28th International Conference on
Automated Planning and Scheduling

June 24-29, 2018, Delft, the Netherlands

COPLAS 2018

Proceedings of the 13th Workshop on

Constraint Satisfaction Techniques for Planning and Scheduling

Edited by:

Miguel A. Salido,
Roman Barták.

Program Committee

Federico Barber
Universitat Politecnica de Valencia, Spain

Roman Barták
Charles University, The Czech Republic

Amedeo Cesta
ISTC-CNR, Italy

Minh Binh Do
NASA Ames Research Center, USA

Serdar Kadioglu
Fidelity Investments, USA

Eva Onainda
Universitat Politecnica de Valencia, Spain

Massimo Paolucci
University of Genova, Italy

Gilles Pesant
cole Polytechnique de Montral, Canada

Nicola Policella
European Space Agency, Germany

Enrico Pontelli
New Mexico State University, USA

Hana Rudová
Masaryk University, The Czech Republic

Miguel A. Salido
Universitat Politecnica de Valencia, Spain

Pierre Schaus
Universit catholique de Louvain, Belgian

Dunbing Tang
Nanjing University of Aeronautics and Astronomics, China

Ramiro Varela
Universidad de Oviedo, Spain

Petr Vilím
ILOG, France

Neil Yorke-Smith
Delft University of Technology, the Netherlands

Foreword

The areas of AI planning and scheduling have seen important advances thanks to the application of constraint satisfaction models and techniques. Especially solutions to many real-world problems need to integrate plan synthesis capabilities with resource allocation, which can be efficiently managed by using constraint satisfaction techniques.

The workshop aims at providing a forum for researchers in the field of Artificial Intelligence to discuss novel issues on planning, scheduling, constraint programming/constraint satisfaction problems (CSPs) and many other common areas that exist among them. On the whole, the workshop mainly focuses on managing complex problems where planning, scheduling and constraint satisfaction must be combined and/or interrelated, which entails an enormous potential for practical applications and future research. Formulations of P&S problems as CSPs, resource and temporal global constraints, and inference techniques are of particular interest of COPLAS.

This workshop has been hold from 2006 in the context of ICAPS and CP conferences, which provided a broader audience and gave the participants of both events the opportunity to exchange ideas and approaches that lead to a valuable and fruitful discussion, and inspired forthcoming research. COPLAS is ranked as CORE B in ERA Conference Ranking and it is covered in selected Elsevier database products.

In this edition, nine papers were accepted. They represent an advance in the integration of constraint satisfaction techniques in planning and scheduling frameworks and their application to real life problems. These papers are distributed between theoretical papers and application papers.

Miguel A. Salido and Roman Barták
June 2018

Contents

X and more Parallelism
Integrating LTL-Next into SAT-based Planning with Trajectory Constraints while Allowing for even more Parallelism

Gregor Behnke and **Susanne Biundo**
Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany
{gregor.behnke, susanne.biundo}@uni-ulm.de

Abstract

Linear temporal logic (LTL) provides expressive means to specify temporally extended goals as well as preferences. Recent research has focussed on compilation techniques, i.e., methods to alter the domain ensuring that every solution adheres to the temporally extended goals. This requires either new actions or an construction that is exponential in the size of the formula. A translation into boolean satisfiability (SAT) on the other hand requires neither. So far only one such encoding exists, which is based on the parallel $\exists$-step encoding for classical planning. We show a connection between it and recently developed compilation techniques for LTL, which may be exploited in the future. The major drawback of the encoding is that it is limited to LTL without the X operator. We show how to integrate X and describe two new encodings, which allow for more parallelism than the original encoding. An empirical evaluation shows that the new encodings outperform the current state-of-the-art encoding.

1 Introduction

Linear temporal logic (LTL (Pnueli 1977)) is a generic and expressive way to describe (state-)trajectory constraints. It is often used to specify temporal constraints and preferences in planning, e.g., to describe safety constraints, to state necessary intermediate goals, or to specify the ways in which a goal might be achieved. Most notably, the semantics of such constraints in PDDL 3.0 (Gerevini and Long 2005) is given in terms of LTL formulae, which is the de-facto standard for specifying planning problems.

Traditionally, LTL constraints are handled by first compiling them into an equivalent Büchi Automaton, and then translating the automaton into additional preconditions and effects for actions (see e.g. Edelkamp (2003)). This compilation might be too expensive as the Büchi Automaton for a formula ϕ can have up to $2^{|\phi|}$ states. Recent work proposed another compilation using Alternating Automata (Torres and Baier 2015). These automata have only $\mathcal{O}(|\phi|)$ states allowing for a guaranteed linear compilation. There are also planners that do not compile the model, but evaluate the formula during forward search, e.g., TALplanner (Doherty and Kvarnström 2001), TLplan (Bacchus and Kabanza 2000), or the work by Hsu et al. (2007). However, heuristics have to be specifically tailored to incorporate the formula, or else the search becomes blind. TALplanner and TLplan even use the temporally extended goals for additional search guidance.

Another option is to integrate LTL into planning via propositional logic. Planning problems can be translated into (a sequence of) boolean formulae. A temporally extended goal can then be enforced by adding additional clauses to this formula. So far only one such encoding has been developed by Mattmüller and Rintanen (2007). It uses an LTL to SAT translation from the model checking community, which assumes that only a single state transition is executed at a time. The main focus of their work lies on integrating the efficient $\exists$-step encoding with this representation of LTL formulae. In the $\exists$-step encoding operators can be executed simultaneously, as long as they are all applicable in the current state, the resulting state is uniquely determined, and there is an ordering in which they are actually executable. Mattmüller and Rintanen presented alterations to the $\exists$-step formula restricting the parallelism such that LTL formulas without the next-operator are handled correctly.

We point out an interesting relationship between the LTL encoding of Mattmüller and Rintanen and the Alternating Automaton encoding by Torres and Baier, showing that both use the same encoding technique, although derived by different means. This insight might prove useful in the future, e.g., to allow for optimisation of the propositional encoding using automata concepts. Next, we show how the propositional encoding by Mattmüller and Rintanen can be extended to also be able to handle the next-operator X. We introduce a new concept – partial evaluation traces – to capture the semantics of the encoding with respect to an LTL formula and show that our extension is correct. Based on partial evaluation traces, we show that the restrictions posed by Mattmüller and Rintanen (2007) on allowed parallelism can be relaxed while preserving correctness. We provide an alteration of their encoding allowing for more parallelism. We present an alternative encoding, also based on partial evaluation traces, which allows for even more parallelism by introducing intermediate timepoints at which the formula is evaluated. Our empirical evaluation of all encodings shows that our new encodings outperform the original one.

2 Preliminaries

Planning

We consider propositional planning without negative preconditions. This is known to be equivalent to STRIPS (al-

lowing for negative preconditions) via compilation. Also note that all our techniques are also applicable in the presence of conditional effects. We do not consider them in this paper to keep the explanation of the techniques as simple as possible. For the extension to conditional effects, see Mattmüller and Rintanen (2007).

Let A be a set of proposition symbols and $Lit(A) = \{a, \neg a \mid a \in A\}$ be the set of all literals over A. An action a is a tuple $a = \langle p, e\rangle$, where p – the preconditions – is a subset of A and e – the effects – is a subset of $Lit(A)$. We further assume that the effects are not self-contradictory, i.e., that for no $a \in A$ both a and $\neg a$ are in e. A state s is any subset of A. An action $a = \langle p, e\rangle$ is executable in s, iff $p \subseteq s$. The state resulting form executing a in s is $(s \setminus \{a \mid \neg a \in e\}) \cup \{a \mid a \in e\}$. A planning problem $\mathcal{P} = \langle A, O, s_I, g\rangle$ consists of a set of proposition symbols A, a set of operators O, the initial state s_I, and the goal $g \subseteq A$. A sequence of actions $o_1, \ldots, o_n$ is a plan for $\mathcal{P}$ iff there exists a sequence of states $s_0, \ldots, s_{n+1}$ such that for every $i \in \{1, \ldots, n+1\}$, o_i is applicable in s_i, its application results in s_{i+1}, $s_0 = s_I$, and $g \subseteq s_{n+1}$. This sequence of states is called an execution trace.

Linear Temporal Logic

Formulae in Linear Temporal Logic (LTL) are constructed over a set of primitive propositions. In the case of planning these are the proposition symbols A. LTL formulae are recursively defined as any of the following constructs, where p is a proposition symbol and f and g are LTL formulae.

$$\bot \mid \top \mid p \mid \neg f \mid f \wedge g \mid f \vee g \mid Xf \mid \mathring{X}f \mid Ef \mid Gf \mid fUg$$

X, $\mathring{X}$, E, G, and U are called temporal operators. There are several further LTL-operators (like $\mathring{U}$, R, or S and T, see e.g. (Biere et al. 2006)). Each of them can be translated into a formula containing only the temporal operators X, $\mathring{X}$, and U. The semantics of an LTL formula ϕ is given with respect to an execution trace. In general this trace can be infinitely long, as LTL can describe repeated behaviour. We consider only LTL over finite traces, which is commonly called LTL_f (De Giacomo and Vardi 2013). The encodings we present can easily be extended to the infinite case (see Mattmüller and Rintanen 2007). The truth value of an LTL_f formula ϕ is defined over an execution trace $\sigma = (s_0, s_1, \ldots, s_n)$ as $[[\phi]](\sigma)$ where

$$\begin{aligned}
[[p]](s_0, \sigma) &= p \in s_0 \qquad \text{if } p \in A \\
[[\neg f]](\sigma) &= \neg[[f]](\sigma) \\
[[f \wedge g]](\sigma) &= [[f]](\sigma) \wedge [[g]](\sigma) \\
[[f \vee g]](\sigma) &= [[f]](\sigma) \vee [[g]](\sigma) \\
[[Xf]](s_0, \sigma) &= [[\mathring{X}f]](s_0, \sigma) = [[f]](\sigma) \\
[[Xf]](s_0) &= \bot \\
[[\mathring{X}f]](s_0) &= \top \\
[[Ef]](s_0, \sigma) &= [[f]](s_0, \sigma) \vee [[Ef]](\sigma) \\
[[Gf]](s_0, \sigma) &= [[f]](s_0, \sigma) \wedge [[Gf]](\sigma) \\
[[Ef]](s_0) &= [[Gf]](s_0) = [[f]](s_0)
\end{aligned}$$

$$\begin{aligned}
[[fUg]](s_0, \sigma) &= [[g]](s_0, \sigma) \vee \\
&\quad ([[f]](s_0, \sigma) \wedge [[fUG]](\sigma)) \\
[[fUg]](s_0) &= [[g]](s_0)
\end{aligned}$$

The intuitive of the semantics of temporal operators are: Ef – eventually f, i.e., f will hold at some time, now or in the future, Gf – globally f, i.e., f will hold from now on for ever, fUg – f until g, i.e., g will eventually hold and until that time f will always hold, and Xf – next f, i.e., f holds in the next state of the trace. Since we consider the case of finite LTL, we have – in addition to standard LTL – a new operator: weak next $\mathring{X}$. The formula Xf requires that there is a next state and that f holds in that state. In contrast, $\mathring{X}f$ asserts that f holds if a next state exists; if there is none, $\mathring{X}f$ is always true, taking care of the possible end of the state sequence.

As a preprocessing step, we always transform an LTL formula ϕ into negation normal form without increasing its size, i.e., into a formula where all negations only occur directly before atomic propositions. This can be done using equivalences like $\neg Gf = E\neg f$. Next, we add for each proposition symbol $a \in A$ a new proposition symbol $\overline{a}$. Its truth value will be maintained such that it is always the inverse of a. I.e. whenever an action has $\neg a$ as its effect, we add the effect $\overline{a}$ and when it has the effect a we add $\neg\overline{a}$. Lastly, we replace $\neg a$ in ϕ with $\overline{a}$, resulting in a formula not containing negation.

Given a planning problem $\mathcal{P}$ and a LTL formula ϕ, LTL planning is the task of finding a plan π whose execution trace σ will satisfy ϕ, i.e., for which $[[\phi]](\sigma)$. For a given LTL formula ϕ we define $A(\phi)$ as the set of predicates contained in ϕ and $\mathcal{S}(\phi)$ to be the set of all its subformulae. We write $[o]_e^\phi$ for the intersections of the effects of o and $A(\phi)$, i.e. all those effects that occur in ϕ.

3 State-of-the-art LTL→SAT encoding

As far as we are aware, there is only a single encoding of LTL planning problems into boolean satisfiability, developed by Mattmüller and Rintanen (2007). They adapted a propositional encoding for LTL developed by Latvala et al. 2004 for bounded model checking. The main focus of Mattmüller and Rintanen's work lies on integrating modern, non-sequential encodings of planning problems into the formula. The encoding models evaluating the LTL formula in timesteps, which correspond to the states in a trace. In Latvala et al.'s encoding (which was not developed for planning, but for a more general automata setting) only a single action may be executed at each timestep in order to evaluate the formula correctly. Research in translating planning problems into propositional formulae has however shown that such sequential encodings perform significantly worse than those that allow for a controlled amount of parallel action execution (Rintanen, Heljanko, and Niemelä 2006). Mattmüller and Rintanen addressed the question of how to use the LTL encoding by Latvala et al. in a situation where multiple state transitions take care in parallel – as is the case in these planning encodings. They used the property of stutter-equivalence which holds for LTL_{-X} (i.e. LTL with-

out the X and $\mathring{X}$ operators) to integrate Latvala et al.'s encoding. To exploit stutter-equivalence, they had to restrict the allowed amount of parallelism to ensure correctness.

Their encoding, which we will denote with M&R'07, is based on the $\exists$-step encoding of propositional planning by Rintanen, Heljanko, and Niemelä (2006). As such, we start by reviewing the $\exists$-step encoding in detail. In this encoding the plan is divided into a sequence of timesteps $0, \ldots, n$. Each timestep t is assigned a resulting state using decision variables a^t for all $a \in A$ and $t \in \{1, \ldots, n+1\}$, each indicating that the proposition symbol a holds at timestep t, i.e. after executing the actions at timestep $t-1$. The initial state is represented by the variables a^0. Actions can be executed between two neighbouring timesteps t and $t+1$, which is represented by decision variables o^t for all $o \in O$ and $t \in \{0, \ldots, n\}$. If o^t is true the action o is executed at time t. The encoding by Kautz and Selman (1996) is then used to determine which actions are executable in a^t and how the state a^{t+1} resulting from their application looks like. In a sequential encoding, one asserts for each timestep t that at most one o^t atom is true. Intuitively, this is necessary to ensure that the state a^{t+1} resulting from executing the actions o^t is uniquely determined. Consider, e.g., a situation where two actions `move-a-to-b` and `move-a-to-c` are simultaneously applicable, but result in conflicting effects. Executing these two actions in parallel has no well-defined result. Interestingly, the mentioned constraint is not necessary in this case, as the encoding by Kautz and Selman already leads to an unsatisfiable formula. There are however situations, where the resulting state is well-defined, but it is not possible to execute the actions in any order. Consider two actions `buy-a` and `buy-b`, both requiring money, spending it, and achieving possession of `a` and `b`, respectively. Both actions are applicable in the same state and their parallel effects are well-defined, as they don't conflict. It is not possible to find a sequential ordering of these two actions that is executable, as both need money, which won't be present before executing the second action. This situation must be prohibited, which can easily be achieved by forbidding parallel action execution at all, as in the sequential encoding.

In the $\exists$-step encoding, executing actions in parallel is allowed. Ideally, we would like to allow any subset S of o^t to be executable in parallel, as long as there exists a linearisation of S that is executable in the state s^t represented by a^t and all executable linearisations lead to the same state s^{t+1}. This property is however practically too difficult to encode (Rintanen, Heljanko, and Niemelä 2006). Instead, the $\exists$-step encoding uses a relaxed requirement. Namely, (1) all actions in S must be executable in s^t, then it chooses a total order of all actions O, (2) asserts that if a set of actions S is executable, it must be executable in that order, and (3) that the state reached after executing them is this order is s^{t+1}. The encoding by Kautz and Selman ensures the first and last property. The $\exists$-step encoding has to ensure the second property. It however does not permit all subsets $S \subseteq O$ to be executable in parallel, but only those for which this property can be guaranteed.

As a first step, we have to find an appropriate order of actions in which as many subsets $S \subseteq O$ as possible can be executed. For this, the Disabling Graph (DG) is used. It determines which actions can be safely executed in which order without checking the truth of propositions inside the formula. In practice, one uses a relaxed version of the DG (i.e. one containing more edges), as it is easier to compute (Rintanen, Heljanko, and Niemelä 2006).

Definition 1. *Let $\mathcal{P} = \langle A, O, s_I, g\rangle$ be a planning problem. An action $o_1 = \langle p_1, e_1\rangle$* affects *an other action $o_2 = \langle p_2, e_2\rangle$ iff $\exists l \in A$ s.t. $\neg l \in e_1$ and $l \in p_2$.*

A Disabling Graph $DG(\mathcal{P})$ is a directed graph $\langle O, E\rangle$ with $E \subseteq O \times O$ that contains all edges (o_1, o_2) where o_1 affects o_2 and a state s exists that is reachable from s_I in which both o_1 and o_2 are applicable.

The DG is a domain property and is not tied to any specific timestep, as such the restrictions it poses apply to every timestep equally. The DG encodes which actions disable the execution of other actions after them in the same timestep, i.e., we ideally want the actions to be ordered in the opposite way in the total ordering chosen by the $\exists$-step encoding. If the DG is acyclic, we can execute all actions in the inverted order of the disabling graph, as none will disable an action occurring later in that order. If so, the propositional encoding does not need any further clauses, as any subset S of actions can be executed at a timestep – provided that their effects do not interfere.

The DG is in practice almost never acyclic. Problematic are only strongly connected components (SCCs) of the DG, were we cannot find an ordering s.t. we can guarantee executability for all subsets of actions. Instead we fix some order $\prec$ for each SCC, asserting that the actions in it will always be executed in that order, and ensure in the SAT formula that if two actions o_1 and o_2 with $o_1 \prec o_2$ are executed, o_1 does not affect o_2. This way, we can safely ignore *some* of the edges of the DG – as their induced constraints are satisfied by the fixed ordered $\prec$ – while others have to be ensured in the formula. I.e. if we ensure for every edge (o_1, o_2) with $o_1 \prec o_2$ that if o_1 is part of the executed subset o_2 is not, we know that there is a linearisation in which the chosen subset S is actually executable. To ensure this property, Rintanen, Heljanko, and Niemelä introduced *chains*. A chain $chain(\prec; E; R; l)$ enforces that whenever an action o in E is executed, all actions in R that occur after a in $\prec$ cannot be executed. Intuitively, E are those actions that produce some **e**ffect a, while the actions in R **r**ely on $\neg a$ to be true. The last argument l is a label that prohibits interference between multiple chains.

$$
\begin{aligned}
&chain(o_1, \ldots, o_n; E; R; l) = \\
&\quad \bigwedge \{o_i \rightarrow \mathfrak{a}^{j,l} \mid i < j, o_i \in E, o_j \in R, \{o_{i+1}, .., o_{j-1}\} \cap R = \emptyset\} \\
&\quad \cup \{l^i \rightarrow a^{j,l} \mid i < j, \{o^i, o^j\} \subseteq R, \{o_{i+1}, .., o_{j-1}\} \cap R = \emptyset\} \\
&\quad \cup \{l^i \rightarrow \neg o_i \mid o^i \in R\}
\end{aligned}
$$

To ensure that for any SCC S of $DG(\mathcal{P})$ the mentioned condition holds for the chosen ordering $\prec$ of S, we generate for every proposition symbol $a \in A$ a chain with

$$
\begin{aligned}
E_a &= \{o \in S \mid o = \langle p, e\rangle \text{ and } \neg a \in e\} \\
R_a &= \{o \in S \mid o = \langle p, e\rangle \text{ and } a \in p\}
\end{aligned}
$$

Based on the $\exists$-step encoding, Mattmüller and Rintanen (2007) added support for LTL formulae ϕ by exploiting the stutter-equivalence of LTL_{-X}. This stutter-equivalence ensures that if multiple actions are executed in a row but don't change the truth any of the predicates in $A(\phi)$, the truth of the formula is not affected, i.e., the truth of the formula does not depend on how many of these actions are executed in a row. Consequently the formula only needs to be checked whenever the truth of propositions in $A(\phi)$ changes. Their construction consists of two parts. First, they add clauses to the formula expressing that the LTL formula ϕ is actually satisfied by the trace restricted to the states where propositions in $A(\phi)$ change. These states are the ones represented in the $\exists$-step encoding by a^t atoms. Second, they add constraints to the $\exists$-step parallelism s.t. in every timestep the first action executed according to $\prec$ that changes proposition symbols in $A(\phi)$ is the only one to do so. Other actions in that timestep may not alter the state w.r.t to $A(\phi)$ achieved by that first action, but can assert the same effect.

In their paper, they provide a direct translation of ϕ into a proposition formula. In practice however, this formula cannot be given to a SAT solver, as it requires a formula in conjunctive normal form (CNF). The formula given in the paper is not in CNF and translating it into CNF can lead to a CNF of exponential size. They instead introduce additional variables (Mattmüller 2006), allowing them to generate (almost) a CNF[1]. For every sub-formula $\psi \in \mathcal{S}(\phi)$ and every timestep t they introduce the variable ψ^t_{LTL}, stating that ψ holds for the trace starting at timestep t. They then assert: (1) that ϕ^0_{LTL} holds and (2) that for every ψ^t_{LTL} the consequences must hold that make ψ true for the trace starting at time t. The latter is expressed by clauses $\psi^t_{LTL} \rightarrow [[\psi]]^t$, where $[[\psi]]^t$ is given in Tab. 1. Note that the M&R'07 encoding cannot handle the next operators X and $\mathring{X}$, as they are sensitive to stuttering, i.e., stutter-equivalence does not hold for formulae that contain X or $\mathring{X}$. We have added the encoding for X (Latvala et al. 2004). In addition, we have restricted the original encoding from infinite to finite LTL-traces and added a new encoding for $\mathring{X}$. Note that the M&R'07 encoding will lead to wrong results if used with the presented encoding of the X and $\mathring{X}$ operators. It is however correct, if used in conjunction with a sequential encoding (Latvala et al. 2004). We show in Sec. 5 how the M&R'07 encoding can be changed to handle X and $\mathring{X}$ correctly. Lastly, clauses need to be added in order to ensure that actions executed in parallel do not alter the truth of propositions in $A(\phi)$ – except for the first action that actually changes them. The extension of $\exists$-step encoding achieving this is conceptually simple, as it consists of only two changes to the original encoding:

1. add for every two actions o_1, o_2 which are simultaneously applicable the edge (o_1, o_2) to the DG iff $[o_2]^\phi_e \backslash [o_1]^\phi_e \neq \emptyset$, i.e., o_2 would change more than o_1 with respect to $A(\phi)$.
2. add for every literal $l \in Lit(A(\phi))$ and SCC of $DG(\mathcal{P})$ with its total order $\prec$ the chain $chain(\prec; E^\phi_l; R^\phi_l; \phi_l)$ with
   (a) $E^\phi_l = \{o \in O \mid o = \langle p, e\rangle \text{ and } l \notin e\}$
   (b) $R^\phi_l = \{o \in O \mid o = \langle p, e\rangle \text{ and } l \in e\}$;

Mattmüller and Rintanen (2007) have proven that these clauses suffice to ensure a correct and complete encoding.

[1]The translations of $f \wedge g$, Gf and fUg contain one conjunction each, which can be multiplied out easily.

ϕ	$t < n$	$t = n$
$[[p]]^t \quad p \in A$	p^t	p^t
$[[f \wedge g]]^t$	$f^t_{LTL} \wedge g^t_{LTL}$	$f^t_{LTL} \wedge g^t_{LTL}$
$[[f \vee g]]^t$	$f^t_{LTL} \vee g^t_{LTL}$	$f^t_{LTL} \vee g^t_{LTL}$
$[[Xf]]^t$	f^{t+1}_{LTL}	$\bot$
$[[\mathring{X}f]]^t$	f^{t+1}_{LTL}	$\top$
$[[Ef]]^t$	$f^t_{LTL} \vee (Ef)^{t+1}_{LTL}$	f^t_{LTL}
$[[Gf]]^t$	$f^t_{LTL} \wedge (Gf)^{t+1}_{LTL}$	f^t_{LTL}
$[[fUg]]^t$	$g^t_{LTL} \vee ((fUg)^{t+1}_{LTL} \wedge f^t_{LTL})$	f^t_{LTL}

Table 1: Transition rules for LTL formulae

4 Alternating Automata and M&R'07

In recent years, research on LTL planning focussed on translation based approaches. There, the original planning problem is altered in such a way that all solutions for the new problem adhere to the formula (e.g. (Baier and McIlraith 2006; Torres and Baier 2015), see (Camacho et al. 2017) for an overview). Traditionally, these approaches translate the LTL formula into a Büchi automaton (essentially a finite state machine, with a specific accepting criterion for infinite traces) and then integrate the automaton into the model. The major drawback of these translations is that the Büchi automaton for an LTL formula can have up to $2^{|\phi|}$ many states.

Torres and Baier (2015) proposed a translation diverging from the classical LTL to Büchi translation. They instead based it on Alternating Automata, which are commonly used as an intermediate step when constructing the Büchi automaton for an LTL ϕ formula (see e.g. (Gastin and Oddoux 2001)). Alternating Automata have a guaranteed linear size in $|\phi|$, but have a more complex transition function.

Definition 2. *Given a set of primitive propositions A, an alternating automaton is a 4-tuple $\mathcal{A} = (Q, \delta, I, F)$ where*

- *Q is a finite set of states*
- *$\delta : Q \times 2^A \rightarrow \mathcal{B}^+(Q)$ is a transition function, where $\mathcal{B}^+(Q)$ is the set of positive propositional formulas over the set of states Q, i.e., those formulae containing only $\vee$ and $\wedge$.*
- *$I \subseteq Q$ is the initial state*
- *$F \subseteq Q$ is set of final states.*

A run of an alternating automaton over a sequence of sets of propositions (execution trace) $(s_1, \ldots, s_n)$ is a sequence of sets of states $(Q_0, \ldots, Q_n)$ such that

- *$Q_0 = I$*
- *$\forall i \in \{1, \ldots, n\} : Q_i \models \bigwedge_{q \in Q_{i-1}} \delta(q, s_i)$*

The alternating automaton accepts the trace iff $Q_n \subseteq F$

Torres and Baier (2015) generate an alternating automaton for an LTL formula ϕ as follows. They choose Q as the set of sub-expression of ϕ starting with a temporal operator plus a state q_F representing that the end of the execution trace has been reached. Being in a state q means, that from the current time on, we have to fulfill the formula q. The automaton is given as $A_\phi = (Q, \delta, \{q_\phi\}, \{q_F\})$ where $Q = \{q_\alpha \mid \alpha \in \mathcal{S}(\phi)\} \cup \{q_F\}$ and the transition function δ is defined as follows:

$$\delta(q_l, s) = \begin{cases} \top & \text{, if } l \in s \\ \bot & \text{, if } l \notin s \end{cases}$$
$$\delta(q_F, s) = \bot$$
$$\delta(q_{f \vee g}, s) = \delta(q_f, s) \vee \delta(q_g, s)$$
$$\delta(q_{f \wedge g}, s) = \delta(q_f, s) \wedge \delta(q_g, s)$$
$$\delta(q_{X f}, s) = q_f$$
$$\delta(q_{\mathring{X} f}, s) = q_F \vee q_f$$
$$\delta(q_{E f}, s) = \delta(f, s) \vee q_{E f}$$
$$\delta(q_{G f}, s) = \delta(f, s) \wedge (q_{G f} \vee q_F)$$
$$\delta(q_{f U g}, s) = \delta(q_g, s) \vee (\delta(q_f, s) \wedge q_{f U g})$$

Note that we have to enumerate all states that are relevant to the formula, i.e., all states $s \subseteq 2^{A(\phi)}$, to construct the formula. Using the Alternating Automaton as the basis for a translation leads to a guaranteed linear increase in size when constructing the translated planning problem. This is due to the fact that the encoding does not actually has to construct the automaton, but only has to simulate its states. We will elaborate on this later. Also, it was demonstrated that the new encoding is more efficient that other current translation techniques (Torres and Baier 2015).

