Real-Time Harmonic Tasks. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems - RTNS '14*, pages 331–340, Versaille, France, 2014. ACM Press. doi:10.1145/2659787.2659806.

24 Asef Nazari, Dhananjay Thiruvady, Aldeida Aleti, and Irene Moser. A mixed integer linear programming model for reliability optimisation in the component deployment problem. *Journal of the Operational Research Society*, 67(8):1050–1060, August 2016. doi:10.1057/jors.2015.119.

25 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming—CP 2007*, pages 529–543. Springer, 2007.

26 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via Lazy Clause Generation. *Constraints*, 14(3):357–391, September 2009. doi:10.1007/s10601-008-9064-x.

27 Marie Pelleau, Antoine Miné, Charlotte Truchet, and Frédéric Benhamou. A Constraint Solver Based on Abstract Domains. In *Verification, Model Checking, and Abstract Interpretation*, pages 434–454. Springer, 2013. doi:10.1007/978-3-642-35873-9_26.

28 R. Queck. Analysis of Ethernet AVB for automotive networks using Network Nalculus. In *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, pages 61–67, July 2012. doi:10.1109/ICVES.2012.6294261.

29 RealTime-at-Work. Rtaw-pegase, 2022. URL: https://www.realtimeatwork.com/rtaw-pegase/.

30 Andrea Rendl, Tias Guns, Peter J. Stuckey, and Guido Tack. MiniSearch: a solver-independent meta-search language for MiniZinc. In *Principles and Practice of Constraint Programming*, pages 376–392. Springer, 2015. doi:10.1007/978-3-319-23219-5_27.

31 Pierre Schaus and Renaud Hartert. Multi-Objective Large Neighborhood Search. In *Principles and Practice of Constraint Programming*, volume 8124, pages 611–627. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40627-0_46.

32 Christian Schulte, Guido Tack, and Mikael Lagerkvist. *Modeling and Programming with Gecode*, 2020.

33 Joseph Scott. *Other Things Besides Number: Abstraction, Constraint Propagation, and String Variable Types*. PhD thesis, Acta Universitatis Upsaliensis, Uppsala, 2016. OCLC: 943721122.

34 Seyed Mohammadhossein Tabatabaee, Marc Boyer, Jean-Yves Le Boudec, and Jörn Migge. Efficient and accurate handling of periodic flows in time-sensitive networks. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 303–315, 2023. doi:10.1109/RTAS58335.2023.00031.

35 Pierre Talbot, Éric Monfroy, and Charlotte Truchet. Modular Constraint Solver Cooperation via Abstract Interpretation. *Theory and Practice of Logic Programming*, 20(6):848–863, 2020. doi:10.1017/S1471068420000162.

36 Dhananjay Thiruvady, I. Moser, Aldeida Aleti, and Asef Nazari. Constraint Programming and Ant Colony System for the Component Deployment Problem. *Procedia Computer Science*, 29:1937–1947, 2014. doi:10.1016/j.procs.2014.05.178.

37 Tindell, Hansson, and Wellings. Analysing real-time communications: controller area network (CAN). In *Proceedings Real-Time Systems Symposium REAL-94*, pages 259–263, San Juan, Puerto Rico, 1994. IEEE Comput. Soc. Press. doi:10.1109/REAL.1994.342710.

38 K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-Hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992. doi:10.1007/BF00365407.

39 P. M. Yomsi, D. Bertrand, N. Navet, and R. Davis. Controller Area Network (CAN): Response Time Analysis with Offsets. In *9th IEEE International Workshop on Factory Communication Systems*, pages 43–52, United States, May 2012. IEEE. doi:10.1109/WFCS.2012.6242539.

A MiniZinc Model

We describe the full MiniZinc constraint model implementing the mathematical model given in Section 2. We first give the parameters of the model with the corresponding mathematical notations in blue comments:

```
% I. The hardware graph $\langle H, L, hc, lc \rangle$.
int: locations;
set of int: LOCATIONS = 1..locations;          % $H$
int: num_links;
set of int: NUM_LINKS = 1..num_links;          % $L$
array[LOCATIONS] of int: cpu_capacity;         % $hc$
array[NUM_LINKS] of int: capacity;             % $lc$

% II. The software graph $\langle S, Com, sc, cc \rangle$.
% $Com$ is implicitly represented by the adjacency matrix coms where
    $\mathsf{coms}[s_i][s_j] = 0$ if the services $s_i$ and $s_j$ do not communicate.
int: services;
set of int: SERVICES = 1..services;            % $S$
array[SERVICES] of int: services_cpu_usage;    % $sc$
array[SERVICES, SERVICES] of int: coms;        % $cc$

% III. The path function
% $\mathsf{shortest\_path}[h_i, h_j]$ contains all the edges belonging to the shortest path
    between $h_i$ and $h_j$.
% Interestingly, we do not need to know the order of the edges on the
    shortest path, thus we can use a set.
array[LOCATIONS, LOCATIONS] of set of NUM_LINKS: shortest_path;

% IV. Not part of the mathematical specification: this is to display the
    solutions with locations and services names instead of indexes.
array[LOCATIONS] of string: locations2names;
array[SERVICES] of string: services2names;
```

The decision variable is the function $d : S \to H$ which is modelled as a MiniZinc array:

```
array[SERVICES] of var LOCATIONS: services2locs;  % $d : S \to H$
```

The constraints are defined using intermediate arrays of variables to simplify their definitions.

```
% I. CPU load constraint.
% $\forall h \in H,\ \sum_{s \in d^{-1}(h)} sc(s) \leq hc(h)$
array[LOCATIONS] of var int: cpu_usage;
constraint forall(l in LOCATIONS)
                  (cpu_usage[l] =
                     sum(s in SERVICES)
                        (services_cpu_usage[s] * (services2locs[s] == l)))
    ;
constraint forall(l in LOCATIONS)(cpu_usage[l] >= 0 /\ cpu_usage[l] <=
    cpu_capacity[l]);

% II. Network load constraint.
% $\forall \ell \in L,\ \sum_{c \in Com} com(c, \ell) \leq lc(\ell)$
array[NUM_LINKS] of var int: slack;
constraint forall(link in NUM_LINKS)(
            slack[link] = capacity[link] -
              sum(s1,s2 in SERVICES)(
                coms[s1,s2] * (link in shortest_path[services2locs[s1],
    services2locs[s2]])
              ));
% Then we ensure the slack is always greater or equal to 0.
constraint forall(link in NUM_LINKS)(slack[link] >= 0 /\ slack[link] <=
    capacity[link]);
```

The multi-objective aspect of the problem is not treated within the MiniZinc model itself, but by the MiniZinc Python interface. To communicate which objectives we seek to minimize, we use a special array variable `objs` describing all three objectives described in Section 2.1.

```
array[1..3] of var int: objs;

% $\min \max_{h \in H} \sum_{s \in d^{-1}(h)} sc(s)$
constraint objs[1] = max(l in LOCATIONS)(cpu_usage[l]);

% $\min \max_{\ell \in L} (\sum_{c \in Com} com(c, \ell)) / lc(\ell)$
% We use an intermediate array charge and channeling constraint to
    represent the charge of a link in percentage.
array[NUM_LINKS] of var 0..100: charge;
constraint forall(link in NUM_LINKS)(charge[link] == (capacity[link] -
    slack[link]) div (capacity[link] div 100));
constraint objs[2] = max(link in NUM_LINKS)(charge[link]);

% $\min |d(S)|$
constraint objs[3] = sum(l in LOCATIONS)(cpu_usage[l] > 0);
```