A drawback of their translation was the need for introducing additional actions, performing bookkeeping on the current state of the alternating automaton. A translation into SAT, on the other hand, will not have this drawback, as we will show. We will again extend the $\exists$-step encoding and call the encoding AA (**A**lternating **A**utomaton). The restriction posed on parallelism by M&R'07 does not depend on the encoding of the formula itself, as long as it does not contain the X or $\mathring{X}$ operators[2]. We introduce new decision variables q^t for each state $q \in Q$ and timestep t, signifying that the automaton A_ϕ is in state q after executing the actions of timestep t. To express the transition function of the Alternating Automaton, we use formulae of the form $\left(q^t \wedge \bigwedge_{a \in s} a\right) \rightarrow \delta(q, s)$ for each state q of the automaton and set of propositions s. We also replace each occurrence of a state q_f in $\delta(q, s)$ with the decision variable q_f^{t+1} and introduce intermediate decision variables to break down complex formulae as in the M&R'07 encoding. The following theorem holds by construction.

Theorem 1. *AA in conjunction with M&R'07's $\exists$-step encoding is correct for LTL_{-X}.*

[2]The actual encoding of the formula can be exchanged in the proof of their main theorem as long as a similar version of their Theorem 2 can be proven, which is obvious in our case.

We here want to point out that the AA encoding is not (as the one by Torres and Baier (2015)) polynomial in the size of the formula. The reason lies in the explicit construction of the alternating automaton, which requires a single transition for every possible state that might be relevant to the formula, i.e., for every subset of $A(\phi)$. The translation encoding by Torres and Baier circumvents this construction by adding new operators, which can evaluate the necessary expression during planning. I.e. they have actions for each transition rule $\delta(\cdot, \cdot)$, which produce as their effects the right-hand sides of these above equations. Lastly, they introduce synchronisation actions to ensure that $\delta(\cdot, \cdot)$ is fully computed before another "real" actions is executed.

If we apply this idea to the AA encoding, we would end up with the M&R'07 encoding. Since the states of the automaton are the sub-formulae of ϕ starting with a temporal operator, these decision variables are identical to the ψ^t_{LTL} variables of the M&R'07 encoding, where ψ starts with a temporal operator. The encoding by Torres and Baier also needs to introduce state variables for every sub formulae not starting with a temporal operator to represent the step-wise computation of $\delta(\cdot, \cdot)$ correctly. If translated into propositional variables, these correspond to the ψ^t_{LTL} variables of M&R'07, where ψ does not start with a temporal operator. Lastly, the transition rules for both encodings are identical.

As such, the M&R'07 encoding can also be interpreted as a direct translation of an Alternating Automaton into propositional logic using the compression technique of Torres and Baier (2015). Interestingly, the original proof showing correctness of the LTL-part of the M&R'07 encoding by Latvala et al. (2004) does not rely on this relationship to Alternating Automata, neither do they mention this connection. We think it is an interesting theoretical insight, as it might enable to further improve LTL encoding, e.g., based on optimisations of the Alternating Automaton.

5 X, Parallelism, and Partial Evaluation

We have noted that both M&R'07 and AA cannot handle LTL formulae containing the X or $\mathring{X}$ operators in conjunction with the $\exists$-step encoding. They are however correct if used together with the sequential encoding, where only a single action to be executed at each timestep. In order to derive extensions that can handle X and $\mathring{X}$, we first present a new theoretical foundation for both encodings. We will use that fact that in M&R'07, we know which parts of the formula are made true at which time and by which propositions. To formalise this, we introduce evaluation traces, which specify how an LTL formula is fulfilled over a trace.

Definition 3. *Let ϕ be an LTL formula. We call a sequence $\psi = (f_0, \ldots, f_n)$ with $f_i \subseteq \mathcal{S}(\phi)$ an evaluation trace for ϕ iff $\phi \in f_0$ and for all $i \in \{0, \ldots, n\}$*

1. *if $f \vee g \in f_i$ then $f \in f_i$ or $g \in f_i$*
2. *if $f \wedge g \in f_i$ then $f \in f_i$ and $g \in f_i$*
3. *if $Xf \in f_i$ then $i < n$ and $f \in f_{i+1}$*
4. *if $\mathring{X} f \in f_i$ then $i = n$ or $f \in f_{i+1}$*
5. *if $Ef \in f_i$ then $f \in f_i$ or $i < n$ and $Ef \in f_{i+1}$*
6. *if $Gf \in f_i$ then $f \in f_i$ and if $i < n$ then $Gf \in f_{i+1}$*

7. *if* $fUg \in f_i$ *then* $g \in f_i$ *or* $i < n$ *and* $f \in f_i$ *and* $fUg \in f_{i+1}$

A trace $\pi = (s_0, \ldots, s_n)$ *satisfies an evaluation trace* $\psi = (f_0, \ldots, f_n)$ *iff for all* $a \in f_i \cap A$ *also* $a \in s_i$.

The following theorem follows directly, as the definition just emulates checking an LTL formula.

Theorem 2. *An execution trace* π *satisfies an LTL formula* ϕ *iff an evaluation trace* ψ *for* ϕ *exists that satisfies* π.

In M&R'07, the LTL formula is only evaluated after a set of parallel actions have been executed. To capture this, we define partial evaluation traces.

Definition 4. *Let* $\pi = (s_0, \ldots, s_n)$ *be an execution trace and* ϕ *and LTL formula. We call an evaluation trace* $\theta = (f_0, \ldots, f_l)$ *with* $l \leq n$ *a partial evaluation trace (PET) for* π *if a sequence of indices* $0 = i_0 < i_1 < ... < i_l = n + 1$ *exists such that for each* $k \in \{0, \ldots, l-1\}$ *holds*

$$s_{i_k} \cap (f_k \cap A) = \cdots = s_{i_{k+1}-1} \cap (f_k \cap A)$$

and if $Xf \in f_k$ *or* $\mathring{X}f \in f_k$ *and* $k > 0$ *then* $i_{k-1} + 1 = i_k$. *The PET* θ *is satisfied by the execution trace* π, *iff the execution trace* $(s_{i_1-1} \cap f_0 \cap A, \ldots, s_{i_l-1} \cap f_l \cap A)$ *satisfies* θ *in the sense of Def. 3.*

A satisfying valuation of the M&R'07 encoding corresponds to a partial evaluation trace that satisfies the formula ϕ. M&R'07 also asserts that PET is satisfied by the execution trace corresponding to the sequential plan generated by the $\exists$-step formula. The main property of Def. 4, which is necessary for showing that we actually generate a PET, is ensured by the chains added to the original $\exists$-step encoding and the additional edges in the Disabling Graph. The following theorem states that every partial evaluation trace that is satisfied by an executed trace can be extended to a full evaluation trace and thus forms a witness that the execution trace satisfies the LTL formula. This gives us a second, independent proof of correctness for the M&R'07 encoding.

Theorem 3. *Let* π *be a trace and* θ *be an PET for the formula* ϕ *on* π. *If* π *satisfies* θ, *then* π *satisfies* ϕ.

Proof. We need to show that the PET $\psi = (f_0, \ldots, f_l)$ can be extended to a full evaluation trace on $\pi = (s_0, \ldots, s_n)$, s.t. π satisfies that evaluation trace. If so, we can apply Thm. 2 and conclude that π also satisfies ϕ. Let $i_0, \ldots, i_l$ be the indices of Def. 4 for which θ is a PET. We claim that

$$\theta^* = (\overbrace{f_0, \ldots f_0}^{i_1 - i_0 \text{ times}}, \overbrace{f_1, \ldots, f_1}^{i_2 - i_1 \text{ times}}, \ldots, \overbrace{f_l, \ldots, f_l}^{i_l - i_{l-1} \text{ times}})$$

is an evaluation trace that satisfies π, and that π satisfies ϕ. First we show that θ^* is an evaluation trace. We start by proving the enumerated properties of Def. 3. Consider the ith element f^* of θ^* and let f^{**} be the $i+1$th element (if such exists).

1. trivially satisfied
2. trivially satisfied
3. $Xf \in f^*$. We know that f^* is the only repetition some f_j in the trace, as θ is a PET. Also $f^{**} = f_{j+1}$. Consequently $f \in f^{**}$. In case f^{**} does not exist, θ cannot be an PET.
4. $\mathring{X}f \in f^*$. We know that f^* is the only repetition of some f_j in the trace, as θ is a PET. In case f^{**} does not exist, we have nothing to show. Else, $f^{**} = f_{j+1}$ and $f \in f^{**}$.

For the last three requirements relating to the temporal operators E, G, and U, we can distinguish three cases depending on where f^* is situated in the sequence θ^*

- f^* is the last element of θ^*
  5. if $Ef \in f^*$ then $f \in f^*$, as θ is a PET.
  6. if $Gf \in f^*$ then $f \in f^*$, as θ is a PET.
  7. if $fUg \in f^*$ then $g \in f^*$, as θ is a PET.
- $f^* \neq f^{**}$, i.e., the last repetition of f^*. We know that $f^* = f_j$ and $f^{**} = f_{j+1}$ for some $j \in \{0, \ldots, l-1\}$.
  5. if $Ef \in f^*$ then either $f \in f_j = f^*$ or $Ef \in f_{j+1} = f^{**}$
  6. if $Gf \in f^*$ then $f \in f_j = f^*$ and $Gf \in f_{j+1} = f^{**}$
  7. if $fUG \in f^*$ then either $g \in f_{j+1} = f^{**}$ or $f \in f_j = f^*$ and $fUg \in f_{j+1} = f^{**}$
- if $f^* = f^{**}$
  5. if $Ef \in f^*$ then $Ef \in f^{**}$
  6. if $Gf \in f^*$ then $Gf \in f^{**}$ and $f \in f^*$
  7. if $fUg \in f^*$ then either $g \in f^*$, or $f \in f^*$, but then also $fUG \in f^{**}$, since $f^* = f^{**}$

$\phi \in f_0$ holds as θ is a PET, which concludes conclude the proof that θ^* is an evaluation trace.

Lastly, we have to show that θ^* satisfies π, i.e., we have to show for each timestep $j \in \{0, \ldots n\}$ and every $a \in A$ with is true in the ith element of θ^* that $a \in s_j$ holds. Consider first the indices of the last repetitions of each f_k, i.e., the states s_{i_k-1}. Since θ is a PET, it satisfies $(s_{i_1-1} \cap f_0 \cap A, \ldots, s_{i_l-1} \cap f_l \cap A)$, so it satisfies the required property for all time-steps $i_k - 1$. Consider any other timestep t and its next timestep in the PET $i_k - 1$ (which always exists, since the last index is equal to n). Since θ is a PET, we know that $s_t \cap (f_k - 1 \cap A) = s_{i_k-1} \cap (f_k \cap A)$. We have chosen to set the tth element of θ^* to f_k. Since $s_{i_k-1} \cap (f_k \cap A)$ satisfies the required property for f_k, so must $s_t \cap (f_k \cap A)$ and thus s_t itself (it can have only more true predicates). □

We can now use this result to integrate support for X and $\mathring{X}$ into the M&R'07 encoding. For that, we have to assert that the second last condition of Def. 4 holds, as all other requirements are already checked by the M&R'07 encoding. We first add four new variables per time step t.

- $exactOne^t$ – exactly one action is executed at time t
- $atLeastOne^t$ – at least one action is executed at time t
- $atMostOne^t$ – at most one action is executed at time t
- $none^t$ – no action is executed at any time $\geq t$

To enforce the semantics of these variables, we add the following clauses per timestep:

$$\forall o \in O : none^t \rightarrow \neg o^t$$
$$none^t \rightarrow none^{t+1}$$
$$atLeastOne^t \rightarrow \bigvee_{o \in O} o^t$$
$$exactOne^t \rightarrow atLeastOne^t \wedge atMostOne^t$$

Encoding the $atMostOne^t$ atom is a bit more complicated. A native encoding requires $\mathcal{O}(|O|^2)$ clauses. There are however better encodings for the at most one constraint in SAT. We have chosen the log-counter encoding, which introduces $\log(|S|)$ new variables while only requiring $|S|\log(|S|)$ clauses (Frisch et al. 2005). To ensure the semantics of the atom $atMostOne^t$, we add it as a guard to the log-counter encoding, i.e., we add $\neg atMostOne^t$ to every clause. If $atMostOne^t$ is required to be true, the log-counter clauses have to satisfies, i.e., at most one action atom can be true. If $atMostOne^t$ can be false, we can simply set it to $\bot$ thereby satisfying all log-counter clauses. We lastly set $exactOne^t$ for $t = -1$ to $\top$ and $atLeastOne^{n+1}$ to false. Based on these new variables, we can add the constraints necessary for evaluating X and $\mathring{X}$ correctly under the $\exists$-step encoding. For each timestep t, we add

$$(Xf)^t_{LTL} \rightarrow exactOne^{t-1} \wedge f^{t+1} \wedge atLeastOne^t$$
$$(\mathring{X}f)^t_{LTL} \rightarrow exactOne^{t-1}$$
$$(\mathring{X}f)^t_{LTL} \rightarrow (f^{t+1} \wedge atLeastOne^t) \vee none^t$$

A valuation of this encoding represents a partial evaluation trace satisfied by the execution trace of its actions. By applying Thm. 3, we know that a satisfying evaluation trace for the plan exists. Showing completeness for the encoding is trivial, since a sequential assignment satisfies the formula for every plan. We will also call this extended encoding M&R'07 in our evaluation, as it is exactly identical to this one, provided no X or $\mathring{X}$ operator is present.

So far, we have not used the – in our view – most significant improvement: Relaxing the restrictions on parallelism to only those actually needed by the current formula. Doing so based on our theorem is surprisingly easy. Consider a timestep of the M&R'07 encoding, in which actions can be executed in parallel, which are given an implicit order $\prec$ by the $\exists$-step encoding. For each literal $l \in Lit(A(\phi))$ a chain ensures that the action changing it is applied first, i.e., that the "first" action of a timestep performs all changes relevant to the whole formula and can thus check the formula only in the resulting state. By applying Def. 4 and Thm. 3, we have to ensure this property only for those proposition symbols that need to be evaluated after the actions have been executed, i.e., for all a^{t+1}_{LTL} that are true. We can do this simply by adding the literal $\neg a^t_{LTL}$ as a guard (simiar to $atMostOne^t$) to every clause in the chains $chain(\prec; E^\phi_l; R^\phi_l; \phi_a)$ and $chain(\prec; E^\phi_l; R^\phi_l; \phi_{\neg a})$. If we have to make the literal a^t_{LTL} true, the chains become active, if not they are inactive (as they are trivially satisfied by $\neg a^t_{LTL}$). We will denote this improved encoding with Improved-M&R'07.

To illustrate the effects of the improved M&R'07 encoding, consider the following planning problem. There are six proposition symbols a, b, c, d, e and f, of which a and b are true in the initial state and e has to hold in the goal state. There are five actions described in Tab. 2. We consider this domain in conjunction with the formula $\phi = G((a \wedge d) \rightarrow Ef) = G(\neg a \vee \neg d \vee Ef)$. The disabling graph, extended by the edges needed for the M&R'07 encoding, is depicted in Fig. 1. This planning problem has only a single solution, namely: Y, X, V, W, Z. Under the M&R'07 encoding, we need four timesteps to find a plan, i.e., {Y}, {X}, {V}, {W,Z}. This is due to the fact that most action's effects contain one of the three predicates contained in ϕ, but not the others. Using the improved M&R'07 encoding, we only need three timesteps, as now Y and X can be executed together in the first timestep. The reason for this beeing possible is quite unintuitive, but it shows the strength of our approach. In the M&R'07 encoding, the encoding correctly detects that there is a state in which a and d are true simultaneously[3] after the action Y has been executed and that thus Ef has to hold after executing Y. In the plan for the improved encoding, the solver can simply choose to achieve Ef after {Y,X} has been executed. This way the solver is never forced to achieve $\neg a$ or $\neg d$ and can thus completely ignore the associated constraints, i.e., chains.

	X	Y	Z	V	W
pre	a,b	a,b	c,d	c,d	g
add	c	d	e	g	f
del	a		c		

Table 2: Actions in the example domain

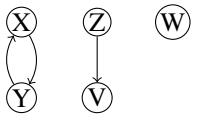


Figure 1: Extended disabling graph for the example domain.

6 Parallelism with Tracking

There is however still room for more parallelism even compared to the improved M&R'07 encoding. The key observation is that in many domains only a few actions actually influence the truth of variables in an LTL formula, and that those that do are usually close to each other in a topological ordering of the inverse disabling graph. The $\exists$-step semantic guarantees that if actions are executed in parallel, they can be sequentially executed in this order. Let this ordering be $(o_1, \ldots, o_n)$. We can divide it into blocks, such that for each block $(o_i, \ldots, o_j)$ it holds that

$$[o_i]^\phi_e \supseteq [o_{i+1}]^\phi_e \supseteq \cdots \supseteq [o_j]^\phi_e$$

Along the actions in a block the effects that contain predicates in $A(\phi)$ can only "decrease". A block forms a set of actions that can always – without further checking at runtime – be executed in parallel in the M&R'07 encoding. The number of such blocks is surprisingly small for most domains (see Tab. 3). We denote with $B = ((o_1, \ldots, o_i), \ldots, (o_j, \ldots, o_n))$ the sequence of blocks for a given ordering of actions.

If an action from a blocks has been executed, at least (usually more) the first action of the next block cannot be executed any more in the same timestep, as it would change the truth of some $a \in A(\phi)$, even it would be possible

[3] Technically both $\overline{a}$ and $\overline{d}$ are not true.

in the pure $\exists$-step encoding. We present a method to circumvent this restriction on parallelism. Instead of restricting the amount of parallel actions executing inside a timestep, we (partially) trace the truth of an LTL formula within that timestep to allow maximal parallelism. This is based on the insight that all proofs by Mattmüller and Rintanen (2007) do not actually require an action to be present at any timestep, i.e., the set of actions executed in parallel can also be empty. So, conceptually, we split each timestep into $|B|$ many timesteps and restrict the actions in the ith splitted step to be those of the ith block. Then we use the M&R'07 encoding, without the need to add chain-clauses apart from those of the $\exists$-step encoding, as they are automatically fulfilled. The resulting encoding would be sound and complete for LTL formulae without X and $\mathring{X}$ by virtue of the results proven by Mattmüller and Rintanen (2007).

We can however improve the formula even further. In the proposed encoding, we would compute the state after each block using the Kautz&Selman encoding. This is unnecessary, as know from the $\exists$-step encoding that we only need to compute it after all blocks of one original timestep have been executed. We only need to trace the truth of propositions in $A(\phi)$ between blocks. For that we don't need to check preconditions – they are already ensured by the $\exists$-step encoding. We end up with the $\exists$-step encoding, where we add at every timestep a set of intermediate timepoints at which the truth of propositions in $A(\phi)$ and the truth of the LTL formula ϕ is checked. Thus we call this encoding OnParallel.

As we have noted above, this construction works only for the original M&R'07 encoding, as supporting the X and $\mathring{X}$ requires to be able to specify that in the next timestep some action must be executed. This might not be possible with splitted timesteps, as the next action to be executed may only be contained in a timestep $|B|$ steps away. Luckily, we can fix this problem by slightly altering the encoding we used to track the truth of X and $\mathring{X}$ operators.

$$(Xf)^t \rightarrow atMostOne^{t-1} \wedge ((atLeastOne^{it} \wedge f_{LTL}^{t+1}) \vee (nextNone^{t+1} \wedge (Xf)_{LTL}^{t+1}))$$

$$(\mathring{X}f)^t \rightarrow atMostOne^{t-1} \wedge ((f_{LTL}^{t+1} \wedge atLeastOne^{t}) \vee (nextNone^{t+1} \wedge (\mathring{X}f)_{LTL}^{t+1}) \vee none^t))$$

The semantics of $noneAt^t$ is ensured by clauses $nextNone^t \rightarrow \neg o^t$ for all $i \in O$. Lastly, we add $\neg X f_{LTL}^{n++1}$ for any $Xf \in \mathcal{S}$ stating that a next-formula cannot be made true at the last timestep. This would else be possible, since $atMostOne^n$ could simply be made true. The OnParallel encoding is correct by applying Thm. 3.

7 Evaluation

We have conducted an evaluation in order to asses the performance of our proposed encodings. We used the same experimental setting as Mattmüller and Rintanen (2007) in their original paper. We used the domains `trucks` and `rover` from the preference track of IPC5 (the original paper considered only `rover`), which contain temporally extended goals to specify preference. In these domains, temporally-extended goals are formulated using the syntax of PDDL 3.0 (Gerevini and Long 2005). We parse the preferences and transform them into LTL formulae using the patterns defined by Gerevini and Long (Gerevini and Long 2005). As did Mattmüller and Rintanen, we interpret these preferences as hard constraints and randomly choose a subset of three constraints per instance[4]. To examine the performance of our encoding for X and $\mathring{X}$, we have also tested the instances of the `trucks` domain with a formula that contains these operators. We have used the following formula[5]:

$$\phi = \forall ?l{-}Location ?t - Truck : G(at(?l, ?t) \rightarrow \mathring{X}(\neg at(?l, ?t) \vee \mathring{X} \neg at(?l, ?t)))$$

It forces each truck to stay at a location for at most one timestep – either it leaves the location right after entering it, or in the next timestep. The domain contains an explicit symbolic representation of time, which is used in temporal goals. When planning with ϕ, the number of timesteps is never sufficed to find a plan satisfying ϕ. We have therefore removed all preconditions, effects, and action parameters pertaining to the explicit representation of time. As a result, the domain itself is easier than the original one. We denote these instances in the evaluation with trucks-XY-ϕ.

Each planner was given 10 minutes of runtime and 4GB of RAM per instance on an Intel Xeon E5-2660 v3. We've used the SAT solver Riss6 (Manthey, Stephan, and Werner 2016), one of the best-performing solvers in the SAT Competition 2016. We have omitted results for all the trucks instances $11 - 20$, as no planner was able to solve them.

Table 3 shows the results of our evaluation. We show per instance the number of ground actions and blocks. The number of blocks is almost always significantly smaller than the number of ground operators. In the largest `rover` instance, only $\approx 1.4\%$ of operators start a new block. For the `trucks` domain this is $\approx 1.7\%$ for the largest instance.

For every encoding, we show both the number of parallel steps (i.e. timesteps) necessary to find a solution, as well as the time needed to solve the respective formula and the number of sequential plan steps found by the planner. In almost all instances the OnParallel encoding performs best, while there are some where the improved M&R'07 encoding is faster. Our improvement to the M&R'07 encoding almost always leads to a faster runtime. Also, the improved parallelism actually leads to shorter parallel plans. In approximately half of the instances we can find plans with a fewer parallel steps. In the experiments with the formula ϕ containing the $\mathring{X}$ operator, this is most pronounced. The OnParallel encoding cuts the number of timesteps by half and is hence significantly faster, e.g., on trucks-03-ϕ where the runtime is reduced from 165s to 6s. On the other hand, the sequential plans found are usually a few actions longer, although the same short plan could be found – this result is due to the non-determinism of the SAT solver.

[4]Mattmüller and Rintanen (2007) noted that it is impossible to satisfy all constraints at the same time and that a random sample of more than three often leads to unsolvable problems. If a sample of 3 proved unsolvable we have drawn a new one.

[5]We handlethese lifted LTL constraints by grounding them using the set of delete-relaxed reachable ground predicates.

Instance			M&R'07			Improved-M&R'07			OnParallel			AA		
	$\|O\|$	$\|B\|$	p. steps	s. steps	time	p. steps	s. steps	time	p. steps	s. steps	time	p. steps	s. steps	time
rover-01	63	10	7	16	**0.10**	7	16	0.11	7	16	**0.10**	7	**14**	0.23
rover-02	53	8	5	13	0.06	**4**	13	0.03*	**4**	14	**0.03**	4	**12**	0.04
rover-03	76	9	7	16	0.25	7	20	0.2*	**6**	17	**0.16**	7	**14**	0.35
rover-04	86	16	5	13	0.13	5	11	0.12*	**3**	**9**	**0.05**	5	14	0.12
rover-05	144	15	7	**26**	0.38	7	30	0.39	**6**	27	**0.29**	7	**26**	0.63
rover-06	178	16	10	42	1.08	**9**	42	0.73*	**8**	43	**0.6**	10	**39**	1.39
rover-07	151	7	5	31	0.37	5	29	0.33*	5	28	**0.31**	5	**21**	0.59
rover-08	328	9	7	40	1.06	7	43	1.11	7	45	**0.95**	7	**36**	1.24
rover-09	362	19	9	62	3.47	9	56	**3.13***	9	59	3.38	9	**36**	3.92
rover-10	382	14	6	54	1.73	**5**	46	1.17*	**4**	44	**0.77**	6	**40**	1.62
rover-11	436	9	10	**42**	3.91	10	**42**	3.77*	**9**	43	**2.93**	10	47	6.46
rover-12	366	15	5	27	1.28	5	29	1.18*	5	33	**1.07**	5	**25**	1.43
rover-13	749	11	8	**61**	4.51	8	65	4.75	8	66	**3.99**	8	62	5.27
rover-14	525	19	7	**40**	3.66	7	43	3.66	7	42	**3.31**	7	50	4.13
rover-15	751	12	8	64	6.18	**7**	60	4.85*	**7**	60	**4.36**	7	**47**	5.34
rover-16	671	14	6	44	3.66	6	47	3.41*	6	50	**3.32**	6	**43**	3.99
rover-17	1227	13	11	106	34.97	11	105	34.49*	11	103	**31.45**	11	**62**	37.17
rover-18	1837	49	5	65	10.63	5	65	11.04	5	**64**	**10.11**	5	70	10.59
rover-19	2838	17	8	91	20.28	8	94	19.63*	8	100	**19.00**	8	**86**	21.38
rover-20	3976	58	8	130	65.99	8	127	65.97*	8	119	**65.2**	8	**98**	66.38
trucks-01	333	82	8	15	0.69	**7**	**14**	**0.44***	**7**	15	0.50	8	15	0.98
trucks-02	624	56	9	20	1.73	**8**	**17**	1.26*	**8**	18	**1.21**	9	18	1.87
trucks-03	1065	68	10	24	2.43	10	**22**	2.47	**9**	**22**	**1.68**	10	23	6.34
trucks-04	1692	48	12	27	8.48	11	27	**6.68***	11	26	6.93	12	**25**	12.80
trucks-05	2541	85	12	29	13.70	11	30	11.07*	11	**27**	**10.56**	12	28	13.37
trucks-06	4928	112	14	37	42.01	14	36	**39.39***	14	**35**	41.37	14	**35**	88.09
trucks-07	7380	267	14	40	171.81	**13**	38	119.95*	**13**	39	**119.75**	14	**36**	247.5
trucks-08	9760	260	13	42	**151.22**	13	41	153.05	13	**39**	153.81	–	–	TLE
trucks-09	12628	203	14	45	432.99	14	**42**	373.30*	14	44	**362.54**	14	43	465.15
trucks-10	16032	267	–	–	TLE	–	–	TLE	–	–	TLE	–	–	TLE
trucks-01-ϕ	123	6	18	18	2.12	18	18	2.27	**9**	18	**0.24**	18	18	18.45
trucks-02-ϕ	162	6	24	24	14.44	24	24	14.55	**12**	24	**1.09**	24	24	498.93
trucks-03-ϕ	201	6	29	29	164.56	29	29	165.64	**15**	29	**5.97**	–	–	TLE
trucks-04-ϕ	240	6	–	–	TLE	–	–	TLE	**18**	36	**84.20**	–	–	TLE
trucks-05-ϕ	279	6	–	–	TLE	–	–	TLE	–	–	TLE	–	–	TLE

Table 3: SAT-solver runtime and solution length of several encodings. Per run, we give both the number of parallel steps (p. steps) and the number of sequential (s. steps). Minimum run-times and sequential plan lengths are marked in bold. If the number of parallel plan steps decreased for any of the two new encodings, they are also marked bold. Decreases in Runtime for the improve M&R'07 encoding compared to the original one are marked with an asterisk. TLE indicates exceeding the timelimit of 10 minutes.

8 Conclusion

In this paper, we have improved the state-of-the-art in translating LTL planning problems into propositional formulae in several ways. We have first pointed out an interesting theoretical connection between the propositional encoding by Mattmüller and Rintanen (2007) and the compilation technique by Torres and Baier (2015). Next, we have presented a new theoretical foundation for the M&R'07 encoding – partial evaluation traces. Using them, we presented (1) a method to allow the X and $\mathring{X}$ operators in the M&R'07 encoding, (2) a method to further improve the parallelism in the M&R'07 encoding, and (3) a new encoding for LTL planning. In an evaluation, we have shown that both our improved M&R'07 encoding and the OnParallel encoding perform empirically better than the original encoding by Mattmüller and Rintanen. We plan to use the developed encoding in a planning-based assistant (Behnke et al. 2018) for enabling the user to influence the instructions he is presented by the assistant, which is turn are based on the solution generated by a planner. Instructions given by the user can be interpreted as LTL goal and integrated into the plan using the presented techniques.

Acknowledgement

This work is funded by the German Research Foundation (DFG) within the technology transfer project "Do it yourself, but not alone: Companion Technology for Home Improvement" of the Transregional Collaborative Research Centre SFB/TRR 62. The industrial project partner is the Corporate Research Sector of the Robert Bosch GmbH.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Baier, J., and McIlraith, S. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on AI (AAAI 2006)*, 788–795. AAAI Press.

Behnke, G.; Schiller, M.; Kraus, M.; Bercher, P.; Schmautz, M.; Dorna, M.; Minker, W.; Glimm, B.; and Biundo, S. 2018. Instructing novice users on how to use tools in DIY projects. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence (IJCAI-ECAI 2018)*. AAAI Press.

Biere, A.; Heljanko, K.; Junttila, T.; Latvala, T.; and Schuppan, V. 2006. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science* 2(5):1–64.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A. 2017. Non-deterministic planning with temporally extended goals: Ltl over finite and infinite traces. In *Proceedings of the 31st National Conference on AI (AAAI 2017)*, 3716–3724. AAAI Press.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 854–860. AAAI Press.

Doherty, P., and Kvarnström, J. 2001. TALPLANNER – A temporal logic-based planner. *The AI Magazine* 22(3):95–102.

Edelkamp, S. 2003. On the compilation of plan constraints and preferences. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*. AAAI Press.

Frisch, A.; Peugniez, T.; Doggett, A.; and Nightingale, P. 2005. Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings. *Journal of Automated Reasoning (JAR)* 35(1-3):143–179.

Gastin, P., and Oddoux, D. 2001. Fast ltl to büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, 53–65. Springer-Verlag.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2007. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1924–1929. AAAI Press.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, 1194–1201.

Latvala, T.; Biere, A.; Heljanko, K.; and Junttila, T. 2004. Simple bounded LTL model checking. In *Proceedings of the 5th Conference on Formal Methods in Computer-Aided Design (FMCAD 2004)*, 189–200. FMCAD Inc.

Manthey, N.; Stephan, A.; and Werner, E. 2016. Riss 6 solver and derivatives. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. University of Helsinki.

Mattmüller, R., and Rintanen, J. 2007. Planning for temporally extended goals as propositional satisfiability. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1966–1971. AAAI Press.

Mattmüller, R. 2006. Erfüllbarkeitsbasierte Handlungsplanung mit temporal erweiterten Zielen.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, 46–57. IEEE.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1696–1703. AAAI Press.

Forward-Search Temporal Planning with Simultaneous Events

Daniel Furelos-Blanco and **Anders Jonsson**
Department of Information and Communication Technologies
Universitat Pompeu Fabra
Roc Boronat 138, 08018 Barcelona, Spain
{daniel.furelos, anders.jonsson}@upf.edu

Héctor Palacios
Nuance Communications
Montreal, Canada
hector.palaciosverdes@nuance.com

Sergio Jiménez
Department of Computer Systems and Computation
Universitat Politècnica de València
Camino de Vera s/n. 46022 Valencia, Spain
serjice@dsic.upv.es

Abstract

In this paper we describe STP, a novel algorithm for temporal planning. Similar to several existing temporal planners, STP relies on a transformation from temporal planning to classical planning, and constructs a temporal plan by finding a sequence of classical actions that solve the problem while satisfying a given set of temporal constraints. Our main contribution is that STP can solve temporal planning problems that require simultaneous events, i.e. the temporal actions have to be scheduled in such a way that two or more of their effects take place concurrently. To do so, STP separates each event into three phases: one phase in which temporal actions are scheduled to end, one phase in which simultaneous effects take place, and one phase in which temporal actions are scheduled to start. Experimental results show that STP significantly outperforms state-of-the-art temporal planners in a domain requiring simultaneous events.

Introduction

How expressive can a forward-search temporal planner be? The third *International Planning Competition* (IPC) introduced the most common language for modeling temporal planning problems, PDDL 2.1 (Fox and Long 2003). PDDL 2.1 is compatible with classical planning problems, and its semantics are defined in terms of the semantics of classical actions. This connection was studied further by Rintanen (2007), who proved that plan existence for temporal planning with succinct models is EXPSPACE-complete. As a motivation, Rintanen mentioned a basic reduction from temporal planning to classical planning called TEMPO (Cushing et al. 2007), and proposed a restriction to PDDL 2.1 that is reducible to classical planning, implying a decrease in complexity from EXPSPACE to PSPACE.

Later, Jiménez, Jonsson, and Palacios (2015) built on this idea and provided an effective implementation of TEMPO relying on a simple modification of a classical planner. In contrast to Cushing et al. (2007) and later work, Jiménez, Jonsson, and Palacios dealt with more expressive forms of concurrency, i.e. concurrency does not only occur in the form of single hard envelopes.

Rintanen (2007) argued that the complexity of temporal planning remains in PSPACE if we avoid an unbounded dependency on past and present information for determining the next temporal state. His restrictions prohibit a temporal action from executing in parallel with itself, and assumes that time is discrete.

In this work we explore further the expressivity of classical planning for solving complex temporal problems, focusing on the case of *simultaneous events* in which the effects of temporal actions take place concurrently. Many situations in the real-world involve simultaneous events. A clear example are relay races where a runner gives the relay at the same time that another runner receives it. This scenario, for instance, could be translated into an assembly line where robotic arms give and receive mechanical pieces.

Given the importance of classical planning as a basic model for multi-agent planning (Brafman and Domshlak 2008), we foresee the importance of truly concurrent temporal planning for enabling interesting forms of multi-agent temporal planning. Furthermore, Rintanen (2015b) showed that PDDL 2.1 induces temporal gaps between consecutive independent actions; thus, no current approach taking PDDL problems as input are capable of producing plans with simultaneous events.

Our approach builds on previous work by Jiménez, Jonsson, and Palacios (2015), but we stress the differences with respect to the initial ideas of Rintanen (2007), such that our translation is not necessarily affected by the time scale or the durations of the actions.

The rest of the paper is organized as follows. First we introduce classical and temporal planning models. Next we present STP, motivating the compilation and proving its soundness. Then we present experimental results, showing that STP is particularly strong in the case of simultaneous events. Finally, we comment on related work and conclude.

Background

In this section we introduce the formalisms of classical planning and temporal planning. Since we are interested in temporal planning with simultaneous events, we focus specifically on the semantics of concurrent action execution. Fur-

thermore, we introduce the mechanism for preserving the temporal constraints between actions.

Classical Planning

Let F be a set of propositional variables or *fluents*. A *state* $s \subseteq F$ is a subset of fluents that are true, while all fluents in $F \setminus s$ are implicitly assumed to be false. A subset of fluents $F' \subseteq F$ *holds* in a state s if and only if $F' \subseteq s$.

A classical planning instance is a tuple $P = \langle F, A, I, G \rangle$, where F is a set of fluents, A is a set of actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a goal condition (usually satisfied by multiple states). Each action $a \in A$ has precondition $\mathsf{pre}(a) \subseteq F$, add effect $\mathsf{add}(a) \subseteq F$, and delete effect $\mathsf{del}(a) \subseteq F$, each a subset of fluents. Action a is *applicable* in state $s \subseteq F$ if and only if $\mathsf{pre}(a)$ holds in s, and applying a in s results in a new state $s \ltimes a = (s \setminus \mathsf{del}(a)) \cup \mathsf{add}(a)$.

A *plan* for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces a state sequence $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_0 = I$ and, for each i such that $1 \leq i \leq n$, a_i is applicable in s_{i-1} and results in the next state $s_i = s_{i-1} \ltimes a_i$. The plan π *solves* P if and only if G holds in the last state, i.e. if $G \subseteq s_n$.

Actions may have *conditional effects*, a common extension to classical actions. Each conditional effect has a condition and effects $\langle \mathsf{cond}_i(a), \mathsf{cadd}_i(a), \mathsf{cdel}_i(a) \rangle$. When an action a with conditional effects is applicable in an state s, the effects of a include effects whose conditions hold in s, assuming the usual consistency requirements. Let's say

$$\mathsf{tadd} = \mathsf{add}(a) \cup \textstyle\bigcup_{s \models \mathsf{cond}_i(a)} \mathsf{cadd}_i(a),$$
$$\mathsf{tdel} = \mathsf{del}(a) \cup \textstyle\bigcup_{s \models \mathsf{cond}_i(a)} \mathsf{cdel}_i(a).$$

Then, $s \ltimes a = (s \setminus \mathsf{tdel}(a)) \cup \mathsf{tadd}(a)$.

A *concurrent action* $\mathcal{A} = \{a^1, \ldots, a^k\}$ is a set of multiple actions from A. We adopt the definition of valid concurrent actions from PDDL 2.1 (Fox and Long 2003):

Definition 1. *A concurrent action $\mathcal{A} = \{a^1, \ldots, a^k\}$ is* valid *if an only if does not exist a fluent $f \in F$ and an action pair $(a^i, a^j) \subseteq \mathcal{A}$ such that $f \in \mathsf{add}(a^i) \cup \mathsf{del}(a^i)$ and $f \in \mathsf{pre}(a^j) \cup \mathsf{add}(a^j) \cup \mathsf{del}(a^j)$.*

Intuitively, if f is an effect of an action $a^i \in \mathcal{A}$, f cannot appear as a precondition or effect of another action $a^j \in \mathcal{A}$. Though this definition imposes a strong restriction on concurrent actions, it is commonly used in temporal planning, and implemented as part of VAL (Howey, Long, and Fox 2004), a tool used to validate temporal plans.

We can view a valid concurrent action $\mathcal{A} = \{a^1, \ldots, a^k\}$ as a classical action by defining its precondition and effects as the union of the individual preconditions and effects:

$$\mathsf{pre}(\mathcal{A}) = \bigcup_{i=1}^{k} \mathsf{pre}(a^i), \quad \mathsf{add}(\mathcal{A}) = \bigcup_{i=1}^{k} \mathsf{add}(a^i),$$

with $\mathsf{del}(\mathcal{A})$ defined analogously. Due to Definition 1, $\mathcal{A}$ is a well-defined classical action without conflicting effects.

A *concurrent plan* for P is a sequence of concurrent actions $\pi = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$. The concurrent plan π solves P if and only if each concurrent action $\mathcal{A}_i$, $1 \leq i \leq n$, is valid and the resulting sequence of equivalent classical actions solves P according to the semantics of classical plans.

Temporal Planning

A *temporal planning problem*[1] is a tuple $P = \langle F, A, I, G \rangle$, where the fluent set F, initial state I and goal condition G are defined as for classical planning. The action set A consists of *temporal* or *durative* actions $a \in A$ composed of:

- $d(a)$: duration.
- $\mathsf{pre}_s(a), \mathsf{pre}_o(a), \mathsf{pre}_e(a)$: preconditions of a at start, over all, and at end, respectively.
- $\mathsf{add}_s(a), \mathsf{add}_e(a)$: add effects of a at start and at end.
- $\mathsf{del}_s(a), \mathsf{del}_e(a)$: delete effects of a at start and at end.

Although a has a duration, its effects apply instantaneously at the start and end of a, respectively. The preconditions $\mathsf{pre}_s(a)$ and $\mathsf{pre}_e(a)$ are also checked instantaneously, but $\mathsf{pre}_o(a)$ has to hold for the entire duration of a.

The semantics of temporal actions can be defined in terms of two discrete *events* start_a and end_a, each of which is naturally expressed as a classical action (Fox and Long 2003):

$$\begin{aligned} \mathsf{pre}(\mathsf{start}_a) &= \mathsf{pre}_s(a), & \mathsf{pre}(\mathsf{end}_a) &= \mathsf{pre}_e(a), \\ \mathsf{add}(\mathsf{start}_a) &= \mathsf{add}_s(a), & \mathsf{add}(\mathsf{end}_a) &= \mathsf{add}_e(a), \\ \mathsf{del}(\mathsf{start}_a) &= \mathsf{del}_s(a), & \mathsf{del}(\mathsf{end}_a) &= \mathsf{del}_e(a). \end{aligned}$$

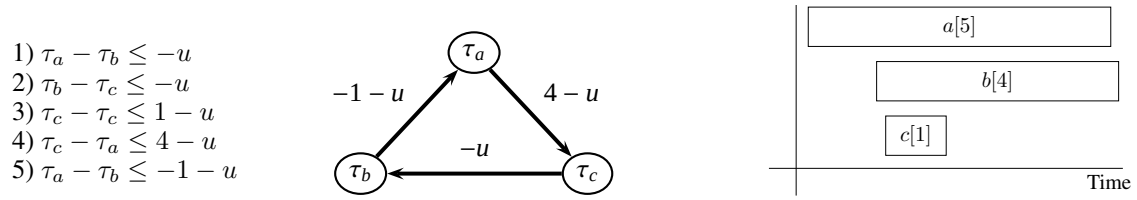
Starting temporal action a in state s is equivalent to applying the classical action start_a in s, first verifying that $\mathsf{pre}(\mathsf{start}_a)$ holds in s. Ending a in state s' is equivalent to applying end_a in s', first verifying that $\mathsf{pre}(\mathsf{end}_a)$ holds in s'. The duration $d(a)$ and precondition over all $\mathsf{pre}_o(a)$ impose restrictions on this process: end_a has to occur exactly $d(a)$ time units after start_a and $\mathsf{pre}_o(a)$ has to hold in all states between start_a and end_a. For brevity, we use the term *context* to refer to a precondition over all $\mathsf{pre}_o(a)$, and we use $F_o \subseteq F$ to denote the set of fluents that appear in contexts.

A *temporal plan* for P is a set of action-time pairs $\pi = \{(a_1, t_1), \ldots, (a_k, t_k)\}$. Each action-time pair $(a, t) \in \pi$ is composed of a temporal action $a \in A$ and a scheduled start time t of a, and induces two events start_a and end_a with associated timestamps t and $t + d(a)$, respectively. If we order events by their timestamp and merge events with the same timestamp, the result is a concurrent plan $\pi' = \langle \mathcal{A}_1, \ldots, \mathcal{A}_m \rangle$ for the associated classical planning problem $P' = \langle F, A', I, G \rangle$, where $A' = \{\mathsf{start}_a, \mathsf{end}_a : a \in A\}$.

A temporal plan $\pi = \{(a_1, t_1), \ldots, (a_k, t_k)\}$ solves P if and only if the induced concurrent plan $\pi' = \langle \mathcal{A}_1, \ldots, \mathcal{A}_m \rangle$ solves the associated classical planning problem P' and, for each $(a, t) \in \pi$ with $\mathsf{start}_a \in \mathcal{A}_i$ and $\mathsf{end}_a \in \mathcal{A}_j$, the context $\mathsf{pre}_o(a)$ holds in the states $s_i, \ldots, s_{j-1}$ of the state sequence induced by π', i.e. in all states between actions $\mathcal{A}_i$ and $\mathcal{A}_j$.

For π to solve P, the concurrent actions of the induced concurrent plan $\pi' = \langle \mathcal{A}_1, \ldots, \mathcal{A}_m \rangle$ have to be valid according to Definition 1. The plan π contains *simultaneous events* if and only if $m < 2k$, i.e. if at least two induced events share time stamps. The context $\mathsf{pre}_o(a)$ of a temporal action a is not affected by simultaneous events: if start_a is part of a concurrent action $\mathcal{A}$, it is safe for another event in $\mathcal{A}$ to add a fluent $f \in \mathsf{pre}_o(a)$, and if end_a is part of a concurrent action $\mathcal{A}'$, it is safe for an event in $\mathcal{A}'$ to delete f.

[1] We use the definition of PDDL 2.1 (Fox and Long 2003).



(a) Temporal constraints. (b) STN. (c) Resulting temporal plan.

Figure 1: Example temporal constraints, associated STN and resulting temporal plan.

The quality of a temporal plan is given by its *makespan*, i.e. the temporal duration from the the start of the first temporal action to the end of the last temporal action. Without loss of generality, we assume that the first temporal action is scheduled to start at time 0, i.e. $\min_{(a,t)\in\pi} t = 0$. In this case, the makespan of a temporal plan π is formally defined as $\max_{(a,t)\in\pi}(t + d(a))$.

Simple Temporal Networks (STNs)

Temporal constraints on time variables can be represented using *simple temporal networks*, or STNs (Dechter, Meiri, and Pearl 1991). An STN is a directed graph with time variables τ_i as nodes, and an edge (τ_i, τ_j) with label c represents a constraint $\tau_j - \tau_i \leq c$. Dechter, Meiri, and Pearl (1991) showed that scheduling fails if and only if an STN contains negative cycles. Else, the range of feasible assignments to a time variable τ_i is given by $[-d_{i0}, d_{0i}]$, where d_{ij} is the shortest distance in the graph from τ_i to τ_j and τ_0 is a reference time variable whose value is assumed to be 0. Floyd-Warshall's shortest path algorithm can be used to compute shortest paths and test for negative cycles (i.e. whether the cost of a shortest path from a node to itself is negative).

We illustrate the application of STNs to temporal planning using an example from (Cushing et al. 2007). Let a, b and c be temporal actions with durations $d(a) = 5$, $d(b) = 4$ and $d(c) = 1$, respectively. Assume that we are given an event sequence $\langle \mathsf{start}_a, \mathsf{start}_b, \mathsf{start}_c, \mathsf{end}_c, \mathsf{end}_a, \mathsf{end}_b \rangle$. Then there are three associated time variables τ_a, τ_b and τ_c; we designate τ_a as the reference time variable whose value is 0 since a is the temporal action that starts first. Given the above event sequence, the temporal constraints induced by consecutive events in the sequence are:

1. $\tau_a < \tau_b$,
2. $\tau_b < \tau_c$,
3. $\tau_c < \tau_c + d(c)$,
4. $\tau_c + d(c) < \tau_a + d(a)$,
5. $\tau_a + d(a) < \tau_b + d(b)$.

Since the temporal constraints of TEMPO are strict, we introduce a slack unit of time u and rewrite each constraint $\tau_j + x < \tau_i + y$ on the form $\tau_j - \tau_i \leq y - x - u$ (this is possible since τ_i and x are non-negative). Figure 1a shows the temporal constraints rewritten this way. Note that constraint 5) subsumes constraint 1), and that constraint 3) is trivially satisfied whenever $u < 1$.

Figure 1b shows the associated STN after removing constraints 1) and 3). This STN has no negative cycles, and the range of feasible assignments to τ_b is given by $[-d_{ba}, d_{ab}] = [1 + u, 4 - 2u]$. Likewise, the range of feasible assignments to τ_c is given by $[-d_{ca}, d_{ac}] = [1 + 2u, 4 - u]$. This makes sense: a starts at time 0, so for b to end after a ends, b has to start after time 1. For c to end before a ends, c has to start before time 4. The remaining bounds are implied by the fact that c starts after b starts. Since one of the goals of temporal planning is to minimize makespan, i.e. the time until the last action of the temporal plan ends, we always select the smallest possible assignment to each time variable τ_i, i.e. $-d_{i0}$. In the example, this results in a temporal plan $\{(a, 0), (b, 1 + u), (c, 1 + 2u)\}$, illustrated in Figure 1c.

Theoretically, STNs can be modeled in PDDL, using fluents to represent the entries of a matrix and actions to simulate updates of Floyd-Warshall. However, each entry of the matrix can take on a range of values, and the size of the matrix is *not* bounded by the number of active actions, but rather the total number of temporal actions in the plan. In practice, the enormous number of necessary fluents and actions makes this approach unfeasible.

Jiménez, Jonsson, and Palacios (2015) proposed an alternative approach for incorporating STNs into temporal planning. They represent lifted temporal states as part of the search nodes of the Fast Downward planning system (Helmert 2006), which is where auxiliary information about a state is stored (e.g. the predecessor state). Specifically, to each search node they add an STN, a list of active actions and the latest event.

Each time a compiled action is applied (either start_a or end_a for some a), Fast Downward generates a successor state. To the search node associated with this state they add an STN which is a copy of the STN of its predecessor, but with a single new edge corresponding to the temporal constraint generated by the successor rule of TEMPO. They then recompute the shortest paths of the STN.

Given an STN (V, E) with accompanying shortest paths, Cesta and Oddi (1996) described an $O(|V||E|)$ algorithm for adding a single edge to the STN and recomputing the shortest paths. However, Jiménez, Jonsson, and Palacios take a different approach to updating the STN. Instead of explicitly representing the STN, they represent the STN implicitly using the matrix of shortest distances (as in Floyd Warshall's algorithm). When a new edge (τ_i, τ_j) is added to the STN, for each pair of nodes there is a single new

candidate shortest path, namely that via (τ_i, τ_j). Since the shortest distances to τ_i and from τ_j are already represented, the update can be performed in time $O(|V|^2)$, which is typically much smaller than $O(|V||E|)$. In the modified version of Fast Downward, they prune a search node whenever the corresponding STN contains negative cycles, i.e. when the temporal actions cannot be scheduled in a way that coincides with the current event sequence. The temporal constraints can thus be viewed as an implicit precondition of actions which is invisible to the planner (e.g. when computing heuristics).

POPF (Coles et al. 2010) and OPTIC (Benton, Coles, and Coles 2012) also use STNs for solving temporal planning problems. POPF encodes the STN using linear programming, which allows it to compute plans with actions that cause continuous linear numeric changes. On the other hand, OPTIC encodes the STN as a mixed integer problem which additionally allows handling temporally dependent costs.

The STP Planner

In this section we describe STP (Simultaneous Temporal Planner). STP is built on top of the same framework as TP (Jiménez, Jonsson, and Palacios 2015), but incorporates additional machinery in order to handle simultaneous events. STP shares several characteristics with TP:

1. Both TP and STP apply a modified version of the Fast Downward (FD) planning system to generate temporal plans. The modified version of FD incorporates simple temporal networks or STNs to represent temporal constraints.

   There is a time variable τ_i for each temporal action a_i of a temporal plan, and a feasible assignment to τ_i corresponds to the time t_i when a_i should be scheduled to form an action-time pair (a_i, t_i). During the search process, a branch is pruned if the temporal constraints are violated. At the end of the section we describe the temporal constraints imposed by STP.

2. Both TP and STP impose a bound K on the number of *active* temporal actions, that started but did not end yet. Hence no more than K temporal actions can execute concurrently.

3. Both compilations are described for problems with fixed durations and no duration dependent effects.

Unlike TP, STP also defines a constant C that represents the maximum value of a cyclic counter that starts at 0, and resets to 0 after reaching C (more details later).

STP works by protecting the contexts of temporal actions in case a naïve execution of events using classical planning would produce inconsistent results. Our compilation divides each concurrent event into three phases:

1. End phase (immediately before the event). This is where active actions are scheduled to end, and in doing so, the corresponding counters of fluents in context are decremented.

2. Event phase (concurrent event itself). This is where simultaneous events take place, both ending and starting actions. Here we check preconditions and apply effects, and verify that the concurrent event is valid.

3. Start phase (immediately after the event). Here we check that the contexts of active actions that just started are satisfied (possibly as a result of being added during the concurrent event itself), and increment the corresponding counters of fluents in context.

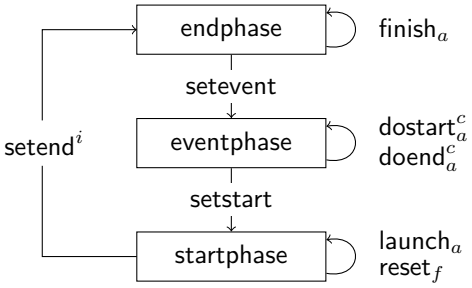


Figure 2: Interaction between the different actions introduced by the STP planner in the different phases.

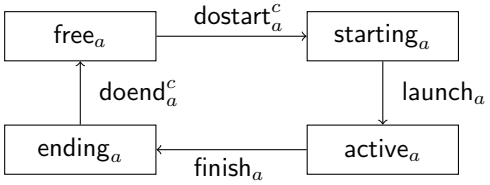


Figure 3: Fluents that are enabled each time an action in the compilation is executed.

Figure 2 shows the interconnection between the phases and actions introduced in the compilation. Actions setevent, setstart and setend^i change the current phase, whereas the other actions can only be applied (if their preconditions hold) in the corresponding phase. Actions dostart and doend correspond to the semantic events. Execution begins in the endphase, and ends in the startphase.

Figure 3 shows the cycle each action $a \in A$ passes through in the compilation. Between $\mathsf{dostart}_a$ and doend_a, actions launch_a and finish_a execute the start phase and end phase, respectively. Each time we transition from one state to another, we delete the auxiliary fluent of the state, and add the next, thus obtaining a mutex invariant. We also use additional fluents $\mathsf{nstarting}_a$ and $\mathsf{nending}_a$, that have the opposite values of $\mathsf{starting}_a$ and ending_a, respectively.

Now, we are ready to present the compilation itself. Let $P = \langle F, A, I, G \rangle$ be a temporal planning problem. Given constants K and C, STP compiles P into a classical planning problem $\Pi_{K,C} = \langle F_{K,C}, A_{K,C}, I_{K,C}, G_{K,C} \rangle$.

Fluents

To ensure that the contexts of temporal actions are not violated, STP follows the same scheme as TP: for each fluent $f \in F_o$ that appears in contexts, we introduce fluents count_f^c that model the number c of active temporal actions that have

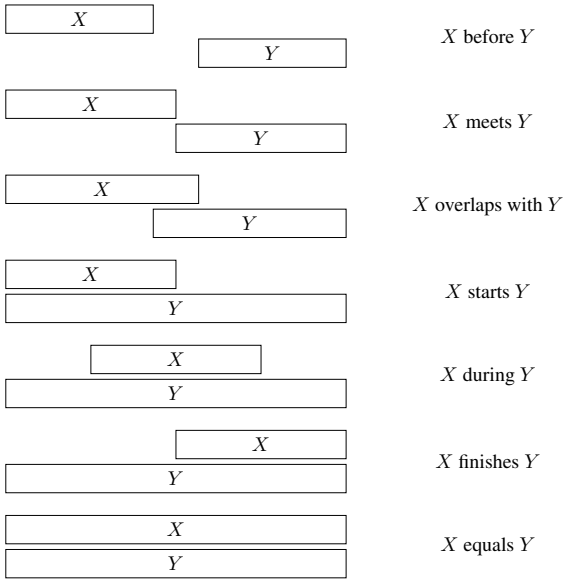


Figure 4: The seven relations on interval pairs (X, Y) in Allen's interval algebra.

f as context. For an event to delete a fluent $f \in F_o$, its count has to equal 0 (no active actions can have f as context).

STP also has to ensure that joint events are valid, and ensure that contexts are properly handled at the start and end of a temporal action. Recall that a fluent $f \in \mathsf{pre}_o(a)$ in the context of a temporal action a may be added by an event that is simultaneous with the start of a, and that f may be deleted by an event that is simultaneous with the end of a.

The set $F_{K,C} \supseteq F$ contains the following new fluents:

- For each $a \in A$, fluents free_a and active_a indicating that a is free (i.e. did not start) or active.
- For each $f \in F_o$ and each $c, 0 \leq c \leq K$, a fluent count_f^c indicating that c active actions have f as context.
- For each $c, 0 \leq c \leq K$, a fluent concur^c indicating that there are c concurrent active actions.
- Fluents $\mathsf{endphase}$, $\mathsf{eventphase}$ and $\mathsf{startphase}$ corresponding to the three phases described above.
- For each $a \in A$, fluents $\mathsf{starting}_a$, ending_a, $\mathsf{nstarting}_a$ and $\mathsf{nending}_a$ indicating that a is starting, ending, not starting and not ending, respectively.
- For each $f \in F$, fluents canpre_f and caneff_f indicating that we can use f as a precondition or effect.
- Fluents $\mathsf{endcount}^i, 0 \leq i < C$, that model the number of times the end phase has occurred. This counter is cyclic, i.e. if $i = C - 1$, then $i + 1 = 0$.

To motivate the role of the end count, we consider an example instance of the Allen's Interval Algebra (AIA) domain (Jiménez, Jonsson, and Palacios 2015), where a set of time intervals must be scheduled such that they comply with a set of relations between them (see Figure 4). The example is the following: the start of i_1 and i_2 should be simultaneous, as should the end of i_2 and i_3. The durations of the intervals are 5 for i_1 and i_3, and 11 for i_2. Figure 5a shows the only possible solution for this problem, whereas Figure 5b shows two different intermediate solutions A and B that the planner might explore. The black dotted lines indicate end phases. In solution A, the start of i_3 is concurrent with the end of i_1, which is not the case in solution B. The solid red line indicates the time at which solutions A and B assign the same values to the fluents in $F_{K,C}$.

If no end counts are maintained and a planner first explores A, it will not find a solution since ending i_2 and i_3 simultaneously violates the temporal constraints. When the planner later explores B, it will find the state on fluents identical to A and prune this branch of search. As a result, the planner will report that no solution exists, even though the instance does have a valid solution, namely B. The end count allows the planner to distinguish between A and B.

Note that the end count is not infallible: since it is cyclic, state repetitions can still happen. The higher the end count is, the less likely it is that states are repeated. However, increasing the end count increases the complexity of the problem (higher number of fluents and actions), so it can be more difficult to obtain a solution.

Lemma 1. *The number of fluents of $\Pi_{K,C}$ is given by $|F_{K,C}| = 3\,|F| + 6\,|A| + (K+1)\,(|F_o| + 1) + C + 3$.*

Proof. By inspection of the fluents in $F_{K,C}$. For each $f \in F$, $F_{K,C}$ contains three fluents f, canpre_f and caneff_f. For each $a \in A$, $F_{K,C}$ contains six fluents free_a, active_a, $\mathsf{starting}_a$, ending_a, $\mathsf{nstarting}_a$ and $\mathsf{nending}_a$. For each $f \in F_o$, $F_{K,C}$ contains $K+1$ fluents of type count_f^c, and there are $K+1$ fluents of type concur^c. Finally, there are C fluents of type $\mathsf{endcount}^i$, and three fluents $\mathsf{endphase}$, $\mathsf{eventphase}$ and $\mathsf{startphase}$. □

The initial state $I_{K,C}$ is defined as

$$I_{K,C} = I \cup \{\mathsf{free}_a, \mathsf{nstarting}_a, \mathsf{nending}_a : a \in A\}$$
$$\cup \{\mathsf{concur}^0, \mathsf{endphase}, \mathsf{endcount}^0\} \cup \{\mathsf{count}_f^0 : f \in F_o\}$$
$$\cup \{\mathsf{canpre}_f, \mathsf{caneff}_f : f \in F\},$$

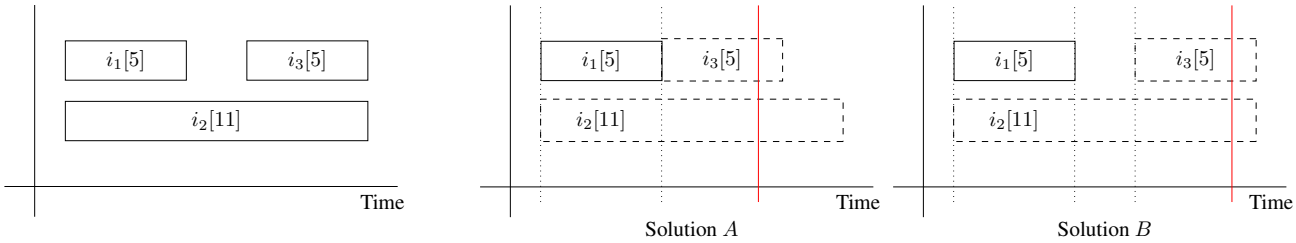
and the goal is $G_{K,C} = G \cup \{\mathsf{concur}^0\} \cup \{\mathsf{startphase}\}$.

Actions

The action set $A_{K,C}$ contains several actions corresponding to each temporal action $a \in A$: $\mathsf{dostart}_a^c$ and launch_a for starting a, and doend_a^c and finish_a for ending a. For each c, $0 \leq c < K$, action $\mathsf{dostart}_a^c$ is defined as

$$\mathsf{pre} = \mathsf{pre}_s(a) \cup \{\mathsf{eventphase}, \mathsf{concur}^c, \mathsf{free}_a\}$$
$$\cup \{\mathsf{count}_f^0 : f \in F_o \cap \mathsf{del}_s(a)\} \cup \{\mathsf{canpre}_f : f \in \mathsf{pre}_s(a)\}$$
$$\cup \{\mathsf{caneff}_f : f \in \mathsf{add}_s(a) \cup \mathsf{del}_s(a)\},$$
$$\mathsf{add} = \mathsf{add}_s(a) \cup \{\mathsf{concur}^{c+1}, \mathsf{starting}_a\},$$
$$\mathsf{del} = \mathsf{del}_s(a) \cup \{\mathsf{concur}^c, \mathsf{free}_a, \mathsf{nstarting}_a\}$$
$$\cup \{\mathsf{caneff}_f : f \in \mathsf{pre}_s(a)\}$$
$$\cup \{\mathsf{canpre}_f, \mathsf{caneff}_f : f \in \mathsf{add}_s(a) \cup \mathsf{del}_s(a)\}.$$

For a given $c < K$, we can only start a in the event phase if there are c active actions and a is free. All contexts deleted



(a) The only possible scheduling.

(b) Two possible intermediate solutions. With no end counter, B is not explored (see text).

Figure 5: Example instance of the AIA domain: i_1 and i_2 start at the same time, while i_2 and i_3 end at the same time.

at start of a need a count of 0, and all preconditions and effects at start of a have to be available. Starting a adds fluent $\mathsf{starting}_a$, deletes free_a and $\mathsf{nstarting}_a$, and increments the number of active actions. Moreover, deleting fluents of type canpre_f and caneff_f prevents invalid joint events: if f is a precondition at start of a, f can no longer be used as an effect in this event phase, and if f is an effect at start of a, f can no longer be used as a precondition or effect.

For each $c, 0 \leq c < K$, action doend_a^c is defined as

$$\begin{aligned}
\mathsf{pre} &= \mathsf{pre}_e(a) \cup \{\mathsf{eventphase}, \mathsf{concur}^{c+1}, \mathsf{ending}_a\} \\
&\cup \{\mathsf{count}_f^0 : f \in F_o \cap \mathsf{del}_e(a)\} \cup \{\mathsf{canpre}_f : f \in \mathsf{pre}_e(a)\} \\
&\cup \{\mathsf{caneff}_f : f \in \mathsf{add}_e(a) \cup \mathsf{del}_e(a)\}, \\
\mathsf{add} &= \mathsf{add}_e(a) \cup \{\mathsf{concur}^c, \mathsf{free}_a, \mathsf{nending}_a\}, \\
\mathsf{del} &= \mathsf{del}_e(a) \cup \{\mathsf{concur}^{c+1}, \mathsf{ending}_a\} \\
&\cup \{\mathsf{caneff}_f : f \in \mathsf{pre}_e(a)\} \\
&\cup \{\mathsf{canpre}_f, \mathsf{caneff}_f : f \in \mathsf{add}_e(a) \cup \mathsf{del}_e(a)\}\}.
\end{aligned}$$

For a given value of c, we can only end a in the event phase if there are $c+1$ active actions and a is already scheduled to end, represented by fluent ending_a. Ending a adds fluents free_a and $\mathsf{nending}_a$ and decrements the number of active actions. The remaining action definition is analogous to $\mathsf{dostart}_a^c$ and controls the validity of the joint event.

Action launch_a is responsible for completing the start of a during the start phase:

$$\begin{aligned}
\mathsf{pre} &= \mathsf{pre}_o(a) \cup \{\mathsf{startphase}, \mathsf{starting}_a\}, \\
\mathsf{add} &= \{\mathsf{active}_a, \mathsf{nstarting}_a\}, \\
\mathsf{del} &= \{\mathsf{starting}_a\}.
\end{aligned}$$

This is where we check that the contexts of a hold, and due to the precondition $\mathsf{starting}_a$ we can only launch a if a was started during the event phase. The result is adding active_a and $\mathsf{nstarting}_a$ and deleting $\mathsf{starting}_a$. In addition to the described effects, the action launch_a includes conditional effects $\langle\{\mathsf{count}_f^l\}, \{\mathsf{count}_f^{l+1}\}, \{\mathsf{count}_f^l\}\rangle$, $f \in \mathsf{pre}_o(a)$ and $0 \leq l < K$, incrementing the count of each $f \in \mathsf{pre}_o(a)$.

Finally, action finish_a is needed to schedule a for ending:

$$\begin{aligned}
\mathsf{pre} &= \{\mathsf{endphase}, \mathsf{active}_a\}, \\
\mathsf{add} &= \{\mathsf{ending}_a\}, \\
\mathsf{del} &= \{\mathsf{active}_a, \mathsf{nending}_a\}.
\end{aligned}$$

The result is adding ending_a and deleting active_a and $\mathsf{nending}_a$. In addition, finish_a includes conditional effects for decrementing the context counts of fluents in $\mathsf{pre}_o(a)$.

The action set $A_{K,C}$ also needs actions $\mathsf{setevent}$, $\mathsf{setstart}$ and setend^i, $0 \leq i < C$, whose purpose is to switch between phases. Action $\mathsf{setevent}$ is defined as

$$\begin{aligned}
\mathsf{pre} &= \{\mathsf{endphase}\}, \\
\mathsf{add} &= \{\mathsf{eventphase}\}, \\
\mathsf{del} &= \{\mathsf{endphase}\}.
\end{aligned}$$

Action $\mathsf{setstart}$ is defined as

$$\begin{aligned}
\mathsf{pre} &= \{\mathsf{eventphase}\} \cup \{\mathsf{nending}_a : a \in A\}, \\
\mathsf{add} &= \{\mathsf{startphase}\}, \\
\mathsf{del} &= \{\mathsf{eventphase}\}.
\end{aligned}$$

Note that we cannot leave the event phase unless all actions in the joint event have ended.

Action setend^i is defined as

$$\begin{aligned}
\mathsf{pre} &= \{\mathsf{startphase}, \mathsf{endcount}^i\} \cup \{\mathsf{nstarting}_a : a \in A\} \\
&\quad \cup \{\mathsf{canpre}_f, \mathsf{caneff}_f : f \in F\}, \\
\mathsf{add} &= \{\mathsf{endphase}\} \cup \{\mathsf{endcount}^j : j = (i+1) \bmod C\}, \\
\mathsf{del} &= \{\mathsf{startphase}, \mathsf{endcount}^i\}.
\end{aligned}$$

Note that we cannot leave the start phase unless all actions in the joint event have started and all fluents are available as preconditions or effects. In addition, setend^i increments the end count.

The resulting action set, $A_{K,C}$, also contains a reset action reset_f for each $f \in F$:

$$\begin{aligned}
\mathsf{pre} &= \{\mathsf{startphase}\}, \\
\mathsf{add} &= \{\mathsf{canpre}_f, \mathsf{caneff}_f\}, \\
\mathsf{del} &= \emptyset.
\end{aligned}$$

These actions can only be applied in the start phase.

Lemma 2. *The number of actions of the classical planning problem $\Pi_{K,C}$ is $|A_{K,C}| = (2K+2)\,|A| + C + |F| + 2$.*

Proof. For each $a \in A$, $A_{K,C}$ contains $2K+2$ actions $\mathsf{dostart}_a^c$, doend_a^c, $\mathsf{launch_a}$ and finish_a, $0 \leq c < K$. $A_{K,C}$ also contains $C+2$ actions $\mathsf{setevent}$, $\mathsf{setstart}$ and setend^i, $0 \leq i < C$, and $|F|$ actions of type reset_f. □

In the modification of FD, we introduce temporal constraints every time we generate events. For a given event e, let $\tau_e = \tau_a$ if $e = \mathsf{start}_a$ for some temporal action a, and let $\tau_e = \tau_a + d(a)$ if $e = \mathsf{end}_a$. Let $\{e_1, \ldots, e_k\}$ be a concurrent event generated during the event phase of our compilation. To ensure that the events are scheduled at the same time, we introduce the temporal constraint $\tau_{e_j} \leq \tau_{e_{j+1}}$ for each j, $1 \leq j < k$, as well as the temporal constraint $\tau_{e_k} \leq \tau_{e_1}$. In addition, for each active action a' that started before the concurrent event, we introduce the temporal constraint $\tau_{e_j} + u \leq \tau_{a'} + d(a')$ for each j, $1 \leq j < k$, where u is a slack unit of time which ensures that the end of a' takes place strictly after e_j. This last constraint is redundant since the end of a' will eventually be scheduled after the given concurrent event, but in practice it helps ensure that unsound temporal plans are pruned as soon as possible.

For two consecutive concurrent events $\{e_1, \ldots, e_k\}$ and $\{e'_1, \ldots, e'_m\}$, we introduce the constraint $\tau_{e_k} + u \leq \tau_{e'_1}$. Our modification of FD maintains an STN that is updated each time a new temporal constraint is added, and prunes a search node as soon as the temporal constraints are impossible to satisfy, i.e. when the STN contains negative cycles.

Theorem 3 (Soundness)**.** *Let π' be a plan that solves the classical planning instance $\Pi_{K,C}$ by our modified version of FD. Given π', we can always construct a temporal plan π that solves the temporal planning instance Π.*

Proof. The system can only be in one phase at a time, and we can only cycle through phases in the order $\mathsf{endphase} \rightarrow \mathsf{eventphase} \rightarrow \mathsf{startphase} \rightarrow \mathsf{endphase}$ using actions $\mathsf{setevent}$, $\mathsf{setstart}$ and setend^i. The system is initially in the end phase, and the goal state requires us to be in the start phase with no actions active (due to goal condition concur^0).

A temporal action a can start in the event phase and launch in the start phase. Specifically, the fluent $\mathsf{nstarting}_a$ is deleted by $\mathsf{dostart}^c_a$ and added by launch_a. After starting a in the event phase, we cannot end a until another subsequent event phase since the precondition ending_a of action doend^c_a is only added by finish_a, which is only applicable in the end phase. Together with the fact that no action is active in the goal, starting a implies that we have to fully cycle through all the phases at least one more time. In turn, this requires us to apply action setend^i. Due to the precondition $\mathsf{nstarting}_a$ of setend^i, we cannot start a in the event phase without launching a in the very next start phase.

Likewise, a temporal action a can finish (i.e. be scheduled for ending) in the end phase, and end in the event phase. Specifically, the fluent $\mathsf{nending}_a$ is deleted by finish_a and added by doend^c_a. After ending a in the event phase, we have to apply action $\mathsf{setstart}$ at least one more time since the goal state requires us to be in the start phase. Due to the precondition $\mathsf{nending}_a$ of $\mathsf{setstart}$, we cannot finish a in the end phase without ending a in the very next event phase.

For each $f \in F$, fluents canpre_f and caneff_f are true each time we apply action $\mathsf{setevent}$, i.e. when entering the event phase. These fluents are true in the initial state, i.e. in the end phase, and are only deleted by actions of type $\mathsf{dostart}^c_a$ and doend^c_a, which are only applicable in the event phase. The precondition $\{\mathsf{canpre}_f, \mathsf{caneff}_f\}$ of action setend^i requires us to reset f in the start phase using action reset_f, and there are no actions that delete these fluents in the end phase.

A solution plan π' for $\Pi_{K,C}$ thus has the following form:

$$\langle \underline{\mathsf{setevent}}, \mathsf{dostart}_a, \underline{\mathsf{setstart}}, \mathsf{launch}_a, \mathsf{reset}_f, \underline{\mathsf{setend}}, \ldots, \ldots, \underline{\mathsf{setend}}, \mathsf{finish}_a, \underline{\mathsf{setevent}}, \mathsf{doend}_a, \underline{\mathsf{setstart}} \rangle$$

For clarity, actions that alter the phases are underlined. We may, of course, start and launch multiple actions at once, as well as finish and end multiple actions at once. We may also start and end actions during the same event phase.

We show that each joint event induced by π' is valid. Each time a fluent f appears as an effect of an event, deleting fluents canpre_f and caneff_f prohibits f from appearing as a precondition or effect of another event in the same event phase (reset_f is not applicable until the following start phase). Likewise, each time f appears as a precondition, deleting caneff_f prohibits f from appearing as an effect of another event. Because of the mechanism for finishing and launching temporal actions, the context of a temporal action a may be added by an event simultaneous with starting a and deleted by an event simultaneous with ending a.

Since π' is reported as a solution plan for $\Pi_{K,C}$ by our modified version of FD, the resulting STN does not contain negative cycles, making it possible to satisfy all temporal constraints. Since the temporal constraints require all concurrent events to take place simultaneously and all subsequent events to take place after a given concurrent event, we can convert π' into a temporal plan π by assigning a starting time $t = -d_{i0}$ to each action a_i of π', where d_{i0} is the shortest distance in the graph from τ_i to τ_0, the temporal action that starts at time 0. This ensures that event sequence induced by π is identical to π'. Since π' solves $\Pi_{K,C}$ and ensures that no contexts of temporal actions are violated, this ensures that the goal condition G is satisfied after the execution of the temporal plan π, implying that π solves P. □

Although STP can deal with many kinds of temporal problems (sequential, with single hard envelopes, simultaneous events, ...), it is not complete. The reason precisely depends on the parameters K and C. First, there can be problems for which a given K is not enough to solve the problem; for instance, STP will not solve a problem requiring 5 concurrent actions if $K < 5$. Second, the end count C is cyclic: it reduces the risk of ignoring propositionally equal but temporally different states, but it does not remove it.

Given the dependency on K and C, an appropriate strategy for trying to solve a temporal problem using STP could consist on starting from low values of K and C and increasing them while the solution is not found.

Results

We performed an evaluation in all 10 domains of the temporal track of IPC-2014. Moreover, we added the DRIVERLOGSHIFT (DLS) domain (Coles et al. 2009), the AIA domain (Jiménez, Jonsson, and Palacios 2015), and a domain based on an STN example by Cushing et al. (2007) (from

now on, we refer to this domain as CUSHING)[2].

STP was executed for values of K in the range $1, \ldots, 4$ and with a fixed end count $C = 10$, which proved to work fine in AIA instances. We compared STP to several other planners that compile problems into classical planning: the TP planner using the same values of K, and the TPSHE planner using the LAMA-2011 setting of FD to solve the compiled instance. We also ran experiments for POPF2 (Coles et al. 2010) (the runner-up at IPC-2011), YAHSP3-MT (Vidal 2014) (the winner at IPC-2014), and ITSAT (Rankooh and Ghassem-Sani 2015).

Table 1 shows, for each planner, the IPC quality score and the coverage, i.e. the number of instances solved per domain. Experiments were executed on a Linux computer with Intel Core 2 Duo 2.66GHz processors. Each experiment had a cutoff of 10 minutes or 4GB of RAM memory.

The benchmark domains can be classified into four categories. Seven domains (DRIVERLOG, FLOORTILE, MAPANALYSER, PARKING, RTAM, SATELLITE and STORAGE) can be solved using sequences of temporal actions, i.e. there is no need for actions to execute concurrently. In these domains, TPSHE comes out on top, solving 98 instances with an IPC score of 76.44, followed by TP(1) and YAHSP3-MT. The latter can in fact *only* solve domains of this type. We remark that at IPC-2014, YAHSP3-MT solved 103 instances; one reason for this discrepancy is that we used a shorter cutoff, and YAHSP3-MT is also sensitive to input parameters.

All variants of STP performed particularly poorly in these seven domains, with the top performer being STP(1) with 34 instances solved. Compared to TP(1), which solved more than twice that number, STP(1) incorporates additional fluents and actions associated with the three phases used to simulate events. Since events are not concurrent, these additional fluents and action only hurt performance, resulting in a larger state space and an increased branching factor.

A further four domains (DLS, MATCHCELLAR, TMS and TURN&OPEN) can be solved using temporal plans that only incorporate concurrency in the form of *single hard envelopes* (Coles et al. 2009). ITSAT is the top performer in these domains, solving 60 instances with an IPC score of 56.97, followed by TPSHE with 59 instances solved. TP(1) and STP(1) cannot solve any instance since they do not allow for concurrency. TP(2) solves 40 instances, compared to 24 instances of STP(2), STP(3) and STP(4). Again, since simultaneous events are not needed, the increased size of the compilation in STP makes instances harder to solve.

The third category is represented by the CUSHING domain, which requires concurrency *not* in the form of single hard envelopes, but does not require events to take place simultaneously. In this domain, ITSAT and TPSHE cannot solve any instances, and the top performer is instead POPF2, which solves all instances. TP(3) and TP(4) also solve all instances but produce plans with longer makespan, and STP(3) and STP(4) solve 14 and 5 instances, respectively. Yet again the additional machinery of STP does not pay off. The fact that STP(4) performs much worse than STP(3) highlights the fact that the increased number of fluents and actions makes instances more difficult to solve.

[2]The code of the compilation and the domains are available at https://github.com/aig-upf/temporal-planning.

Domain	Compression	Domain	Compression
AIA	0.94	MATCHCELLAR	0.83
CUSHING	0.85	PARKING	0.66
DLS	0.66	SATELLITE	0.72
MAPANALYSER	0.72		

Table 2: Average level of compression of plans returned by STP(K) in relation to the best results of the other planners.

The only domain in which STP excels is AIA, in which many instances do require concurrency in the form of simultaneous events, representing the fourth and final category. STP(4) is the only planner that can solve all 25 instances. POPF2 and TP(4) solve all 10 instances that do not require simultaneous events, and all planners solve the 3 instances that can be solved using sequential temporal plans.

Although STP performs poorly in most domains, we still argue that the ability to deal with simultaneous events in a forward-search temporal planner is a contribution to the field of temporal planning, for two reasons. The benchmark domains commonly used in temporal planning present a clear bias towards domains that are challenging from a combinatorial perspective, but the temporal aspect is almost trivial, which is the reason that few temporal planners are able to handle concurrent temporal actions in a robust manner. The reason STP performs poorly is mostly because of the combinatorial aspect, since the ability to deal with simultaneous events comes with a price: an increased number of fluents and actions, resulting in a larger search space.

The second reason that STP may have an impact is if researchers develop a way to analyze temporal planning domains prior to solving them. If the analysis shows that a domain is temporally simple, then there is no need to run STP, since it will always perform worse than several other alternatives. Only when the analysis determines that more intricate forms of concurrency are required , STP will be executed.

Since STP can output plans with simultaneous events, we compare the number of events in its plans with the number of events in other planners. Table 2 shows how much STP(K) planners compress (on average) plans in comparison to the other planners (TPSHE, TP(K), POPF2, YAHSP3-MT and ITSAT). Given a problem, we compare the number of events in plans output by STP(K) against the number of events output by a planner from the other group. The comparison is done just if the problem is solved by both groups.

From the table, we observe that solutions returned by STP always contain less events than the other planners. In domains like DLS and PARKING, the solutions contain around 30% less events compared to the other planners.

Related Work

Several authors have provided theoretical justification for splitting durative actions into classical actions (Cooper, Maris, and Régnier 2013) and proposed a compilation for doing so. An early approach, LPGP (Long and Fox 2003), turned out to be unsound and incomplete since it failed to (1) ensure that temporal actions end before reaching the goal,

	TPSHE	TP(1)	TP(2)	TP(3)	TP(4)	STP(1)	STP(2)	STP(3)	STP(4)	POPF2	YAHSP3-MT	ITSAT
AIA[25]	3/3	3/3	6.5/8	7.5/9	8.5/10	3/3	17.17/22	19.51/24	**23.5/25**	10/10	3/3	3/3
CUSHING[20]	0/0	0/0	0/0	4.07/**20**	4.93/**20**	0/0	0/0	3.31/14	2.28/5	**20/20**	0/0	0/0
DRIVERLOG[20]	**14.78/15**	1.42/5	0.93/3	1.08/4	0.91/3	0/0	0/0	0/0	0/0	0/0	2.31/4	1/1
DLS[20]	9.37/11	0/0	10/10	7.7/9	8.06/9	0/0	3.78/4	3.9/4	3.49/4	7/7	0/0	**16.18/19**
FLOORTILE[20]	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	4.93/5	**19.7/20**
MAPANALYSER[20]	**17.38/20**	10.16/19	13.08/**20**	12.34/**20**	12.02/19	9.18/19	9.81/17	10.09/16	7.69/12	0/0	1/1	0/0
MATCHCELLAR[20]	15.72/**20**	0/0	15.71/**20**	15.71/**20**	15.71/**20**	0/0	15.71/**20**	15.71/**20**	15.71/**20**	**20/20**	0/0	18.91/19
PARKING[20]	6.73/**20**	5.59/19	5.79/17	5.67/17	5.33/16	1.67/6	1.79/6	1.93/6	1.93/6	12/13	**16.84/20**	0.96/6
RTAM[20]	**16/16**	4.91/11	2.45/6	2.73/6	2.79/6	0/0	0/0	0/0	0/0	0/0	0/0	0/0
SATELLITE[20]	**16.63**/18	7.99/19	4.97/13	5.04/13	4.67/12	2.31/6	0/0	0/0	0/0	2.92/3	13.82/**20**	1.68/7
STORAGE[20]	4.92/**9**	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	3.91/**9**	**9/9**
TMS[20]	0.06/9	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	**16/16**
TURN&OPEN[20]	**15.53/19**	0/0	5.05/10	5.03/10	5.19/10	0/0	0/0	0/0	0/0	7.31/8	0/0	5.88/6
Total	**120.12/160**	33.07/76	64.49/107	66.87/128	68.11/125	16.16/34	48.26/69	54.45/84	54.61/72	79.22/81	45.8/62	92.3/106

Table 1: IPC quality score / coverage per domain for each planner. Total number of instances of each domain between brackets.

(2) ensure that the contexts of temporal actions are not violated, and (3) ensure that temporal constraints are preserved.

Rintanen (2007) proposed a compilation from temporal to classical planning that explicitly represents time units as objects. The compilation includes classical actions that start temporal actions, and keeps track of time elapsed in order to determine when temporal actions should end. The compilation only handles integer duration, potentially making the planner incomplete when events have to be scheduled fractions of time units apart and, as far as we know, this compilation has never been implemented as part of an actual planner.

The planners most similar to ours are TPSHE and TP (Jiménez, Jonsson, and Palacios 2015). Both planners are based on compiling temporal planning problems to classical planning problems. TPSHE only handles instances where required concurrency is in the form of single hard envelopes. In contrast, TP partially compiles temporal actions into classical planning and introduces an STN into the Fast Downward classical planner to enforce temporal constraints. POPF (Coles et al. 2010) and OPTIC (Benton, Coles, and Coles 2012) also use STNs. POPF encodes the STN using linear programming which allows it to compute plans with actions that cause continuous linear numeric changes. OPTIC encodes the STN as a mixed integer problem which additionally allows handling temporally dependent costs.

With respect to planners that perform explicit state-space search, an interesting direction is the exploitation of *l*andmarks. This group includes the TEMPLM planner (Marzal, Sebastia, and Onaindia 2014) that discovers classical landmarks from a temporal instance, and builds a landmark graph that expresses the temporal relations between these landmarks. This approach has proven useful to detect unsolvable instances under deadline constraints. However, in the absence of tightly-constrained dead-ends it does not yield significant benefits over classical causal landmarks. Karpas et al. (2015) do not rely on the presence of deadlines to discover landmarks that are not causal landmarks and define notions of temporal fact landmarks, which state that some fact must hold between two given time points, and temporal action landmarks, which state that the start or end of an action must occur at a given time point.

Satisfiability checking is also an important trend in temporal planning. Similarly to the SAT-based approaches for classical planning, temporal planning instances can be encoded as SAT problems. The SAT encoding for temporal planning instances is more elaborated since it involves choosing the start times of actions and verifying the temporal constraints between them. Moreover, PDDL induces temporal gaps between consecutive interdependent actions that effectively doubles the number of joint events required to solve a given temporal planning instance and hence affecting the performance of SAT-based search approaches. The ITSAT planner (Rankooh and Ghassem-Sani 2015) deals with this issue by abstracting out the duration of actions and separating action sequencing from scheduling. ITSAT assumes that actions can have arbitrary duration and encodes the abstract problem into a SAT formula to generate a causally valid plan without checking the existence of a valid schedule. To find a temporally valid plan, ITSAT then tries to schedule the causally valid plan solving the STN defined by the duration of the actions in the plan. If the STN can be solved, ITSAT returns a valid plan, but if not, ITSAT adds the sequence of events that led to the unsolvable STN as new blocking clauses in the SAT encoding. The process is repeated until a valid temporal plan is achieved. A different approach is producing a SAT encoding that integrates action sequencing and scheduling. Recently the modeling language NDL has been proposed as an alternative to PDDL with the aim of producing a SAT Modulo Theories encoding where action sequencing and scheduling are tightly integrated (Rintanen 2015a). Rintanen (2015b) showed that while PDDL forces temporal gaps in action scheduling (which have a performance penalty), NDL avoids such gaps using the notion of resources.

Conclusions

We introduced a compilation from temporal planing with simultaneous events to classical planning, and proved it returns sound plans if used in a forward-search planner that maintains STNs for checking temporal consistency. We showed that our approach performs well in domains requiring simultaneous events, although it is not competitive in domains requiring simpler forms of concurrency. We only use the actual durations of actions in the STN to verify consistency, making our compilation closer to the state-based definition of temporal planning (Rintanen 2007), where durations did not appear. Moreover, we avoid the proposed counters on the remaining duration of actions, instead relying on

temporal constraints to enforce soundness.

Solving rich planning problems using classical planning usually involves modeling trajectories that can be translated to rich plans, and additional constraints to enforce the classical planning trajectories correspond to sound rich plans (Baier, Bacchus, and McIlraith 2009; Palacios and Geffner 2009). These mechanisms could be challenging for state-of-the-art planners, developed around handmade benchmarks.

Regarding the cost of compilation, the increase in the number of fluents and actions is polynomial, ensuring that the existence of a classical plan remains in PSPACE. The STN, used by STP to verify the consistency of temporal constraints, grows linearly with the length of the plan, as grows the plan representation built by *state-of-the-art* planners.

Acknowledgments

This work has been supported by the Maria de Maeztu Units of Excellence Programme (MDM-2015-0502).

References

Allen, J. F. 1983. Maintaining Knowledge about Temporal Intervals. *Commun. ACM* 26(11):832–843.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, 28–35.

Cesta, A., and Oddi, A. 1996. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Proceedings of the Third International Workshop on Temporal Representation and Reasoning, TIME-96, Key West, Florida, USA, May 19-20, 1996*, 45–50.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artif. Intell.* 173(1):1–44.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 42–49.

Cooper, M. C.; Maris, F.; and Régnier, P. 2013. Managing Temporal Cycles in Planning Problems Requiring Concurrency. *Computational Intelligence* 29(1):111–128.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is Temporal Planning Really Temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 1852–1859.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artif. Intell.* 49(1-3):61–95.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), 15-17 November 2004, Boca Raton, FL, USA*, 294–301.

Jiménez, S.; Jonsson, A.; and Palacios, H. 2015. Temporal Planning With Required Concurrency Using Classical Planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, 129–137.

Karpas, E.; Wang, D.; Williams, B. C.; and Haslum, P. 2015. Temporal Landmarks: What Must Happen, and When. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, 138–146.

Long, D., and Fox, M. 2003. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, 52–61.

Marzal, E.; Sebastia, L.; and Onaindia, E. 2014. On the Use of Temporal Landmarks for Planning with Deadlines. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35:623–675.

Rankooh, M. F., and Ghassem-Sani, G. 2015. ITSAT: An Efficient SAT-Based Temporal Planner. *J. Artif. Intell. Res.* 53:541–632.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 280–287.

Rintanen, J. 2015a. Discretization of Temporal Models with Application to Planning with SMT. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 3349–3355.

Rintanen, J. 2015b. Models of Action Concurrency in Temporal Planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1659–1665.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Compact Tree Encodings for Planning as QBF

Olivier Gasquet, Dominique Longin, Frédéric Maris, Pierre Régnier, Maël Valais

IRIT – University of Toulouse
Toulouse, France

Abstract

Considerable improvements in the technology and performance of SAT solvers has made their use possible for the resolution of various problems in artificial intelligence, and among them that of generating plans. Recently, promising Quantified Boolean Formula (QBF) solvers have been developed and we may expect that in a near future they become as efficient as SAT solvers. So, it is interesting to use QBF language that allows us to produce more compact encodings. We present in this article a translation from STRIPS planning problems into quantified propositional formulas. We introduce two new Compact Tree Encodings: CTE-EFA based on Explanatory frame axioms, and CTE-OPEN based on causal links. Then we compare both of them to CTE-NOOP based on No-op Actions proposed in (Cashmore, Fox, and Giunchiglia 2012). In terms of execution time over benchmark problems, CTE-EFA and CTE-OPEN always performed better than CTE-NOOP.

Introduction

An algorithmic approach for plans synthesis is automated compilation (i.e., transformation) of planning problems. In the SATPLAN planner (Kautz and Selman 1992), a planning problem is transformed into a propositional formula whose models, corresponding to solution plans, can be found using a SAT solver. The SAT approach searches for a solution-plan of fixed length k. In case of failure to find such a plan, this length is increased before restarting the search for a solution. In the classical framework, the complexity of finding a solution to any problem is PSPACE-hard, but the search for a fixed-size solution becomes NP-hard (Bylander 1994). This compilation approach directly benefits from improvements in SAT solvers[1]. The most obvious example is the planner BLACKBOX (Kautz and Selman 1998; 1999) (and its successors SATPLAN'04 (Kautz 2004) and SATPLAN'06 (Kautz, Selman, and Hoffmann 2006)). These planners won the optimal (in the number of plan steps) planning track of the International Planning Competitions[2] IPC-2004 and IPC-2006. This was unexpected because these planners were essentially updates of BLACKBOX and did not include any real novelty: improved performance was mainly due to progresses in the underlying SAT solver.

Numerous improvements of this original approach have been proposed since then, in particular via the development of more compact and efficient encodings (Kautz and Selman 1996; Ernst, Millstein, and Weld 1997; Mali and Kambhampati 1998; 1999; Rintanen 2003; Rintanen, Heljanko, and Niemelä 2004; 2006; Rintanen et al. 2008). Following these works, numerous other similar techniques for encoding planning problems have been developed: Linear Programming (LP) (Wolfman and Weld 1999), Constraint Satisfaction Problems (CSP) (Do and Kambhampati 2001), SAT Modulo Theories (SMT) (Shin and Davis 2005; Maris and Régnier 2008; Rintanen 2015). More recently, a Quantified Boolean Formulas (QBF) approach had been proposed by (Rintanen 2007; Cashmore, Fox, and Giunchiglia 2012).

Currently SAT solvers outperform QBF solvers and the SAT approach is the most effective because SAT solvers and encodings have been greatly improved since 1992. However, over the past decade, there has been a growing interest in the QBF approach. The competitive evaluation of QBF solvers QBFEVAL[3] is now a joint event with the international SAT conference and QBF solvers improve regularly. QBFEVAL'16 had more participants than ever and QBF-related papers represented 27% of all papers published at SAT'16. Some promising techniques have been adapted to QBF solving such as counterexample guided abstraction refinement (CEGAR) (Clarke et al. 2003; Janota and Marques-Silva 2015; Janota et al. 2016; Rabe and Tentrup 2015). For comparable SAT / QBF encodings, the QBF approach also have the advantage to generate more compact formulas (Cashmore, Fox, and Giunchiglia 2012). Even if the QBF approach is not as efficient as the SAT approach, it deserves the interest of the community.

Our paper shows that beyond the implementation of solvers, further work must be done to improve the encodings. In particular, we introduce two new QBF Compact Tree Encodings of STRIPS planning problems: CTE-EFA based on Explanatory frame axioms, and CTE-OPEN based on causal links. Then we compare both of them to CTE-

[1] http://www.satcompetition.org/

[2] http://www.icaps-conference.org/index.php/Main/Competitions

[3] http://www.qbflib.org/index_eval.php

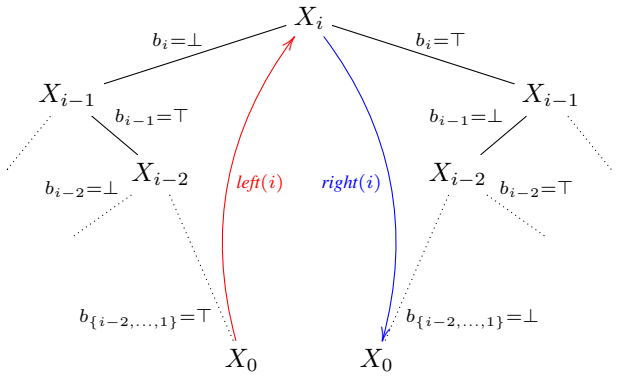


Figure 1: Both possible transitions in a CTE following the branching structure of a QBF: $X_0 \rightarrow X_i$ (from leaf to node on the left) and $X_i \rightarrow X_0$ (from node to leaf on the right). Note that i refers to any level (except for the leaf layer), not only the root.

NOOP based on No-op Actions proposed in (Cashmore, Fox, and Giunchiglia 2012). In terms of execution time over benchmark problems, CTE-EFA and CTE-OPEN always performed better than CTE-NOOP.

Planning as QBF

Two different approaches of planning as QBF have been proposed by (Cashmore, Fox, and Giunchiglia 2012): Flat Encoding, that was first introduced by (Rintanen 2001) as an approach to general reachability, and Compact Tree Encoding (CTE). Cashmore, Fox, and Giunchiglia showed in (2012) that Compact Tree Encodings outperform Flat Encodings. Both these planning encodings make use of the branching structure of the QBF to reuse a single set of clauses that describes a single step in the plan. The two assignments inside each universal variable represent the first and second half of the plan split around that branch. The assignments to each existential set represent action choices within a single step.

Preliminary Definitions

Let $\mathcal{F}$ be a finite set of *fluents* (atomic propositions). A STRIPS *planning problem* is a tuple $\langle I, \mathcal{A}, G\rangle$ where $I \subseteq \mathcal{F}$ is the set of initial fluents, $G \subseteq \mathcal{F}$ is the set of goal fluents and $\mathcal{A}$ is the set of actions. An action $a \in \mathcal{A}$ is a tuple $\langle Pre(a), Add(a), Del(a)\rangle$ where

- $Pre(a) \subseteq \mathcal{F}$ is the set of fluents required to be true in order to execute a,
- $Add(a) \subseteq \mathcal{F}$ and $Del(a) \subseteq \mathcal{F}$ are the sets of fluents respectively added and removed by the action a.

All QBF encodings studied in this paper use propositional variables for actions. The Compact Tree Encoding proposed in (Cashmore, Fox, and Giunchiglia 2012) is based on the *planning graph* introduced in (Blum and Furst 1997) and uses additional no-op actions as frame axioms. We denote it by CTE-NOOP. Considering every action as a propositional variable, we define a set of propositional variables X, given by $X = \mathcal{A} \cup \{noop_f \mid f \in \mathcal{F}\}$.

In a CTE formula, we want to select two consecutive steps in order to define transitions (Figure 1). For each depth i of the tree, X_i denotes a copy of the set of variables X.

For CTE-NOOP, there exists a single variable $a_i \in X_i$ for each action and a single variable $noop_{f,i} \in X_i$ (no-op action) for each fluent used to determine a transition in the plan. At a same depth i, the value of these variables depends on the node (corresponding to a step in the plan) selected by the values of upper universal branching variables $b_{i+1} \ldots b_{depth}$. More details can be find in the slides[4].

An upper bound on the plan length is $2^{k+1} - 1$, where k is the number of alternations of quantifiers in the quantified boolean formula associated with the planning problem. In the case of CTE, k is also the compact tree depth. The number of possible states for a given planning problem is bounded by $2^{|\mathcal{F}|}$. Then, the existence of a plan can be determined using a linear QBF encoding with at most $k =\mid \mathcal{F} \mid$.

In the sequel, we propose two new encodings of planning problems into QBF. The first, denoted by CTE-OPEN, is based on causal links (plan-space). It has been first introduced by (Mali and Kambhampati 1999) but needs to be adapted using additional variables for open conditions. The second, denoted by CTE-EFA, is based on explanatory frame axioms (state-space) first introduced by (Kautz and Selman 1992) and uses variables for fluents as well as for actions.

Causal Link Encoding: CTE-OPEN

The plan-space encodings of (Mali and Kambhampati 1999) cannot be directly adapted to the CTE. All these encodings refer to three indexed (not necessarily consecutive) steps of the plan. This is not possible in a CTE because each rule can refer to only one branch of the tree. To overcome this problem, it would be possible to duplicate the tree by adding, for each branching variable b_i, two more branching variables b'_i and b''_i, and for each node X_i, two node copies X'_i and X''_i, and equivalence rules $\bigwedge_{x_i \in X_i} \big((x_i \leftrightarrow x'_i) \wedge (x_i \leftrightarrow x''_i)\big)$. Unfortunately, this would increase the branching factor unnecessarily. So, we propose a new plan-space encoding which allows us to only refer to consecutive steps in the plan.

For every fluent $f \in \mathcal{F}$, we create a propositional variable $open_f$ to express that f holds in some previous step and must be protected at least until the current step. In Figure 2, the fluent f is an *open condition* in step S_i, entailing that either $f \in I$ or an action a' which adds f is executed in a previous step S_{i-k}. Open conditions are propagated backwards until the initial state or some step in which they are added by an action.

We define the set of "open" variables, denoted as Δ, as $\Delta = \{open_f \mid f \in \mathcal{F}\}$. Considering every action as a propositional variable, we define a set of propositional variables X, given by $X = \mathcal{A} \cup \Delta$.

[4] https://www.irit.fr/~Frederic.Maris/documents/coplas2018/slides.pdf

Quantifiers For each depth i of the tree, X_i denotes a copy of the set of variables X. It exists a single variable $a_i \in X_i$ for each action used to determine last transition in the plan and a single variable $open_{f,i} \in X_i$ for each fluent used to determine if f is an open condition. At a same depth i, the value of these variables depends on the node (corresponding to a step in the plan) selected by the values of upper universal branching variables $b_{i+1} \dots b_{depth}$.

$$\begin{array}{l} \exists_{a \in \mathcal{A}} a_{depth}. \exists_{f \in \mathcal{F}} open_{f,depth}. \forall b_{depth}. \\ \exists_{a \in \mathcal{A}} a_{depth-1}. \exists_{f \in \mathcal{F}} open_{f,depth-1}. \forall b_{depth-1}. \\ \dots \\ \exists_{a \in \mathcal{A}} a_1. \exists_{f \in \mathcal{F}} open_{f,1}. \forall b_1. \exists_{a \in \mathcal{A}} a_0. \exists_{f \in \mathcal{F}} open_{f,0}. \end{array}$$

In the following, a *node* now refers to a non-leaf node (i.e., an inner node) and *depth* is the depth of the tree. The predecessor of a node at level i is the rightmost leaf of the left subtree. The successor of a node at level i is the leftmost leaf of the right subtree. In order to select these transitions, we introduce the leaf-to-node operator $left(i)$ defined as:

$$left(i) \equiv \neg b_i \wedge \bigwedge_{j=1}^{i-1} b_j.$$

Symmetrically, we introduce the node-to-leaf operator $right(i)$ defined as:

$$right(i) \equiv b_i \wedge \bigwedge_{j=1}^{i-1} \neg b_j.$$

Open conditions If an action a is executed in a step of the plan, then each precondition of a must be an open condition at this step (i.e., a causal link is required for this precondition).

$$\bigwedge_{i=0}^{depth} \bigwedge_{a \in \mathcal{A}} \left(a_i \Rightarrow \bigwedge_{f \in Pre(a)} open_{f,i} \right)$$

In the last plan step leading to the goal (i.e. the rightmost leaf of the tree), all the goal fluents must be either open conditions or added by actions executed in this step.

$$\bigwedge_{i=1}^{depth} b_i \Rightarrow \bigwedge_{f \in G} \left(open_{f,0} \vee \bigvee_{\substack{a \in \mathcal{A} \\ f \in Add(a)}} a_0 \right)$$

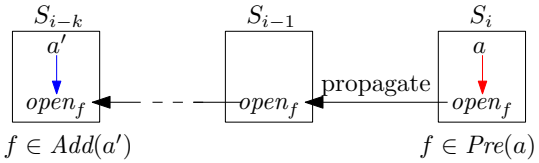


Figure 2: Causal link: a'produces f for a.

Propagate and close No conditions should remain open in the first plan step (i.e. the leftmost leaf of the tree) if it is not provided in the initial state.

$$\bigwedge_{i=1}^{depth} \neg b_i \Rightarrow \bigwedge_{f \in \mathcal{F} \setminus I} \neg open_{f,0}$$

Any open condition in a step must either remain open or be added (closed) by an action in the previous step.

$$\bigwedge_{i=1}^{depth} \bigwedge_{f \in \mathcal{F}} \left(\left(open_{f,i} \wedge left(i) \right) \Rightarrow \left(open_{f,0} \vee \bigvee_{\substack{a \in \mathcal{A} \\ f \in Add(a)}} a_0 \right) \right)$$

$$\bigwedge_{i=1}^{depth} \bigwedge_{f \in \mathcal{F}} \left(\left(open_{f,0} \wedge right(i) \right) \Rightarrow \left(open_{f,i} \vee \bigvee_{\substack{a \in \mathcal{A} \\ f \in Add(a)}} a_i \right) \right)$$

Protect open conditions An open condition in a given step cannot be removed in the previous step. This guarantees not to break any causal link in the plan.

$$\bigwedge_{i=1}^{depth} \bigwedge_{f \in \mathcal{F}} \left(\left(open_{f,i} \wedge left(i) \right) \Rightarrow \bigwedge_{\substack{a \in \mathcal{A} \\ f \in Del(a)}} \neg a_0 \right)$$

$$\bigwedge_{i=1}^{depth} \bigwedge_{f \in \mathcal{F}} \left(\left(open_{f,0} \wedge right(i) \right) \Rightarrow \bigwedge_{\substack{a \in \mathcal{A} \\ f \in Del(a)}} \neg a_i \right)$$

Prevent negative interactions In a given step, if an action removes a fluent which is needed or added by another action, then these two actions cannot be both executed in this step.

$$\bigwedge_{i=0}^{depth} \bigwedge_{a \in \mathcal{A}} \bigwedge_{f \in (Add(a) \cup Pre(a))} \bigwedge_{\substack{a' \in \mathcal{A} \\ a \neq a' \\ f \in Del(a')}} (\neg a_i \vee \neg a'_i)$$

State-Space Encoding: CTE-EFA

In this encoding, we define the set of propositional variables as $X = \mathcal{A} \cup \mathcal{F}$. Each step is now defined by a transition (as in CTE-OPEN) as well as the resulting state (valuation of the fluents in $\mathcal{F}$). The formula is an adaptation to the CTE of the well known state-space SAT encoding rules based on explanatory frame axioms of (Kautz and Selman 1992).

Quantifiers At each depth i of the tree, it exists a single variable a_i for each action used to determine last transition in the plan and a single variable f_i for each fluent used to determine the state. At a same depth i, the values of these variables depend on the node (corresponding to a transition

in the plan and the resulting state) selected by the values of upper universal branching variables $b_{i+1} \dots b_{depth}$.

$$\begin{array}{l} \exists\limits_{a\in\mathcal{A}} a_{depth}.\ \exists\limits_{f\in\mathcal{F}} f_{depth}.\forall b_{depth}. \\ \exists\limits_{a\in\mathcal{A}} a_{depth-1}.\ \exists\limits_{f\in\mathcal{F}} f_{depth-1}.\forall b_{depth-1}. \\ \qquad\qquad \dots \\ \exists\limits_{a\in\mathcal{A}} a_1.\ \exists\limits_{f\in\mathcal{F}} f_1.\forall b_1.\ \exists\limits_{a\in\mathcal{A}} a_0.\ \exists\limits_{f\in\mathcal{F}} f_0. \end{array}$$

Goal In the state after the last plan transition (i.e. the rightmost leaf of the tree), all goal fluents must be achieved.

$$\bigwedge_{i=1}^{depth} b_i \Rightarrow \bigwedge_{f\in G} f_0$$

Conditions and effects of actions If an action a is executed in a transition of the plan, then each effect of a occurs in the resulting state and each condition of a is required in the previous state.

$$\bigwedge_{i=0}^{depth}\bigwedge_{a\in\mathcal{A}}\left(a_i \Rightarrow \left(\bigwedge_{f\in Add(a)} f_i\right) \wedge \left(\bigwedge_{f\in Del(a)} \neg f_i\right)\right)$$

$$\bigwedge_{i=1}^{depth}\bigwedge_{a\in\mathcal{A}}\left(a_i \wedge \mathit{left}(i) \Rightarrow \bigwedge_{f\in Pre(\mathcal{A})} f_0\right)$$

$$\bigwedge_{i=1}^{depth}\bigwedge_{a\in\mathcal{A}}\left(a_0 \wedge \mathit{right}(i) \Rightarrow \bigwedge_{f\in Pre(\mathcal{A})} f_i\right)$$

Moreover, an action which do not have all conditions in initial state cannot be executed in the first plan transition (i.e. the leftmost leaf of the tree):

$$\bigwedge_{i=1}^{depth} \neg b_i \Rightarrow \bigwedge_{\substack{a\in\mathcal{A}\\ Pre(a)\not\subset I}} \neg a_0$$

Explanatory frame axioms If the value of a fluent changes between two consecutive states, then an action which produces this change is executed in the plan transition between these states.

$$\bigwedge_{i=1}^{depth}\bigwedge_{f\in\mathcal{F}}\left((\neg f_0 \wedge f_i \wedge \mathit{left}(i)) \Rightarrow \left(\bigvee_{\substack{a\in\mathcal{A}\\ f\in Add(a)}} a_i\right)\right)$$

$$\bigwedge_{i=1}^{depth}\bigwedge_{f\in\mathcal{F}}\left((\neg f_i \wedge f_0 \wedge \mathit{right}(i)) \Rightarrow \left(\bigvee_{\substack{a\in\mathcal{A}\\ f\in Add(a)}} a_0\right)\right)$$

$$\bigwedge_{i=1}^{depth}\bigwedge_{f\in\mathcal{F}}\left((f_0 \wedge \neg f_i \wedge \mathit{left}(i)) \Rightarrow \left(\bigvee_{\substack{a\in\mathcal{A}\\ f\in Del(a)}} a_i\right)\right)$$

$$\bigwedge_{i=1}^{depth}\bigwedge_{f\in\mathcal{F}}\left((f_i \wedge \neg f_0 \wedge \mathit{right}(i)) \Rightarrow \left(\bigvee_{\substack{a\in\mathcal{A}\\ f\in Del(a)}} a_0\right)\right)$$

An extra rule is also required to describe explanatory frame axioms for the first plan transition from initial state (i.e. the leftmost leaf of the tree):

$$\bigwedge_{f\in\mathcal{F}\setminus I}\left(\left(f_0 \wedge \bigwedge_{i=1}^{depth} \neg b_i\right) \Rightarrow \bigvee_{\substack{a\in\mathcal{A}\\ f\in Add(a)\\ Pre(a)\subset I}} a_0\right)$$

$$\bigwedge_{f\in I}\left(\left(\neg f_0 \wedge \bigwedge_{i=1}^{depth} \neg b_i\right) \Rightarrow \bigvee_{\substack{a\in\mathcal{A}\\ f\in Del(a)\\ Pre(a)\subset I}} a_0\right)$$

Prevent negative interactions Unlike in CTE-NOOP and CTE-OPEN, contradictory effects are already disallowed by previous rules (effects of actions). Then, this rule only need to prevent interactions between conditions and deletes of actions. If an action removes a fluent which is needed by another action, then these two actions cannot be both executed in a same plan transition.

$$\bigwedge_{i=0}^{depth}\bigwedge_{a\in\mathcal{A}}\bigwedge_{f\in Pre(a)}\bigwedge_{\substack{a'\in\mathcal{A}\\ a\neq a'\\ f\in Del(a')}} (\neg a_i \vee \neg a'_i)$$

Experimental Trials

To compare these three encodings on a same basis we used our translator TouIST[5] (Comte et al. 2015) that can use several QBF solvers. We ran all available STRIPS IPC benchmarks (1 through 8, except for the 7th which was not available and authors did not answer) on an Intel Xeon CPU E7-8890 v4 @ 2.20GHz, 512 GB of RAM. The domains tested include Gripper, Logistics, Mystery, Blocks, Elevator, Depots, DriverLog, ZenoTravel, FreeCell, Airport, Pipesworld-NoTankage, Pipesworld-Tankage, PSR, Satellite, OpenStacks, Pathways, Rovers, Storage, TPP, Trucks, ChildSnack, Hiking, VisitAll and the non-IPC Ferry.

We tried to consider as many QBF solvers as possible using the QBFEval 2017 as a reference. Qute (version of 2017-07-09, based on dependency learning QCDCL) and CaQE (version of 2017-07-08, based on CEGAR clausal

[5] https://www.irit.fr/touist

abstraction) were not able to give a valuation for the outer existential quantifier. AIGSolve and Qell weren't available for download. GhostQ was skipped (but we should have included it). DepQBF (version 6.03 of 2017-08-02, based on generalized Q-resolution, described in (Lonsing and Egly 2017)) and RAReQS (version 1.1 of 2013-05-07, based on a CEGAR approach, detailed in (Janota et al. 2012)) were the only solvers left. RAReQS was consistently twice as fast as DepQBF, we thus dismissed DepQBF and only shown results for RAReQS. Finally, we did not apply any QBF preprocessor (e.g., Bloqqer).

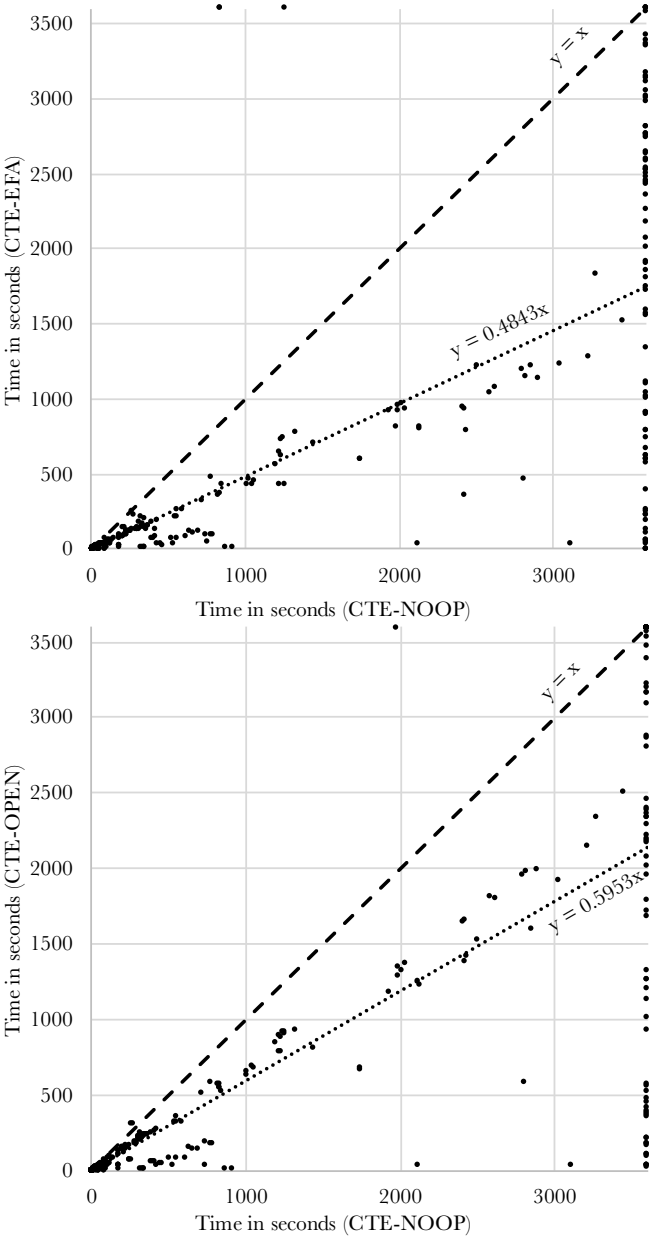


Figure 3: Plan decision time (EFA/OPEN vs NOOP).

We ran these benchmarks using our new encodings CTE-EFA and CTE-OPEN as well as the state-of-the-art compact tree encoding (CTE-NOOP). We compared them two-by-two by considering the time needed to prove the existence of a plan (decision time, Figure 3) and the overall time required to obtain a plan (extraction time, Figure 5). The "decision" step consists of launching incrementally the QBF solver on a CTE of increasing depth until the solver returns true or reaches the upper bound (total number of fluents). The "extraction" step consists of one solver launch per node of the tree in order to retrieve the plan. Each experiment had a 60 minutes[6] timeout for searching the plan and 60 minutes for extracting it. The benchmark results are available as an Excel file[7].

The results show that our encodings CTE-EFA and CTE-OPEN are more efficient than CTE-NOOP both in plan existence as well as in plan extraction. CTE-EFA by a factor of 2.1 (1/0.4843) and CTE-OPEN by a factor of 1.7 (1/0.5953). Also, the comparison between CTE-EFA and CTE-OPEN (Figure 4, Figure 6) consistently shows that CTE-EFA outperforms CTE-OPEN by a factor of 1.4 (1/0.7266). Table 1 gives a summary of the benchmark results.

Contrary to what happens with flat encodings, the gain over CTE-NOOP cannot be explained by the difference in quantifier alternations as the depth is the same in the three encodings. However, the way actions are represented in these encodings may explain this difference.

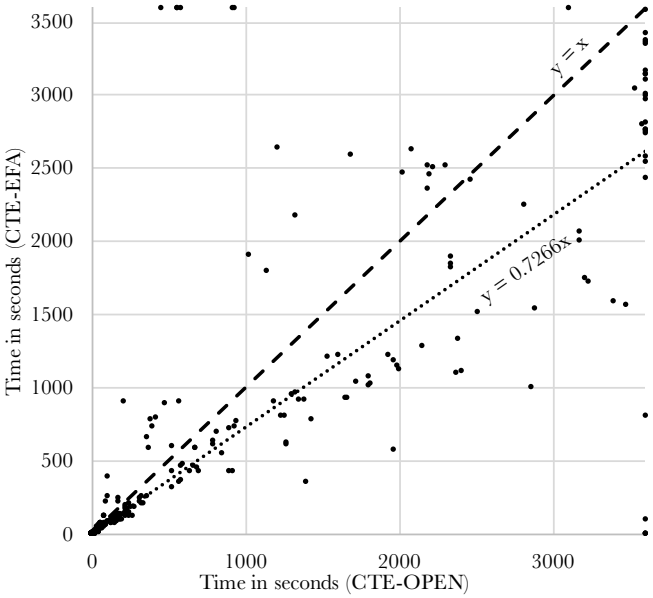


Figure 4: Plan decision time (EFA vs OPEN).

[6]The grounding step (i.e., action instantiation) is not included in the elapsed time.

[7]https://www.irit.fr/~Frederic.Maris/documents/coplas2018/results.xls

Encoding	Solved problems	Decision Time	Literals	Clauses	Transitions-over-nodes ratio
CTE-NOOP	412 over 2112 (20%)	0%	0%	0%	30%
CTE-EFA	463 over 2112 (22%)	-55%	-26%	+15%	47%
CTE-OPEN	445 over 2112 (21%)	-41%	-2%	-28%	17%

Table 1: Comparison of the presented encodings across 65 STRIPS domains from IPC 1 through 8 (IPC 7 excepted) with a total of 2112 problems. Decision time, literals count, clauses count and the transitions-over-nodes ratio are averages. The transitions-over-nodes ratio measures the quantity (in average) of transition-based constraints over node-based constraints.

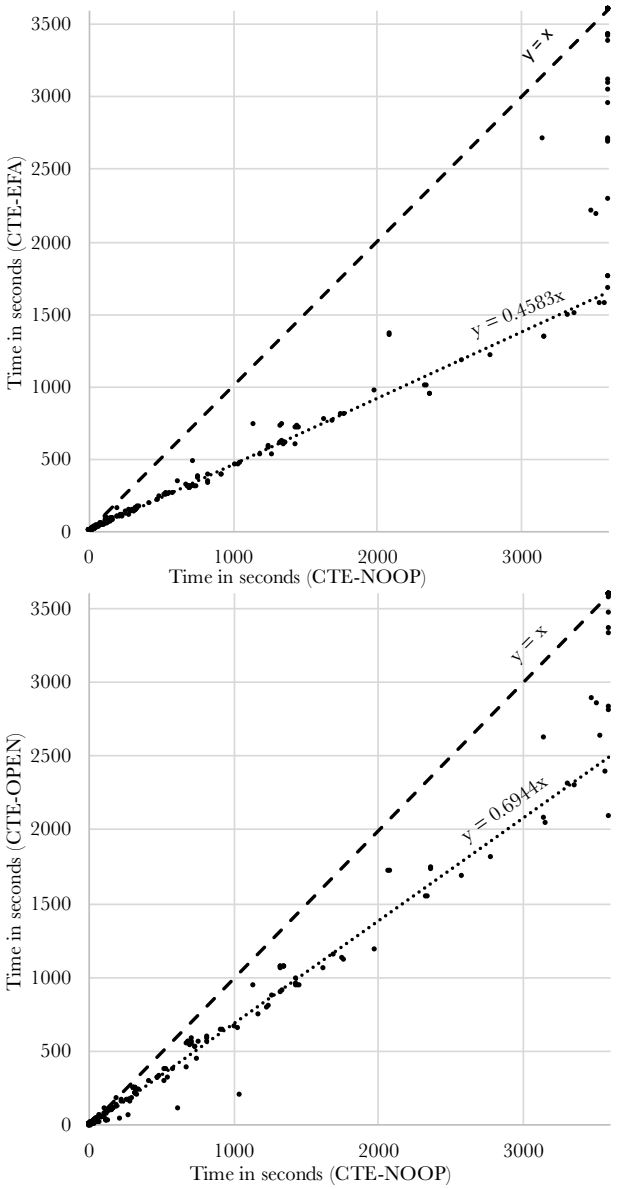


Figure 5: Plan extraction time (EFA/OPEN vs NOOP).

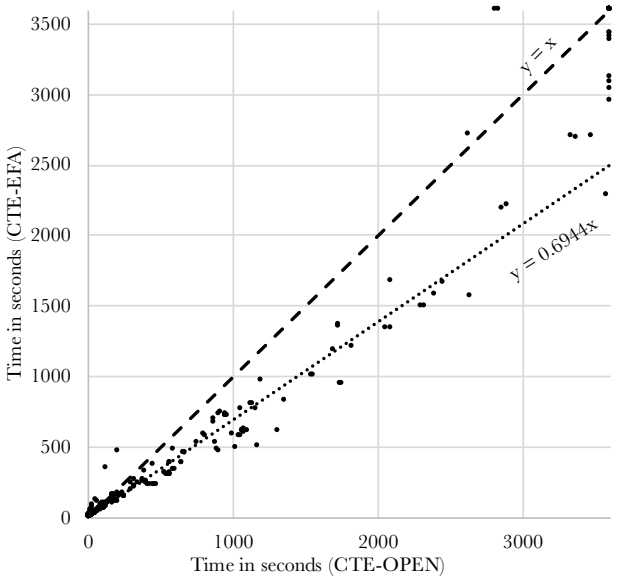


Figure 6: Plan extraction time (EFA vs OPEN).

Discussion

In order to identify the source of these improvements, we propose two hypothesis:

Hypothesis 1 "The performance gain is correlated to a decrease in the number of clauses and/or literals across encodings". Although the size of the problem is known to be noticeably non-correlated with its hardness in SAT, we wondered if we could see the same non-correlation. As shown in Table 1, we do not observe any clear tendency: CTE-EFA tends to have a slightly higher number of clauses (+15%) than CTE-NOOP although having less variables (-26%). CTE-OPEN has the same number of literals and much less clauses than CTE-NOOP, but resulting in a lower performance gain (-41%) than CTE-EFA (-55%). This non-correlation leads us to reject this hypothesis.

Hypothesis 2 "The performance gain is due to a difference in the number of transition-based constraints compared to the number of node-based constraints". Intuitively, one can think that a lower ratio of transition-based constraints over node-based constraints would ease the solving pro-

cess: in node-based constraints, a clause has the same context[8] across the whole QBF expansion. In branch constraints, the corresponding clause has different contexts depending on the selected branch. The idea is that clauses based on different contexts slow the solver down. As displayed in Table 1, this hypothesis does not appear to be correct experimentally: although CTE-OPEN shows a lower transitions-to-nodes ration, it does not lead to the best performance gain. On the contrary, CTE-EFA has a poorer ratio, although being the most efficient compared to CTE-NOOP. We thus refute this hypothesis as we did not see any noticeable correlation supporting it, although we noticed a slight tendency where the decrease of time and of number of clauses were correlated.

Through these hypotheses, we tried to understand the causes of these enhancements. None of these hypothesis proved to be useful.

Conclusion

We have proposed two new QBF Compact Tree Encodings: CTE-OPEN based on causal links (plan-space) and CTE-EFA based on explanatory frame axioms (state-space). We compared these encodings with the state-of-the-art QBF encoding CTE-NOOP. In average over all available STRIPS IPC benchmarks, CTE-EFA performed twice as fast as CTE-NOOP (respectively 1.7 times faster for CTE-OPEN).

Through experiments, we refuted the two hypotheses we had formulated in order to explain the causes of this enhancement: neither the difference in the number of literals and clauses nor the transitions-to-nodes ratio between the three encodings allow us to draw conclusions.

Although it is fair to say that the work we are presenting lacks explanations on the reasons for the gain in performance, we think that this paper aims at showing the interest of systematically studying the statistical properties of the various action representations in encodings in order to understand the ontological choices related to these action representations.

Furthermore, it is noticeable that the performance ranking of the various action representations of SAT encodings (e.g., No-op performs better than EFA) is different in QBF (as we showed, EFA performs better than No-op in the CTE encoding). It would be interesting to study more broadly the methods used in SAT for encoding actions and see how their QBF counterpart behave.

[8]Context and expansion are defined in (Cashmore, Fox, and Giunchiglia 2012). Intuitively, the expansion is a tree representing the QBF and a context is a leaf in that tree.

References

Blum, A., and Furst, M. 1997. Fast planning through planning-graphs analysis. *Artificial Intelligence* 90(1-2):281–300.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artif. Intell.* 69(1-2):165–204.

Cashmore, M.; Fox, M.; and Giunchiglia, E. 2012. Planning as quantified boolean formula. In Raedt, L. D.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 217–222. IOS Press.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50(5):752–794.

Comte, A.; Gasquet, O.; Heba, A.; Lezaud, O.; Maris, F.; Skander-Ben-Slimane, K.; and Valais, M. 2015. Twist your logic with touist. *CoRR* abs/1507.03663.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artif. Intell.* 132(2):151–182.

Ernst, M.; Millstein, T.; and Weld, D. 1997. Automatic SAT-compilation of planning problems. In *Proc. IJCAI-97*.

Janota, M., and Marques-Silva, J. 2015. Solving QBF by clause selection. In Yang, Q., and Wooldridge, M., eds., *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 325–331. AAAI Press.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2012. Solving QBF with counterexample guided refinement. In Cimatti, A., and Sebastiani, R., eds., *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, 114–128. Springer.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2016. Solving QBF with counterexample guided refinement. *Artif. Intell.* 234:1–25.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *ECAI-92*, 359–363.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. AAAI-96*, 1194–1201.

Kautz, H., and Selman, B. 1998. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Proceedings of AIPS-98 Workshop on Planning as Combinatorial Search*.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In *Proc. IJCAI-99*, 318–325.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. Satplan04: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition, IPC-06*.

Kautz, H. 2004. Satplan04: Planning as satisfiability. In *Abstracts of the 4th International Planning Competition, IPC-04*.

Lonsing, F., and Egly, U. 2017. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In de Moura, L., ed., *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, 371–384. Springer.

Mali, A., and Kambhampati, S. 1998. Refinement-based planning as satisfiability. In *Proc. Workshop planning as combinatorial search, AIPS-98*.

Mali, A., and Kambhampati, S. 1999. On the utility of plan-space (causal) encodings. In *Proc. AAAI-99*, 557–563.

Maris, F., and Régnier, P. 2008. TLP-GP: new results on temporally-expressive planning benchmarks. In *(ICTAI 2008), Vol. 1*, 507–514.

Rabe, M. N., and Tentrup, L. 2015. CAQE: A certifying QBF solver. In Kaivola, R., and Wahl, T., eds., *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, 136–143. IEEE.

Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds. 2008. *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*. AAAI.

Rintanen, J.; Heljanko, K.; and Niemelä. 2004. Parallel encodings of classical planning as satisfiability. In *Proc. European Conference on Logics in Artificial Intelligence, JELIA-04*.

Rintanen, J.; Heljanko, K.; and Niemelä. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(1213):1031–1080.

Rintanen, J. 2001. Partial implicit unfolding in the davis-putnam procedure for quantified boolean formulae. In *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings*, 362–376.

Rintanen, J. 2003. Symmetry reduction for sat representations of transition systems. In *Proc. ICAPS-03*.

Rintanen, J. 2007. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, 1045–1050.

Rintanen, J. 2015. Discretization of temporal models with application to planning with SMT. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 3349–3355.

Shin, J., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artif. Intell.* 166(1-2):194–253.

Tange, O. 2011. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine* 36(1):42–47.

Wolfman, S. A., and Weld, D. S. 1999. The LPSAT engine & its application to resource planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, 310–317.

Characterizing and Computing All Delete-Relaxed Dead-ends

Christian Muise
IBM Research
christian.muise@ibm.com

Abstract

Dead-end detection is a key challenge in automated planning, and it is rapidly growing in popularity. Effective dead-end detection techniques can have a large impact on the strength of a planner, and so the effective computation of dead-ends is central to many planning approaches. One of the better understood techniques for detecting dead-ends is to focus on the delete relaxation of a planning problem, where dead-end detection is a polynomial-time operation. In this work, we provide a logical characterization for not just a single dead-end, but for *every* delete-relaxed dead-end in a planning problem. With a logical representation in hand, one could compile the representation into a form amenable to effective reasoning. We lay the ground-work for this larger vision and provide a preliminary evaluation to this end.

1 Introduction

Learning conflicts and reasoning about them has long been recognized as an important component in many fields of artificial intelligence and optimization. Recently, a type of conflict found in planning that represents a state where the goal cannot be achieved, referred to as a *dead-end*, has received increased attention in the field. This interest culminated in the first ever International Planning Contest on Unsolvability in 2016 (Muise and Lipovetzky 2016), which required the planners to detect whether or not the initial state of a planning problem was a dead-end. Effective techniques for dead-end detection are useful for determining problem unsolvability, as evidenced by the winning planners in the contest, as well as solving classical planning problems (Hoffmann, Kissmann, and Torralba 2014; Lipovetzky, Muise, and Geffner 2016), non-deterministic planning problems (Muise, McIlraith, and Beck 2012), and probabilistic planning problems (Kolobov, Mausam, and Weld 2012; Camacho, Muise, and McIlraith 2016).

Perhaps one of the most well understood dead-end detection techniques is to search for a solution in the delete relaxation of a problem (i.e., assuming that actions in our model never delete what is true). Because solvability of the delete relaxation is a polynomial time operation, we can determine if a state is a dead-end in the delete relaxation in polynomial time as well. If this is the case, we can further conclude that the original state is a dead-end; if no plan exists in the delete relaxation, then no plan exists in the original problem.

In this work, we focus on characterizing what it means for a state to be a delete-relaxed dead-end, and we do so by introducing a novel logical encoding of the problem. From the encoding we can compile the theory into an equivalent representation that compactly represents all delete-relaxed dead-ends of a particular minimal form, while at the same time allowing efficient queries or modifications over the theory. For example, one could quantify the likelihood that a partially observable state is in a dead-end, or manipulate the representation to compute all states that could lead to a delete-relaxed dead-end through the application of a particular action by using standard logical operations. The logical representation provides interesting insights into the structure of the delete-relaxed dead-end detection problem in relation to other planning encodings, and we elaborate on these connections throughout the paper.

Our contributions are 3-fold: (1) we provide a simple logical encoding to represent the set of delete-relaxed dead-end states in a STRIPS planning problem; (2) we present an extension that compiles away some of the naïve encoding to give us an equivalent encoding that is easier to reason with; and (3) we present a preliminary evaluation on a range of domains and knowledge compilers.

2 Preliminaries

Delete-Relaxed STRIPS Planning In this work, we assume a STRIPS model of planning (Ghallab, Nau, and Traverso 2004). A problem is defined as a tuple, $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$, where $\mathcal{F}$ is the set of fluents in the problem, $\mathcal{I} \subseteq \mathcal{F}$ is the initial state (we assume all fluents not in $\mathcal{I}$ are false initially), $\mathcal{A}$ is the set of actions in the problem and $\mathcal{G} \subseteq \mathcal{F}$ is the goal that the planner must achieve. Every action $a \in \mathcal{A}$ is described by its precondition $\mathrm{PRE}(a) \subseteq \mathcal{F}$ and add effects $\mathrm{ADD}(a) \subseteq \mathcal{F}$. Traditionally, there is also a set of delete effects, $\mathrm{DEL}(a) \subseteq \mathcal{F}$, but for this work we are only concerned with delete-relaxed planning: i.e., $\forall a \in \mathcal{A}, \mathrm{DEL}(a) = \emptyset$. A complete state $s \subseteq \mathcal{F}$ describes what is true in the world, while every fluent in $\mathcal{F} \setminus s$ is presumed to be false. An action a is applicable in state s if and only if $\mathrm{PRE}(a) \subseteq s$, and the result of applying a in s is defined to be $s \cup \mathrm{ADD}(a)$. Note that because there are no delete effects, applying an action in a state can only add fluents to the state: i.e., make more of the fluents true. Further, because all of the preconditions are fluents as well, when an action is applicable it will remain applicable. We will use the following to refer to the actions that add a particular fluent:

$$adders(f) = \{a \mid a \in \mathcal{A} \text{ and } f \in \mathrm{ADD}(a)\}$$

A planning problem, $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$, is *solvable* if a sequence of action applications transforms the state $\mathcal{I}$ into one where

all of the fluents in $\mathcal{G}$ are true, and it is *unsolvable* otherwise. The state s is a delete-relaxed dead-end for problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ if and only if $\langle \mathcal{F}, s, \mathcal{A}, \mathcal{G} \rangle$ is unsolvable.

Satisfiability We use the standard notion of Boolean logic to characterize the delete-relaxed dead-ends in a planning problem. Here we describe the basic concepts and notation, but the reader is referred to (Biere et al. 2009) for further details. The task of Satisfiability (or simply SAT) is to find a satisfying assignment to a set of Boolean variables given a logical formula. The standard representation for a logical formula, and the one we adopt, is Conjunctive Normal Form (CNF). A CNF formula is a conjunction of clauses, and each clause is a disjunction of literals; either a Boolean variable or its negation. For example, the following CNF formula,

$$(x \vee \neg y) \wedge (\neg x \vee y)$$

has two satisfying assignments: $(x = \top, y = \top)$ and $(x = \bot, y = \bot)$. While CNF is a natural form to express many problems, reasoning with it is a difficult task. Many other forms have been proposed in the literature that are more amenable to reasoning, and *knowledge compilation* is the task of converting from one form (typically CNF) to another in order to make reasoning easier (Darwiche and Marquis 2002). We forgo describing the alternative forms here, but provide some empirical results in Section 4 on the difficulty of knowledge compilation from our proposed CNF encoding to a variety of target forms.

The final notion we will use is *projection*. The projection of a satisfying assignment onto a subset of the variables is simply the portion of the assignment that involves variables in the subset. We use the notion of projection to focus on only one aspect of the model, and it may be the case that many full assignments will map to the same projected assignment.

3 Encoding All Delete-Relaxed Dead-ends

To reason about all of the delete-relaxed dead-ends for a planning problem, we must construct a CNF where satisfying assignments correspond to delete-relaxed dead-ends. However, we do not want just any representation of delete-relaxed dead-ends, but instead one that captures the core reason that a state cannot achieve the goal. This can be useful for interpretability of deadends, but additionally (as we see later) allows for a far more compact encoding than the standard Planning-as-SAT encodings. To that end, we will only model delete-relaxed dead-ends that are in a particular fixed-point of delete-relaxed reachability.

Definition 1 (Fixed-Point State) *We say that a state s in a delete-relaxed planning problem $\langle \mathcal{F}, \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ is a fixed-point state iff $\forall a \in \mathcal{A}, \mathrm{PRE}(a) \not\subseteq s$ or $s \cup \mathrm{ADD}(a) = s$.*

Intuitively, a state is a fixed-point state whenever applying additional actions will have no effect on the state (either an action is not applicable, or adds no new fluents to the state). Every delete-relaxed dead-end will have a corresponding fixed-point state s where $\mathcal{G} \not\subseteq s$, and it is the set of delete-relaxed dead-end fixed-point states that we model with our SAT encoding.

Not only are fixed-point delete-relaxed dead-end states more easy to work with when modeling, but they also represent the relevant core of what is causing a state to be a dead-end. For any state s, its corresponding fixed-point state s' will include everything achievable from s (and thus $s \subseteq s'$). With our encoding, we capture what does *not* hold in the fixed-point state (i.e., $\mathcal{F} \setminus s'$), and this will always be more general (i.e., fewer fluents) than what does not hold in s.

First, we present a simple but naïve encoding that achieves our objective of modeling the fluents not achievable in a fixed-point delete-relaxed dead-end. Then we prove the correctness of the encoding, and show an alternative encoding that captures the same set of solutions using fewer variables. The alternative encoding provides a better representation for knowledge compilers to work with due to the reduction in variables. We conclude with a discussion of some of the interesting properties that our encodings have, and their relation to existing notions in the literature.

3.1 Naive Encoding

The key insight that we use for our encoding is to focus on what *cannot* be achieved, rather than what *can* be achieved. This is in stark contrast with the vast majority of existing SAT encodings for planning-related problems. We should note that our aim is to model the space of all states that are a delete-relaxed dead-end for a particular problem, and not just identify if the initial state is a delete-relaxed dead-end.

Because we are interested in the states from which the goal cannot be achieved, our encoding focuses on those aspects of the problem that are *unachievable*. A fluent f is unachievable from state s whenever the problem $\langle \mathcal{F}, s, \mathcal{A}, \{f\} \rangle$ is unsolvable. Similarly, an action a is unachievable from state s whenever the problem $\langle \mathcal{F}, s, \mathcal{A}, \mathrm{PRE}(a) \rangle$ is unsolvable.

Viewing the task of representing all delete-relaxed dead-ends in terms of what cannot be achieved leads us to a simple CNF encoding where the variables are defined as:

- $x_{\bar{f}}$: For every fluent $f \in \mathcal{F}$, $x_{\bar{f}}$ represents the fact that f is unachievable.
- $x_{\bar{a}}$: For every action $a \in \mathcal{A}$, $x_{\bar{a}}$ represents the fact that a is unachievable.

The clauses that we use to characterize all fixed-point delete-relaxed dead-ends are as follows:

$$\bigvee_{f \in \mathcal{G}} x_{\bar{f}} \tag{1}$$

$$x_{\bar{a}} \rightarrow \bigvee_{f \in \mathrm{PRE}(a)} x_{\bar{f}} \qquad \forall a \in \mathcal{A} \tag{2}$$

$$x_{\bar{f}} \rightarrow x_{\bar{a}} \qquad \forall f \in \mathcal{F}, \; \forall a \in adders(f) \tag{3}$$

The intuition behind each of the clause types is as follows: (1) some aspect of the goal must be unachievable; (2) if an action is unachievable, then some aspect of its precondition

must be unachievable; and (3) if a fluent is unachievable, then every action that could add it must be unachievable.

The encoding is relatively small, and dominated by clauses of type 2 and 3. For type 2, each clause is only as large as the precondition of the action in question, and there are only $|\mathcal{A}|$ clauses of this type. For type 3, each clause is binary, and there will be $\sum_{f \in \mathcal{F}} |adders(f)|$ such clauses (a small number in typical planning problems). We will use $CNF(P)$ to refer to the encoding of planning problem P.

There are two special cases worth noting. First, if a fluent f has no action that can add it (i.e., $adders(f) = \emptyset$), then there will be no clauses of type 3: if f is false in the state of the world, it will remain unachievable. Second, if an action has no precondition then it is trivially *not* unachievable (i.e., it can always be executed, and its effects achieved).

A satisfying assignment will stipulate which actions and fluents are deemed unachievable. The *corresponding state* of a satisfying assignment is the state s defined from the fluent variable as $\{f \mid x_{\bar{f}} = \bot\}$. We can now establish the theoretical connection between the set of fixed-point delete-relaxed dead-ends for a problem P and $CNF(P)$:

Theorem 2 *The set of satisfying assignments for $CNF(P)$, projected to the fluent variables, corresponds one-to-one with the set of fixed-point delete-relaxed dead-ends of P.*

Proof Sketch. We first establish that the projection of any satisfying assignment onto the fluent variables corresponds to a fixed-point delete-relaxed dead-end. Formulae 2 and 3 together ensure that the assignment is a fixed-point state: if it was not, then some action a must be applicable in the corresponding state with $f \in \mathrm{ADD}(a)$ and f marked as being unachievable: i.e., $x_{\bar{a}}$ and $x_{\bar{f}}$ hold in the assignment while a is applicable in the corresponding state. Note, however, that this leads to a contradiction: the contrapositive of formula 2 stipulates that if every precondition fluent of a is not unachievable, then a is not unachievable as well. Finally, the assignment must correspond to a delete-relaxed dead-end, as it is a fixed-point state and some aspect of the goal is unachievable (following formula 1).

For the other direction, consider a candidate delete-relaxed dead-end fixed-point state s. We construct a corresponding satisfying assignment by setting the following variables to true and all others false:

$$\{x_{\bar{f}} \mid f \notin s\} \cup \{x_{\bar{a}} \mid \mathrm{PRE}(a) \not\subseteq s\}$$

Formulae 2 and 3 naturally follow from the properties of s being a fixed-point state, and 1 holds from the fact that s cannot contain the complete goal (as it is a dead-end). □

3.2 Fluent-based Encoding

While the above encoding is simple and intuitive, it unnecessarily contains information in the form of variables for action unachievement. Here, we derive an alternative encoding that removes the variables corresponding to the actions. We do so by using the transitive property of the implications in formulae 3 and 2 above; by combining them into a single formula, we obtain the following:

$$x_{\bar{f}} \rightarrow \bigvee_{f' \in \mathrm{PRE}(a)} x_{\bar{f}'} \qquad \forall f \in \mathcal{F}, \quad \forall a \in adders(f) \qquad (4)$$

Formulae 1 and 4 combined capture the precise set of delete-relaxed dead-ends for a problem. The number of variables is reduced to just the number of fluents in the problem, but the number of clauses is increased as a result: one clause of size $|\mathcal{G}|$ for the goal, and another $\sum_{f \in \mathcal{F}} |adders(f)|$ clauses of size $|\mathrm{PRE}(a)|$. Just as $adders(f)$ is typically small, so is the size of $\mathrm{PRE}(a)$. We found this increase to be negligible. Because the correctness of the fluent-based encoding follows directly from the logical combination of formulae 2 and 3 to produce formula 4, we forgo a formal proof here.

3.3 Discussion

A number of SAT encodings have been proposed for modeling planning problems, but to the best of our knowledge they all focus on modeling the existence of plans in some form. The idea of planning-as-SAT was pioneered in the early 1990's as a method for solving planning problems (Kautz and Selman 1992). Recent advances have drastically improved the performance of planning-as-SAT (Rintanen 2012), focused on optimal planning (Robinson et al. 2010), computation of heuristics (Bonet and Geffner 2006), planning with multi-valued variable representations (Huang, Chen, and Zhang 2012), and computing maximally flexible plans (Muise, Beck, and McIlraith 2016). All of these approaches, however, share a common thread: a satisfying assignment corresponds to a plan.

In contrast, our encodings capture (fixed-point) states of the delete-relaxed problem where no plan exists. Many planning-as-SAT encodings use a layered approach and must fix the number of actions in a plan without knowing in advance what depth would suffice. For those that do not use a layered approach (e.g., (Robinson et al. 2010; Muise, Beck, and McIlraith 2016)), a common bottleneck is the number of clauses required for preventing causal self support: e.g., action a_1 adds f_1, which enables a_2, which adds f_2, which enables a_1, etc. To rule out any such causal loops in a plan, a cubic number of clauses are required to model the transitive closure of "support". Our encodings do not need to pay this high cost in encoding complexity. Intuitively, this is because the fundamental property we are modeling (i.e., if a fluent is unachievable) is universally quantified: a fluent is unachievable if and only if *every* action that adds it is unachievable. This is in contrast with the fundamental *existential* property that non-layered planning-as-SAT encodings aim to capture: a fluent is achievable when *at least one* action (or the initial state) is able to achieve it. [1]

The final connection of interest is prime implicants and minimal delete-relaxed dead-ends. Prime implicants describe only those variable settings required for a Boolean assignment to be satisfying, and minimal dead-ends describe only those unachievable fluents required to be a dead-end, so there is reason to believe they would coincide. However, note that relaxing one Boolean variable in the encodings above amounts to saying that we have a fixed-point delete-relaxed dead-end *regardless* of whether or not a selected fluent or action is unachievable. Changing the setting to just

[1] The ability for the initial state to support a fluent is typically captured through an introduced "initial state action".

Domain	bddmin		c2d		cnf2bdd		DSHARP		minic2d		sharpSAT	
	act	flu	act	flu	act	flu	act	flu	act	flu	act	flu
airport (50)	0	2	7	**12**	2	6	3	4	7	10	4	5
floortile (20)	0	0	10	10	0	7	2	2	4	**14**	5	6
mystery (30)	0	4	7	8	2	6	6	6	6	6	7	**9**
parcprinter (20)	0	0	14	**16**	0	0	0	0	7	8	0	0
pegsol (20)	0	0	**20**	**20**	0	17	0	**20**	0	14	**20**	**20**
sokoban (20)	0	0	9	**10**	0	0	0	0	2	5	3	3
trucks (30)	0	1	10	**11**	0	6	1	3	5	5	2	4
woodworking (20)	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
ALL (210)	0	8	78	**88**	5	43	13	36	32	63	42	48

Table 1: # of Problems Compiled. (Brackets): all encoded problems. **Bold**: greatest number of problems compiled.

one Boolean variable can easily cause the assignment to no longer be satisfying, and as a result the prime implicants do not correspond directly to the minimal delete-relaxed dead-ends. The final note of interest is with respect to minimal delete-relaxed dead-ends: i.e., a delete-relaxed dead-end with the fewest number of unachievable fluents specified. Every minimal delete-relaxed dead-end will correspond directly to some satisfying assignment of the proposed encoding. This follows from the fact that the set of satisfying assignments correspond to every fixed-point delete-relaxed dead-end, which necessarily includes the minimal delete-relaxed dead-ends, and it also underscores the subtle nature of the $x_{\bar{f}}$ variables: when set to false, many of the constraints become satisfied due to the structure of (3) or (4).

4 Evaluation

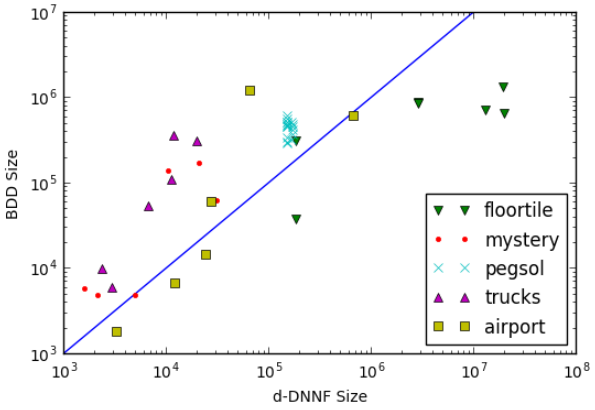


Figure 1: Size of the d-DNNF generated by c2d compared with the size of the BDD generated with cnf2bdd.

There are many possibilities for using the logical characterization of delete-relaxed dead-ends that we presented in Section 3. For our preliminary investigation, we evaluate the potential of compiling the theory into many of the popular target languages for knowledge compilation: (1) Deterministic Decomposable Negation Normal Form or d-DNNF (Darwiche and Marquis 2002); (2) Sentential Decision Diagrams or SDD (Oztok and Darwiche 2015); and (3) Binary Decision Diagrams or BDD (Biere et al. 2009). We evaluate both encodings using the range of compilers that are available. For d-DNNF this includes DSHARP (Muise et al. 2012), c2d (Darwiche 2004), and sharpSAT (Thurley 2006). While sharpSAT is not a compiler, the DSHARP compiler is built on top of the sharpSAT code-base and so it provides a natural bound on performance for DSHARP. For SDD, we use the canonical compiler miniC2D (Oztok and Darwiche 2015). Finally, for BDD, we use both BDD-MiniSAT (Toda and Soh 2015) and the CUDD software package (Somenzi 2015); referred to here as cnf2bdd.

We implemented software to convert STRIPS planning problems (specified in PDDL) into CNF corresponding to the two encodings (the converter, benchmarks, and data will be released along with publication). All experiments were conducted on a 3.4Ghz Linux Desktop, and every trial was limited to 30min and 4Gb of time and memory. We used a selection of common benchmarks known to have dead-ends. In Table 1 we show the coverage for each of the tested compilers (in their best configuration) on each of the encodings. While the comparison across compilers is not direct, as they compile the encoding into different target languages, we can make some general observations.

The first aspect to note is that the fluent-based encoding strictly dominates the action-based one. Despite the increase in number of clauses, the search space becomes much more manageable. Investigating the data further, we found that roughly two thirds of all failures were due to memory violations, which indicates that storing the compiled form is the bottleneck in the action-based encoding.

The next key insight is that d-DNNF (via c2d) is the target language most easily compiled. This is a direct result of the more compact form the target language yields: Fig-

ure 1 shows the size comparison for the fluent-based problems both c2d and cnf2bdd could compile. Generally, the d-DNNF is far more compact with the notable exception of the floortile domain. While efficient reasoning can be done using d-DNNF, common symbolic planning operations such as progression or regression remain difficult. BDD's, on the other hand, are the target language of choice for many symbolic planning approaches, and so it is of particular interest to focus on cnf2bdd. We observed the biggest jump in coverage between encodings for a compiler occurred with cnf2bdd, indicating that there is far more structure in the fluent-based encoding that the BDD can capture succinctly.

While only preliminary, the results paint a broad picture of the capabilities existing compilers have for dealing with the characterization of all delete-relaxed dead-ends in a domain. There remains plenty of room for improvement for the key compilers such as cnf2bdd in the form of planning-specific variable ordering (Kissmann and Hoffmann 2014) and approximate compilation (Soeken et al. 2016).

5 Summary

Dead-end detection for classical planning is central to many state-of-the-art planners, and also a key component for approaches that solve more expressive planning formalisms. In this work, we take the first steps towards characterizing the space of all delete-relaxed dead-ends. We achieve this using a pair of Boolean logic encodings that are elegant in their simplicity, and remarkably, are not susceptible to the same level of complexity that similar related planning encodings face. We also performed a preliminary evaluation to test the range of knowledge compilation techniques that can process the Boolean encodings, and demonstrated the strengths and weaknesses that they have on a suite of benchmarks known to have dead-ends. Our work lays the foundation for incorporating a compact representation of delete-relaxed dead-ends in a larger system, and moving forward we hope to (1) leverage our encodings to improve planner performance; and (2) incorporate notions such as the Π-compilation for stronger dead-end detection by combining fluents (Keyder, Hoffmann, and Haslum 2012).

References

Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T. 2009. *Handbook of satisfiability, frontiers in artificial intelligence and applications*. IOS Press.

Bonet, B., and Geffner, H. 2006. Heuristics for planning with penalties and rewards using compiled knowledge. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, 452–462.

Camacho, A.; Muise, C.; and McIlraith, S. A. 2016. From fond to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *The 26th International Conference on Automated Planning and Scheduling*.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

Darwiche, A. 2004. New advances in compiling CNF to decomposable negation normal form. In *Proceedings of European Conference on Artificial Intelligence*, 328–332.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who Cares? Tailoring Merge-and-Shrink Heuristics to Detect Unsolvability. In *21st European Conference on Artificial Intelligence (ECAI 2014)*, 441–446.

Huang, R.; Chen, Y.; and Zhang, W. 2012. SAS+ planning as satisfiability. *Journal of Artificial Intelligence Research (JAIR)* 43:293–328.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *European Conference on Artificial Intelligence*, 359–363.

Keyder, E. R.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*.

Kissmann, P., and Hoffmann, J. 2014. Bdd ordering heuristics for classical planning. *Journal of Artificial Intelligence Research* 51:779–804.

Kolobov, A.; Mausam; and Weld, D. S. 2012. A theory of goal-oriented mdps with dead ends. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, 438–447.

Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In *The 26th International Conference on Automated Planning and Scheduling*.

Muise, C., and Lipovetzky, N. 2016. Unsolvability international planning competition. `http://unsolve-ipc.eng.unimelb.edu.au`. Accessed: 2018-03-19.

Muise, C.; Beck, J. C.; and McIlraith, S. A. 2016. Optimal partial-order plan relaxation via maxsat. *Journal of Artificial Intelligence Research*.

Muise, C.; Mcilraith, S. A.; Beck, J. C.; and Hsu, E. 2012. DSHARP: Fast d-DNNF Compilation with sharpSAT. In *Canadian Conference on Artificial Intelligence*.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Oztok, U., and Darwiche, A. 2015. A top-down compiler for sentential decision diagrams. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 3141–3148.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2010. Partial weighted maxsat for optimal planning. In *11th Pacific Rim International Conference on Artificial Intelligence*, 231–243.

Soeken, M.; Gro, D.; Chandrasekharan, A.; Drechsler, R.; et al. 2016. Bdd minimization for approximate computing. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 474–479. IEEE.

Somenzi, F. 2015. Cudd: Cu decision diagram package release. `https://github.com/ivmai/cudd`. Accessed: 2018-03-19.

Thurley, M. 2006. sharpSAT — counting models with advanced component caching and implicit BCP. In *Ninth International Conference on Theory and Applications of Satisfiability*, 424–429.

Toda, T., and Soh, T. 2015. Implementing efficient all solutions SAT solvers. *CoRR* abs/1510.00523.

Integrating Meeting and Individual Events Scheduling

Anastasios Alexiadis and **Ioannis Refanidis** and **Ilias Sakellariou**
Department of Applied Informatics, University of Macedonia,
Egnatia 156, 54006, Thessaloniki, Greece.
talex@uom.edu.gr, yrefanid@uom.edu.gr, iliass@uom.edu.gr

Abstract

Automated meeting scheduling is the task of reaching an agreement on a time slot to schedule a new meeting, taking into account the participants' preferences over various aspects of the problem. Such a negotiation is commonly performed in a non-automated manner, that is, the users decide whether they can reschedule existing individual activities and, in some cases, already scheduled meetings in order to accommodate the new meeting request in a particular time slot, by inspecting their schedules. In this work, we take advantage of SELFPLANNER, an automated system that employs greedy stochastic optimization algorithms to schedule individual activities under a rich model of preferences and constraints, and we extend that work to accommodate meetings. For each new meeting request, participants decide whether they can accommodate the meeting in a particular time slot by employing SELFPLANNER's underlying algorithms to automatically reschedule existing individual activities. Time slots are prioritized in terms of the number of users that need to reschedule existing activities. An agreement is reached as soon as all agents can schedule the meeting at a particular time slot, without anyone of them experiencing an overall utility loss, that is, taking into account also the utility gain from the meeting. This dynamic multi-agent meeting scheduling approach has been tested on a variety of test problems with very promising results.

Introduction

Three friends, Alice, Bob and Charlie, want to meet for a coffee. Alice sends a request and proposes three alternative coffee shops, in different areas of the city, as well as a time frame. The duration of the coffee meeting is estimated to one hour.

Each friend has personal preferences for a variety of aspects of the proposed coffee meeting, such as the particular coffee shop, traveling arrangements needed to go there, as well as the alternative time slots. Their agendas are very tight, so rescheduling might be necessary in order to accommodate the new meeting. While rescheduling, they might consider travel time and expenses, as well as overall utility of the alternative schedules.

Meeting scheduling is a multiobjective optimization process, where multiple agents negotiate on a common time slot to hold the meeting, taking also into account their commitments and preferences. An agent may have already scheduled individual activities, such as tasks with deadlines undertaken, family duties, etc., as well as other meetings. Some of these activities may be flexible and can be rescheduled, while others may be not. Furthermore, some activities might be of lesser importance and the user may decide to abandon them, in order to be able to participate in the meeting. The agent may also have preferences concerning the location and time of the meeting, as well preferences towards scheduling its individual activities. It might also have hard or soft constraints, e.g., ordering constraints, between his individual activities.

In this work we assume that the agents do not have preferences over how the meeting will be scheduled; they only care to schedule the meeting. Under this assumption, the simplest case is when the participating agents can find a common free time slot to schedule the meeting, without the need to resort to rescheduling. This time slot is considered optimal for the meeting. Unfortunately, this situations is rather rare. It is quite common that such a time slot does not exist. However, this does not mean that the meeting cannot be scheduled, since participating agents may negotiate to reach an agreement on a time slot that is not initially available to all agents. During the negotiation, time slots not available for all agents are considered, under the condition that they can become "free" if some agents manage to reschedule or even drop already scheduled activities.

The negotiation process can terminate without reaching an agreement, i.e. the agents may conclude that the meeting cannot be scheduled, since rescheduling or dropping out other activities reduces the overall utility received by at least one of the agents (we assume that all the agents are veto participants of the meeting, that is, without anyone of them the meeting cannot hold). As a last resort, the agents might try to reschedule other meetings, either with agents not participating in the current negotiation, or among themselves. In that case, a recursive process may commence, where in order to schedule one meeting, other meetings need to be rescheduled, that may initiate rescheduling of other meetings with a different group of agents and so on.

In this work, we consider the problem of multi-agent meeting scheduling, where each agent has a set of already scheduled individual activities, that can be rescheduled in order to accommodate the new meeting. We adopt a rich model

of individual activities, with a variety of unary and binary constraints and preferences, that has been proposed in (Refanidis and Yorke-Smith 2010). Meetings are described in a simpler way, that is, duration, temporal domain and utility per agent. We consider the problem of meeting scheduling as a multi-agent constraint satisfaction problem, based on solving many single-agent constraint optimization problems, with each agent attempting to maximize his utility, with the latter being a function of all scheduled activities (individual activities and the meeting), as well as the way they have been scheduled (unary and binary preferences). We do not consider recursive meeting rescheduling, i.e. only individual agent activities can be rescheduled during the process.

The novelty lies in meeting scheduling empowered by rescheduling of individual tasks supporting such a rich model. To the best of our knowledge, there is no other work that combines these two aspects of personal time management.

A powerful scheduler based on greedy construction and stochastic optimization (Alexiadis and Refanidis 2016b) is employed by each agent for constructing a schedule for his individual activities. To schedule a meeting among a set of agents, we have designed and implemented a meta-scheduler, that negotiates with each agent a time slot and a location that can be agreed upon. Each such proposal is evaluated by agents using the scheduler mentioned above. For each proposal, the agent employs distributively the individual activities scheduler to decide whether it can accept each proposed time slot or not. To accept a time slot, the overall utility for the agent when scheduling the new meeting in this time slot should be higher than keeping his original schedule and not scheduling the meeting at all.

In this work, we do not consider strategic behaviour on behalf of the agents. We have tested our approach on a variety of problem instances, with very promising results in terms of performance and scalability.

The rest of the paper is structured as follows: Section 2 presents related work. Section 3 presents background information concerning scheduling individual activities. Section 4 defines the integrated scheduling problem and presents the proposed approach. Section 5 presents experimental results and, finally, Section 6 concludes the paper and poses future directions.

Related Work

Not surprisingly, there exist numerous approaches to agent based meeting scheduling, since it has attracted research interest rather early. In fact, it was considered to be one of the main applications of intelligent personal assistants, a class of agent systems aiming to support the user in performing tedious everyday tasks. As expected, all approaches follow a similar pattern of interaction: there is a group of agents, each one representing a participant, and possibly a host/meeting agent which coordinates a negotiation process with proposals and counter-proposals on meeting parameters. The agreement is, in most cases, a time slot that maximizes a collective preference value, computed by individual preference values of participating agents. There are quite a few issues in the above setting, such as the number of proposals in each round, the modelling of constraints and preferences, privacy issues, interoperability issues (semantic web), user modelling via learning, "bumping" strategies, etc. In this section we will present in more detail some of the approaches.

One of the earliest agent-based meeting scheduling applications is reported in (Jennings and Jackson 1995), according to which each participant is "represented" by a meeting scheduling agent (MSA), managing its users calendar. The procedure followed, presents quite a few similarities with the well known Contract Net protocol: An MSA acting on behalf of its meeting host, announces its intention to arrange a meeting, along with respective time constraints and duration. Participants (their MSAs) respond with bids, that are possible time slots annotated with a preference value, which are used by the meeting host to discover the best common time slot with respect to its global preference value. This announce-bid cycle continues until either a suitable slot is found, or scheduling the meeting is determined to be non-possible, in which case, the initial announcement (meeting duration / time constraints) are changed and the process starts over again.

Sen and Durfee (Sen and Durfee 1996) address the problem in a similar setting, however agents report their availability (true/false) on a meeting announcement over n possible slots, along with m alternative slots. In the same work, authors report on rescheduling meeting strategy (bumping) when conflict occurs, maximizing a utility function.

The RETSINA (Reusable Environment for Task-Structured Intelligent Networked Agents) Calendar Agent (RCAL) (Payne, Singh, and Sycara 2002), was aiming to bridge information available on the Semantic Web with the users personal information managers, e.g., Outlook 2000. RCAL uses Contract Net to negotiate a meeting between participants, a process that involves receiving bids from involved parties to determine an appropriate meeting slot. The advantage of RCAL is that information regarding the users schedule is obtained automatically through semantically annotated descriptions, and thus allows for a more "accurate" scheduling of meetings.

(Chun, Wai, and Wong 2003) treats the problem of meeting scheduling from the perspective of user privacy, that is, managing to optimally schedule a meeting without complete knowledge of individual participant preferences. The negotiation takes place between user Secretary Agents (SA), and a Meeting Agent (MA), that coordinates process. Proposals and counter proposals are annotated with a participants preference value w.r.t. the meeting parameters. In each step the MA collects the set of proposals and generates a new set, sorted on global preference estimation values, that is, values computed from the annotated replies of the SA agents.

In DAS (Wang 2003) lightweight agents called coordination agents (CA), are created by the participants for each proposed time slot of the meeting. Agents managing the same slot are combined to a single agent. The approach aims at reducing communication costs, by decoupling user agents from CA, and allowing interactions between the latter on the same computational host. (Franzin et al. 2002) [Franzin et al 2002, 2004] provide a more rigorous constraint modelling of the problem considering hard and soft constraints, with the latter representing user preferences with respect to meeting parameters. The term "common assignment problem with

preferences" is introduced to describe the collaboratively scheduling of a meeting, with agents having common variables to set, but possibly different "internal" constraints on these variables. Meeting scheduling proceeds with rounds of proposal-counter proposal, guided by the preference value, in order to reach optimal solutions. In a similar vein, MSRAC (meeting scheduling with reinforcement of arc consistency) (BenHassine and Ho 2007), handles incremental scheduling, user preferences in the form of soft constraints, user availability by a set of hard constraints and common meeting times with other agents by equality constraints of the meeting time slot, but consider more than one meeting to be scheduled at each time point, i.e. a dynamic iterative value CSP problem. The solving process includes time slot proposals broadcasted by the host to participants, who return their available slots ranked by their preference. Bumping also considered, based on three different heuristic strategies.

CMRadar (Modi et al. 2005) offers a complete approach to calendar management, including multi-agent meeting scheduling capabilities. If a request for a meeting cannot be accommodated within the current calendar of the agent, then complex strategies involving altering other meetings of lower priority value are employed, thus engaging the agent in a multi-negotiation process, referred to as the "bumping" problem in (Modi and Veloso 2005). In that work, multi-agent meeting scheduling is modelled as a partial incremental Multi-agent Agreement problem (piMAP), where an iterative agreement protocol is employed. The main difference between earlier approaches is the use of bumping heuristic strategies, i.e. strategies to decide whether to modify an existing agent schedule to accommodate the meeting, that allow incremental scheduling. groupTime (Brzozowski et al. 2006) is a Web based application that bases meeting scheduling on user preferences, in a semi-automatic fashion, i.e. users have the chance to object to an initial meeting time, set by the system. For each possible time slot the user can set its preference explicitly, or the system assigns a value based on the current users schedule (events). Preference assigned by the system involves extracting schedule-agnostic features for each participants schedule, based on a group-specific ontology and a set of weights that are defined by machine learning techniques using as training data the users events and preferences. The set of preferences is then used to determine the meeting time.

The PTIME system (Berry et al. 2007; 2009) is a calendaring assistant that offers meeting scheduling among a variety of time management functionalities. The aim in PTIME was the system to learn user preferences, using machine learning techniques, providing an adaptive personal assistant. The learner module interacts both with the user and the constraint reasoner in order for the system to provide meeting options on a constantly evolving user preferences model.

Chronos (Zunino and Campo 2009), is a multi-agent meeting scheduler, in which each user is represented by an Organiser Agent (OA), that learns users preferences. Each OA represents its beliefs about both its users preferences and other users he interacts with (acquaintances) using Bayesian Networks (BN). BN express the causal relationships between scheduling parameters, such as time and duration of a meeting and are used to reason about meetings, by computing the probability of an agent to accept a specific proposal. Probabilities are computed for both participants in a meeting and guide a negotiation process, involving proposals and counter proposals in a multi-agent negotiation setting.

Although many approaches consider bumping, i.e. rearranging another meeting in order to accommodate a new request, the issue of rearranging the participants private schedule according to his original constraints with a rich scheduling model, has not been investigated in depth until now.

Background: Individual activity scheduling

This section presents the model and the assumptions for scheduling individual activities, as it has been proposed in (Refanidis and Yorke-Smith 2010) and has been implemented in the SELFPLANNER system. This model has been adopted entirely in the present work and has been extended to support also meetings. So, for the completeness of the paper, it is purposeful to give a quick overview of it.

A person (equivalently an agent) may have a set T of N individual activities to accomplish. Each activity T_i ($1 \leq i \leq N$) can be accomplished only during certain time periods, which constitute its temporal domain. This domain is defined as a set of temporal intervals $D_i = [a_{i1}, b_{i1}) \cup [a_{i2}, b_{i2}) \cup \ldots \cup [a_{i,F_i}, b_{i,F_i})$. The duration of an activity may be not fixed (e.g., visiting a museum), so for each activity T_i he can specify its minimum duration d_i^{min} and its maximum duration d_i^{max}, with more scheduled duration resulting in more utility for the person. Activities can be interruptible, that is, they can be scheduled into parts (e.g., reading a book or writing a paper). For each interruptible activity the person can specify its minimum part duration $smin_i$ and its maximum part duration $smax_i$ (e.g., the person wants to devote in book reading time periods not less that 1 hour and no more than 3 hours). The sum of the durations of an activity's parts (non-interruptible activities are considered to have one part) have to be at least d_i^{min} in order for the activity to be considered scheduled.

The set of locations associated with any of the person's activities is Loc, and let M being their number. A matrix $Dist$, not necessarily symmetric, defines the temporal distance between any pair of locations (in the simple model we assume a single means of transportation). We assume that time, temporal intervals and distances are discrete, particularly integers (as the unit of time it is used the minimum temporal interval of interest, e.g., 10, 15 or 30 minutes).

A set of locations $Loc_i \in Loc$ is associated with each activity T_i. A special location called $ANYWHERE$ denotes that the activity can be executed at any location ($ANYWHERE$ is considered to have a temporal distance of 0 from and to any other location). Traveling times between locations have to be taken into account when scheduling activities in time and space; that is, any two temporally adjacent activities scheduled at different locations (not $ANYWHERE$) should have a large enough temporal gap between them, so as the person can travel between the two locations.

Activities with compatible locations may overlap in time, e.g., watching a lecture while reading emails. Each activity has a utilization value between 0% and 100%, denoting the

percentage of the user's attention that the activity requires. The sum of the utilization values of activities scheduled at the same time cannot exceed 1.

The model supports three types of binary constraints between pairs of activities:

- Proximity constraints define a minimum or/and a maximum temporal distance between two activities. For example, the minimum temporal distance between two heavy load activities should be at least 8 hours. Or, the maximum temporal distance between reading a book and writing its synopsis should be at most two days.
- Ordering constraints define an order in time. For example, preparing the slides for the lecture should precede the lecture itself.
- Implication constraints define prerequisite activities. For example, in order to go to the theater, one should first buy tickets.

Note that, especially for the proximity constraints, they can be defined also over the different parts of an interruptible activity.

The overall single-agent problem of scheduling a set of individual activities is formulated as a constraint optimization problem, with the empty plan being the least preferred solution. The objective function is additive over the various sources of utility. These include:

- Each scheduled activity – they can have different utility values.
- Any scheduled duration above the minimum duration of the activity.
- The person's preference over the temporal intervals when the activity has been scheduled (e.g., morning vs evening).
- Proximity, ordering and implication preferences, that is, soft versions of the aforementioned constraints.

The constraint optimization problem is solved by finding values for the decision variables p_i, denoting the number of parts of activity T_i ($1 \leq i \leq N$), t_{ij} (start times), d_{ij} (durations), l_{ij} (locations), for each Activity part T_{ij} ($1 \leq j \leq p_i$), while trying to maximize the sum of the utility sources. The model, with a first solver based on the Squeaky Wheel Optimization framework has been initially presented in (Refanidis and Yorke-Smith 2010). A more powerful solver, encompassing post-processing local search techniques (mainly simulated annealing) has been presented in (Alexiadis and Refanidis 2016b).

A technique that allows to produce significantly different alternative plans has been presented in (Alexiadis and Refanidis 2016a). Producing such alternative plans and retaining them in some form of cache memory may be very useful in meeting scheduling, since they can be used to immediately check whether a meeting can be accommodated in a particular time slot and what is the cost for this accommodation. Particularly, if the time slot is free at any alternative plan kept in cache memory, then the meeting can be scheduled there and the cost is equal to the utility of the current plan minus the utility of the particular alternative one.

Collaborative Meeting Scheduling

Informal Problem Specification

Building on top of the individual activities problem specification, we assume a set of K agents, each agent i of them having its own set of individual activities, according to the model described in the previous section, as well as an already accepted schedule for them S_i, with instantiated decision variables. Furthermore, each agent may already have agreed to participate in meetings with other agents (not necessarily among the K agents of interest for the new meeting), with decided time and location.

One of the K agents invites the rest of them to participate in a new meeting. This agent will act as the coordinator for the particular meeting. The coordinating agent specifies a fixed duration for the meeting, alternative locations for the meeting (or $ANYWHERE$, if no physical presence is required, e.g., teleconferences), as well as a set of temporal intervals when the meeting could be scheduled. Participating in this meeting results in some utility gain for each invited agent (including the coordinator), which may be different for each agent. The meeting will be considered successful and all participants will get the corresponding utility, only in case all of them will be able to participate in it.

Agents may reschedule already scheduled individual activities (in this work we assume that they will not reschedule already scheduled meetings, in order to accommodate the new one). Rescheduling already scheduled activities may result in a utility loss wrt their current schedule S_i. An agent will accept to participate to the meeting at a particular time and location, only in case the utility loss from rescheduling already scheduled individual activities is no more than the utility gain by participating in the meeting. In this work we do not assume extra constraints and preferences, either unary (that is, concerning the meeting by its own) or binary (that is, concerning the new meeting and already scheduled activities).

The meeting scheduling problem may have one or more solutions or even none at all. In the case multiple solutions exist, various criteria could be adopted to select the best among them, thus treating the problem as an optimization problem. One criterion, in that case, is the sum, that is, maximizing the total utility over all agents. Another criterion is the $maxmin$, that is, maximizing the minimum utility gain, where the gain for each agent i is computed as u_i minus the utility loss due to rescheduling existing activities.

However, in this work we treat the problem as a constraint satisfaction problem, that is, we try to schedule the meeting with the extra constraint that no agent receives less utility from its new schedule (with the meeting), compared to its old schedule (without the meeting). However, from the point of view of each agent individually, it is a constraint optimization problem, that is, it tries to maximize its total utility received from scheduling its individual activities, plus the utility received from the meeting.

Meetings in this work are limited in comparison to activities, that is they are specified with a temporal domain and a fixed duration only. No preferences can be defined over them.

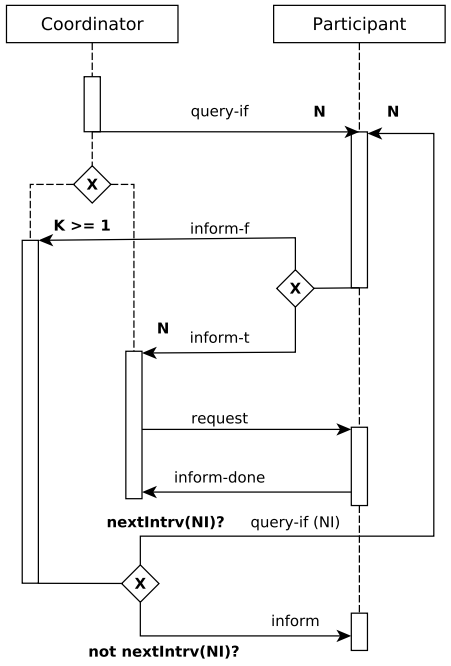


Figure 1: AUML Diagram of the agent interaction during the first phase of the Joint Activity Scheduling.

Algorithm 1 JAS-Phase-1

```
1:  procedure JAS-PHASE-1-COORD(ja,users)
2:      for i ←1 to ja.all_intervals().length() do
3:          interval ← ja.all_intervals()[i]
4:          if ¬available(interval, current_plan) then
5:              continue
6:          available ← 0
7:          for k ← 1 to users.length() do
8:              send(agent_k, query-if(interval))
9:          for k ← 1 to users.length() do
10:             if users[k].msg = inform-t then
11:                 available ← available + 1
12:             else
13:                 break
14:         if available = users.length() then
15:             return interval
16:     return failure
17:
18: procedure JAS-PHASE-1-AGENT
19:     while true do
20:         interval ← get_msg(coordinator)
21:         if available(interval, current_plan) then
22:             send(coordinator, inform-t)
23:         else
24:             send(coordinator, inform-f)
```

Algorithm

The *Joint Activity Scheduler* (JAS) presented in this Section works in two phases, that implement a message exchange mechanism between the coordinator and the invited agents. The first phase (Algorithm 1) attempts to trivially solve the problem without rescheduling already scheduled activities, by looking for a common empty interval in the current schedules of the agents (also shown in Figure 1). The second phase attempts to schedule the meeting by rescheduling already scheduled individual activities.

In the first phase, the coordinator iterates over all possible meeting time intervals, sending for each interval I_i a `query-if` message to ask participants if they are available at that time. If all agents reply affirmatively (`inform-t` message), the meeting is scheduled at I_i. If even a single agent replies negatively (`inform-f` message), then the process is repeated for the next interval in the meeting's temporal domain.

The meeting intervals I_i returned by $ja.all\_intervals()$ are subintervals of the meeting's temporal domain that have a duration equal to the meeting's length. Note that Algorithm 1 does not try to optimize any measure of utility. Since, in case of a successful result, the meeting will be scheduled without rescheduling any existing individual activity, there is no utility loss for any agent due to rescheduling. Furthermore, since the utility gain for any agent by scheduling the meeting is fixed and does not depend on when the meeting has been scheduled, the utility gain for each agent from successfully scheduling the meeting is fixed. So, from the point of view of Algorithm 1, scheduling the meeting is a constraint satisfaction problem (and not a constraint optimization one), so all solutions are equally desirable and the meeting is scheduled at the first commonly free temporal interval found.

On the other hand, if no common free interval is found, Algorithm 2 is employed, in a last attempt to find a common free interval between the agents, using rescheduling of existing individual activities. In that case it is desirable to reduce the overall need for rescheduling, since each call to the individual activity scheduler is a computationally expensive process.

In Algorithm 2 the agents receive four different types of requests from the coordinator. Specifically:

- *l*4–*l*5:[1] Request to return to the coordinator their current busy schedule, without providing information about their scheduled activities.
- *l*6–*l*12: Request to attempt to render free a specific temporal interval, by rescheduling their activities; the reply is $success$ of $failure$, accompanied with the utility gain in the first case. The utility gain is the improvement of the utility of the rescheduled plan (minus the utility of the meeting) to the utility of the old plan. Note that a $failure$ response occurs not only in cases where it was impossible to schedule the meeting in the particular time slot, but also in cases where scheduling the meeting is possible, however it results in overall utility loss (taking into account the meeting's utility). In case of a $success$ reply, the agent must retain in its local memory the corresponding schedule, till the end of the meeting scheduling procedure.

[1] Procedure JAS-PHASE-2-AGENT.

Algorithm 2 JAS-Phase-2

```
1: procedure JAS-PHASE-2-COORD(ja, users)
2:     timeline ← coord.timeline
3:     min_ints ← {−1 ...}
4:     min_mins ← {∞ ...}
5:     for k ← 1 to users.length() do
6:         users[k].timeline ← request_plan(k)
7:         timeline ← timeline + users[k].timeline
8:     for i ← 1 to ja.all_intervals().length() do
9:         int ← ja.all_intervals()[i].start
10:        temp ← timeline[int]
11:        for k ← int + 1 until int + ja.dur do
12:            temp ← temp + timeline[k]
13:        for k ← 1 to NUM_OF_TRIES do
14:            if min_mins[k] > temp then
15:                min_mins.insert(k, temp)
16:                min_ints.insert(k, int)
17:                break
18:    for i ← 1 to NUM_OF_TRIES do
19:        available ← 0
20:        for k ← 1 to users.length() do
21:            if users[k].timeline[min_ints[i]] then
22:                reschedule(k, min_ints[i], ja.dur)
23:        for k ← 1 to users.length() do
24:            if users[k].timeline[min_ints[i]] then
25:                if users[k].wait_for_repl() is yes then
26:                    available ← available + 1
27:                else
28:                    send_cancel_to_all()
29:                    break
30:            else
31:                available ← available + 1
32:        if available = users.length() then
33:            if coord.timeline[min_ints[i]] then
34:                plan ← schedule(problem + [min_ints[i], ja.dur, u_c])
35:                if [min_ints[i], ja.dur] ∈ plan then
36:                    current_plan ← plan
37:                    send_replace_to_all(min_ints[i])
38:                    return [min_ints[i], ja.dur]
39:            else
40:                send_replace_to_all(min_ints[i])
41:                return [min_ints[i], ja.dur]
42:    return −1
```

```
1: procedure JAS-PHASE-2-AGENT
2:     while true do
3:         msg ← get_msg(coordinator)
4:         if msg requests plan then
5:             send_agent(coordinator, current_plan)
6:         else if msg requests reschedule then
7:             plan[msg.start] ← schedule(problem + [msg.start, msg.dur, u_k])
8:             if [msg.start, msg.dur] ∈ plan then
9:                 u_gain = (u(plan[msg.start] − u_k) − u(plan))/u(plan)
10:                send_agent(coordinator, yes, u_gain)
11:            else
12:                send_agent(coordinator, no, 0)
13:        else if msg requests replacing then
14:            if plan[msg.start] then
15:                current_plan ← plan[msg.start]
16:        else if msg requests cancel then
17:            cancel_scheduling()
```

- *l*13–*l*15: Request to schedule the meeting at a particular time interval, following a previously $success$ reply for that interval. The agent adopts the corresponding schedule from its local memory and the meeting scheduling process terminates.
- *l*16–*l*17: Request to cancel the rescheduling process (if it is running); this request arises in case another agent has already replied a $failure$ for the time slot under consideration.

In order to minimize the average workload (by reducing the number of messages exchanged), the coordinator asks for the busy hours of all agents for the meeting's temporal interval and computes the overall workload for each possible time slot when the meeting could be scheduled (*l*5–*l*7).[2] The overall workload is stored in the $timeline$ array (the length is the maximum number of time slots of the users' plans, whereas each i-th value is the number of events scheduled in the i-th time slot). Time slots with lower overall workload (i.e., the least busy) are given priority to try to schedule the meeting. Furthermore, if a more aggressive approach is adopted, not all possible time slots need to be checked, thus sacrificing completeness in favour of reducing computational costs.

Algorithm 2 sorts the potential time slots to schedule the meeting in ascending order in terms of workload (that is, from the least busy to the busiest) (*l*8–*l*17). NUM_OF_TRIES denotes the number of temporal windows that Algorithm 2 will attempt to schedule there the meeting. Setting this constant to a very large number allows for attempting all possible time slots.

Then, for every attempted time slot the coordinator checks the busy schedule of every agent (as it was sent by the respective agent), and asks the agents that are busy to check whether rescheduling of existing activities is possible (*l*18–*l*22), thus being able to schedule the meeting at the time slot under consideration. If any of the agent replies that rescheduling

[2] Procedure JAS-PHASE-2-COORD.

failed (*l*23–*l*31), the coordinator sends a cancel message to the rest of the agents so as they terminate their rescheduling process, and proceeds to the next time slot in the list.

If all the agents reply positively to the time slot under consideration then the coordinator will check its own as well (*l*32–*l*34). If the time slot is either available or the coordinator can reschedule it without a great loss to its current plan utility then the procedure succeeds and the coordinator informs the other agents that the meeting has been scheduled at the interval under consideration (*l*35–*l*38). Otherwise the next interval in line is examined until either NUM_OF_TRIES intervals have been evaluated or there are no more time slots to examine (*l*39–*l*42).

Experimental Evaluation

We implemented a non-distributed version of the above algorithms, i.e. the code is executed as a single process, in C++.[3]

In the current experimental evaluation we are more concerned with qualitative aspects of the algorithm, and not performance issues, like execution time and number of messages exchanged, thus we do not expect the results presented to change under a distributed, multi-agent version.

In order to evaluate JAS, we created 30 participant activity schedules using the SWO+SA scheduler (Alexiadis and Refanidis 2016b). These are the predefined fully specified activity plans of the agents participating in the process. The number of activities involved in each schedule, ranges from 5 to 31, in increments of 5. For each schedule size, we selected 5 instances, taken from the literature (Alexiadis and Refanidis 2016a).

We considered four different meetings, all with a duration of four time units, but with a varying temporal domain, as shown in Table 1.

Table 1: Meeting Definitions

Meeting id	Duration	Interval 1	Interval 2
1	4	[1..20]	-
2	4	[144..151]	-
3	4	[1..15]	[16..20]
4	4	[143..147]	[148..152]

For each meeting, we created 20 random teams of the above agents (a total of 80 experiments), and attempt to schedule the meeting using our JAS implementation. For each meeting, we considered participant populations of different size, from two agents to a max of five.

We set the meeting utility u_k for all users to 4, which is a value lower than the lowest activity utility in these problem instances. It should be noted that in the scheduling problem instances activity utilities were ranging from $5 - -12$. This choice was selected so as the scheduler tries to accommodate the meeting without "dropping" an activity previously scheduled, as explained later in the section. For all experiments we have set $NUM\_OF\_TRIES = 5$.

Table 2 presents an overview of the results of the experiments. The table depicts for all teams of agents, the scheduled interval found for each meeting, the final JAS phase completed (column Ph) and the number of user schedules that required changed to accommodate the meeting (column Rs). In the Rs column we do not count every call to the scheduler but only the number of users whose plans were rescheduled at the end of the negotiation process. For the $schedule()$ function of the second phase of JAS we called the SWO+SA scheduler[4] as an external process. The scheduler was called on a modified version of the problem instance, of the user being rescheduled, that included the meeting at the interval being tested.[5] As the SWO+SA scheduler is an optimizer, that is it tries to find the plan with the maximum utility, it would not omit an activity that gives a greater utility to include an activity that gives a lower one. By providing a low u_k value to JAS, this ensures that the SWO+SA scheduler would not remove one of the already scheduled activities to include the meeting. However, the SWO+SA scheduler could decrease the duration of a scheduled activity T_i though, if the condition $d_i^{min} < d_i^{max}$ was consistent and thus decrease the utility of the agent.

JAS managed to find a common interval in 73 out of the 80 runs of the algorithm. Since a small value was set to the NUM_OF_TRIES parameter, it is possible that a common interval could be found in the remaining 7 unsuccessful cases with more rescheduling tries. Although setting this parameter to higher values would increase the number of scheduler calls significantly, it allows testing every possible meeting interval. It should be noted that out of the 73 scheduled instances only 6 were scheduled in the first phase of JAS, i.e. solved trivially. In the rest, the meeting was successfully scheduled in the second phase, which attempted to reschedule the minimum number of users by using the min_ints list.

The utility change for the rescheduled users was minor in most of the cases. Table 3, shows the maximum, minimum and average percentage of utility change in the agent's plans without taking into account the added utility of the introduced meeting, in order to evaluate how rescheduling affected the previous activity plan of the agent. The worst drop was -15.62%. As the SWO+SA scheduler is stochastic there were many cases were there were even minor utility improvements in the rescheduled plans of the users. The best improvement was 2.46%. Obviously, for the cases that the meeting was not successfully scheduled, the agent utilities did not change and these cases are marked with a dash (-). From the total 175 rescheduled users, there were 95 utility drops in the rescheduled plans and 74 minor improvements. The rest had plans with the same utility.

Conclusions

The work described in this paper addresses the problem of automated meeting scheduling between a number of self-

[3]The full source code and the experiments' results are available online at: `https://drive.google.com/file/d/1vf_c728GK22hsz_tybo--uREZN_RN9-g/view`

[4]The SWO-SA scheduler is not an exact optimization algorithm.

[5]With a utilization value of 100%.

Table 2: Results of applying JAS to Meeting Scheduling Problems.

Size of Team	*2 Participants*			*3 Participants*			*4 Participants*			*5 Participants*		
Meeting ID	Interval	Ph	Rs	Interval	Ph	Rs	Interval	Ph	Rs	Interval	Ph	Rs
	[7..11]	1	0	[16..20]	2	2	[1..5]	2	1	[9..13]	2	2
	[9..13]	1	0	[9..13]	1	0	[1..5]	2	3	[1..5]	2	4
1	[10..14]	2	1	[16..20]	2	2	[16..20]	2	1	[1..5]	2	3
	[6..10]	1	0	[7..11]	2	2	[1..5]	2	3	[16..20]	2	1
	[10..14]	1	0	[1..5]	2	3	[16..20]	2	3	[8..12]	2	2
	[147..151]	2	2	[145..149]	2	3	[146..150]	2	4	[147..151]	2	4
	[147..151]	2	1	[147..151]	2	2	[146..150]	2	4	[144..148]	2	4
2	[144..148]	2	2	[144..148]	2	3	[146..150]	2	3	[144..148]	2	5
	[144..148]	2	1	[147..151]	2	2	[144..148]	2	3	[144..148]	2	5
	[144..148]	2	2	[144..148]	2	2	[144..148]	2	4	[147..151]	2	3
	[16..20]	2	1	[8..12]	2	1	[16..20]	2	2	[1..5]	2	3
	[16..20]	2	1	[11..15]	2	2	[16..20]	2	2	[11..15]	2	5
3	[16..20]	2	2	[16..20]	2	1	[16..20]	2	2	[16..20]	2	2
	[16..20]	2	2	[8..12]	2	2	[16..20]	2	1	[16..20]	2	4
	[7..11]	1	0	[16..20]	2	3	[10..14]	2	1	[16..20]	2	2
	[143..147]	2	2	no schedule	2	0	[143..147]	2	4	no schedule	2	0
	no schedule	2	0	no schedule	2	0	no schedule	2	0	[143..147]	2	5
4	[143..147]	2	2	[143..147]	2	3	[143..147]	2	4	[143..147]	2	5
	[143..147]	2	2	[143..147]	2	3	no schedule	2	0	[143..147]	2	5
	[143..147]	2	2	[143..147]	2	3	[143..147]	2	4	no schedule	2	0

Table 3: Utility Gains Change (%) when Scheduling a Meeting

Size of Team	*2 Participants*			*3 Participants*			*4 Participants*			*5 Participants*		
Meeting ID	max	min	avg	max	min	avg	max	min	avg	max	min	avg
	0	0	0	0	-2.26	-0.92	0	-0.87	-0.22	0.35	-2.46	-0.42
	0	0	0	0	0	0	0.08	-2.69	-0.78	0.29	-2.63	-0.41
1	0	0	0	0.02	-1.85	-0.61	0.05	0	0.01	**2.46**	-6.61	-0.85
	0	0	0	0.31	0	0.1	0.2	-2.35	-0.73	0	-0.02	0
	0	0	0	0	-7.28	-3.25	0	**-15.62**	-3.91	0.11	-1.67	-0.31
	0.48	-0.41	0.04	0.43	-3.14	-0.82	0.05	-1.44	-0.73	0.53	-1.07	-0.23
	0	-0.04	-0.02	0.45	-0.1	0.12	0.37	-0.14	0.12	0.97	-3.94	-0.52
2	-0.17	-6.05	-3.11	0.32	-5.15	-1.71	1.19	-3.43	-0.3	0.68	-15.08	-3.39
	0.09	0	0.05	0	-1.09	-0.54	0.36	-15.14	-3.7	0.07	-5.75	-1.7
	0.26	-0.36	-0.05	0.97	-0.61	0.12	0.06	-0.81	-0.23	0.09	-0.3	-0.05
	0.03	0	0.02	0	-0.32	-0.11	0	-0.52	-0.26	0.19	-0.47	-0.09
	2.13	0	1.07	0.41	-0.21	0.07	1.19	0	0.31	-0.12	-0.93	-0.43
3	1.17	-0.19	0.49	0.08	0	0.03	0.85	-0.9	-0.01	0.16	0	0.06
	1.15	-0.99	0.08	0	-0.59	-0.25	0	-0.26	-0.07	0.15	-0.15	-0.03
	0	0	0	1.74	-0.98	0.35	0	-0.03	-0.01	0.2	-1.47	-0.25
	0.06	0.03	0.05	-	-	-	0.28	-0.59	0.02	-	-	-
	-	-	-	-	-	-	-	-	-	-0.07	-1.23	-0.6
4	0.16	0.02	0.09	1.87	0.07	0.76	0.78	-0.88	-0.06	0.2	-1.18	-0.3
	0.98	-0.43	0.28	0.27	-0.95	-0.21	-	-	-	0.14	-1.29	-0.38
	-0.04	-0.26	-0.15	1.23	-0.17	0.37	0.1	-1.45	-0.35	-	-	-

interested agents, under a rich model of individual activities and a typical model for the meeting. As commonly used in other work, the multi-agent negotiation process that takes place is driven by a coordinator agent responsible for generating proposals, to which agents reply based on utility values derived from employing a greedy construction and stochastic optimization scheduler to the new scheduling problem that includes the proposal. Initial experimental evaluation, demonstrates that the approach performs well, reaching an agreement on the meeting time slot in the majority of experiments.

There are several directions that the present work can be extended. Currently, we do not consider alternative meeting locations in proposals, a feature that would certainly increase the number of necessary negotiation rounds, and although such an option in most business cases might not be of great interest, it might be interesting in other cases. JAS can also be extended so that meetings could be converted to full joint-activities, that is both temporal preferences (i.e., schedule the meeting at morning/noon/evening) and binary preferences could be defined over them by each agent, as in individual activities.

More experiments should also be conducted to evaluate the proposed approach. JAS is deterministic and should always arrive to the same common scheduled interval (given a deterministic rescheduler), but SWO+SA is stochastic. When the interval being tested by SWO+SA is given a high enough utility, JAS should always arrive to the same common scheduled interval, while the rest of the users rescheduled plans may be different though with different utilities between runs. By giving the interval being tested a low enough utility, it is possible that JAS could output different common scheduled intervals between runs. This needs to be tested. Moreover, different parameters for the algorithm, as well as different heuristics for the ordering of the intervals to be evaluated could also be tested. Future experiments should also measure the number of tested intervals and provide a metric for the computational efficiency of the proposed algorithms.

Rearranging agent meetings to accommodate the one under negotiation, i.e. "bumping", presents also a very interesting research direction that we aim to investigate, since there is a number of interesting questions that arise, as for example when should this "recursive" process terminate.

Finally, an experimental evaluation of the proposed algorithms in a real distributed environment would allow to measure the algorithms performance in terms of execution time and number of messages exchanged in order to fully evaluate its potential in a real-life setting. Obviously, integrating the proposed algorithms with modern calendar applications is the ultimate goal.

References

Alexiadis, A., and Refanidis, I. 2016a. Alternative plan generation and online preference learning in scheduling individual activities. *International Journal on Artificial Intelligence Tools* 25(3):1–28.

Alexiadis, A., and Refanidis, I. 2016b. Optimizing individual activity personal plans through local search. *AI Communications* 29(1):185–203.

BenHassine, A., and Ho, T. B. 2007. An agent-based approach to solve dynamic meeting scheduling problems with preferences. *Engineering Applications of Artificial Intelligence* 20(6):857–873.

Berry, P.; Gervasio, M.; Peintner, B.; and Yorke-Smith, N. 2007. Balancing the needs of personalization and reasoning in a user-centric scheduling assistant. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INLIGENCE CENTER.

Berry, P. M.; Donneau-Golencer, T.; Duong, K.; Gervasio, M. T.; Peintner, B.; and Yorke-Smith, N. 2009. Evaluating User-Adaptive Systems: Lessons from Experiences with a Personalized Meeting Scheduling Assistant. In *IAAI*, volume 9, 40–46.

Brzozowski, M.; Carattini, K.; Klemmer, S. R.; Mihelich, P.; Hu, J.; and Ng, A. Y. 2006. groupTime: preference based group scheduling. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, 1047–1056. ACM.

Chun, A.; Wai, H.; and Wong, R. Y. M. 2003. Optimizing agent-based meeting scheduling through preference estimation. *Engineering Applications of Artificial Intelligence* 16(7):727–743.

Franzin, M. S.; Freuder, E. C.; Rossi, F.; and Wallace, R. 2002. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In *Proceedings of the AAAI Workshop on Preference in AI and CP*.

Jennings, N. R., and Jackson, A. J. 1995. Agent-based meeting scheduling: A design and implementation. *Electronics letters* 31(5):350–352.

Modi, P. J., and Veloso, M. 2005. Bumping strategies for the multiagent agreement problem. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 390–396. ACM.

Modi, P. J.; Veloso, M.; Smith, S. F.; and Oh, J. 2005. Cmradar: A personal assistant agent for calendar management. In *Agent-Oriented Information Systems II*. Springer. 169–181.

Payne, T. R.; Singh, R.; and Sycara, K. 2002. Rcal: A case study on semantic web agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, 802–803. ACM.

Refanidis, I., and Yorke-Smith, N. 2010. A constraint-based approach to scheduling an individual's activities. *ACM Trans. Intell. Syst. Technol.* 1(2):12:1–12:32.

Sen, S., and Durfee, E. H. 1996. A contracting model for flexible distributed scheduling. *Annals of Operations Research* 65(1):195–222.

Wang, F. 2003. Adaptive meeting scheduling for large-scale distributed groupware. *BT technology journal* 21(4):138–145.

Zunino, A., and Campo, M. 2009. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications* 36(3):7011–7018.

Planning and Scheduling in Additive Manufacturing

Filip Dvorak
Maxwell Micali
Mathias Mathieu
Oqton
832 Sansome Street
San Francisco, California 94111
{filip.dvorak,maxwell.micali,mathias.mathieu}@oqton.com

Abstract

Recent advances in additive manufacturing (AM) and 3D printing technologies have led to significant growth in the use of additive manufacturing in industry, which allows for the physical realization of previously difficult to manufacture designs. However, in certain cases AM can also involve higher production costs and unique in-process physical complications, motivating the need to solve new optimization challenges. Optimization for additive manufacturing is relevant for and involves multiple fields including mechanical engineering, materials science, operations research, and production engineering, and interdisciplinary interactions must be accounted for in the optimization framework.

In this paper we investigate problem where a set of parts with unique configurations and deadlines must be printed by a set of machines while minimizing time and satisfying deadlines, bringing together bin packing, nesting (two-dimensional bin packing), job shop scheduling, and constraints satisfaction. We first describe the real-world industrial motivation for solving the problem. Subsequently, we encapsulate the problem within constraints and graph theory, create a formal model of the problem, discuss nesting as a subproblem, and describe the search algorithm. Finally, we present the datasets, the experimental approach, and the preliminary results.

Introduction

As engineered components and aesthetic creations become more advanced and intricate, they push the boundaries of what is manufacturable via traditional manufacturing processes. These boundaries may be of a technical nature, in which certain geometries or certain materials simply cannot be produced with traditional equipment; or they may be economic in nature, such that the high mix and low volume of production orders cannot be manufactured in a cost-efficient manner due to high tooling and labor costs incurred with each associated setup. The onset of additive manufacturing (AM) technologies has presented new capabilities to engineers and designers, greatly expanding the domain of the manufacturable design space along several dimensions, allowing for design behaviors and concepts which previously may have only been conceivable and not producible: mass customization of components; intricate geometries with no restrictions on visibility, draft angles, or other design requirements; and exotic material behaviors such as negative stiffness and negative thermal expansion (Duoss et al. 2014).

For the manufacturing and production engineers, AM technologies allow components to be printed "on demand", and for the supply chain to consist of digital files rather than physical components. Figure 1 shows an example of a complex geometry which can only be produced by additive manufacturing. While the unit production cost for a single additive component may be higher when compared to full-scale traditional production, additive manufacturing is drastically less expensive at low and medium volume production. The industry is beginning to embrace these new manufacturing methods in order to streamline aspects of production, such as the well-known example of GE printing fuel injector nozzles for jet engines in a single print (Coykendall et al. 2015). The nozzle previously required producing of 20 complicated components, as well as the additional labor to weld and braze them into an assembly afterward. In addition to being more streamlined to produce, the additive version of the nozzle was also 25% lighter, which directly translates to additional fuel savings for the aircraft (Conner et al. 2014).

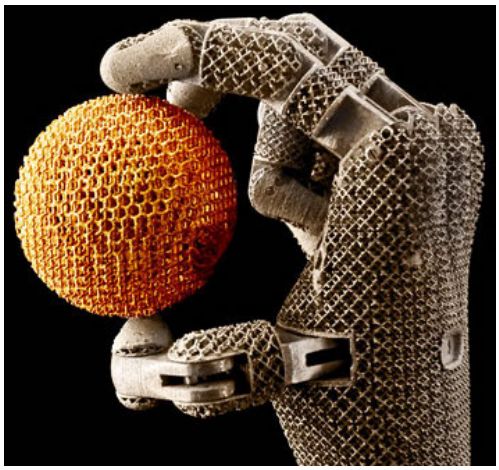


Figure 1: Example of a component which is impossible to manufacture by traditional manufacturing techniques, yet is able to be printed despite the intricate lattice structures. (Kalpakjian and Schmid 2013).

Additive manufacturing can result in drastic reductions in the overall production time of a component. This efficiency stems from reducing the number of procedures involved in

the production process, even if a typical print takes longer than a typical process step in a traditional manufacturing flow. Whereas a single machining operation may take on the order of minutes or hours to execute, a single print is on the order of hours or days. However, traditional production of a part may require dozens of piecewise machining operations for completion, while a print is a single-operation process. Both additive and traditional manufacturing may require some standard finishing operations to accomplish things such as achieving the desired surface finish or removing structures used to fix the component to the machine during production, and these times are process-independent.

Given that additive manufacturing consists of a single, time-consuming, step and that there is an opportunity to fit multiple components onto the same build plate so they can be produced in parallel while incurring only marginal additional time costs, planning and scheduling for additive manufacturing at a factory scale brings a unique set of emerging opportunities and challenges while attempting to optimize the process.

In the rest of this section we describe some existing challenges in additive manufacturing and establish the problem of optimizing scheduling and planning for AM.

Additive Manufacturing

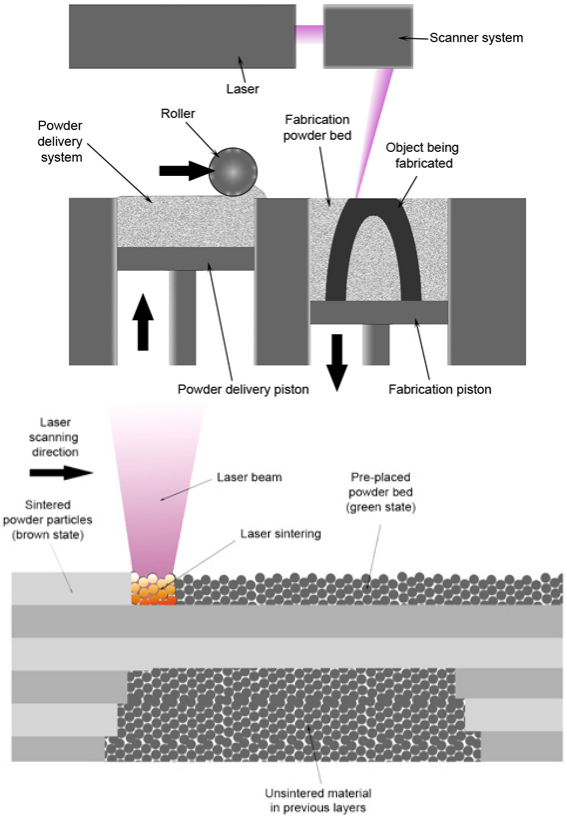


Figure 2: Schematic of selective laser melting (SLM), which is classified as a powder bed fusion process (Wikimedia Commons 2018).

Additive manufacturing, also known as 3D printing and other names, has existed as a commercial technology since 1984 with the invention of stereolithography, although the overall vision of precisely replicating physical objects preceded the invention of stereolithography by a century (Weber et al. 2013). The roots of AM technology date back to the 1800s with developments in the fields of photo sculpture and topography, but the overall vision of printing three-dimensional objects was not possible until advancements in materials science, computer-aided design (CAD), computer numerical control (CNC), and laser technology were made, triggering the rapid development of a variety of AM techniques which can be grouped into seven major technology categories (ASTM 2018):

- *Binder jetting*—an additive manufacturing process in which a liquid bonding agent is selectively deposited to join powder materials.
- *Directed energy deposition*—an additive manufacturing process in which focused thermal energy is used to fuse materials by melting as they are being deposited.
- *Material extrusion*—an additive manufacturing process in which material is selectively dispensed through a nozzle or orifice.
- *Material jetting*—an additive manufacturing process in which droplets of build material are selectively deposited.
- *Powder bed fusion*—an additive manufacturing process in which thermal energy selectively fuses regions of a powder bed.
- *Sheet lamination*—an additive manufacturing process in which sheets of material are bonded to form an object.
- *Vat photopolymerization*—an additive manufacturing process in which liquid photopolymer in a vat is selectively cured by light-activated polymerization.

Figure 2 shows a schematic of the selective laser melting (SLM) process, which is a type of powder bed fusion process. SLM is typically used for producing metallic parts, and due to its industrial value in many manufacturing verticals, SLM is the AM process focused on in this paper.

Many technological advancements have enabled the existence of AM processes, yet many challenges still remain as the field matures. One such challenge is how to incorporate additive manufacturing into a traditional manufacturing process flow, since AM has different planning and scheduling requirements and considerations. Other challenges involve understanding, modeling, and controlling complex physical behaviors and phenomena involved in the process, which are ultimately influenced by the planning and scheduling decisions made upstream of the process. Some examples of these decisions are how to optimally orient the geometries in space for printing, and how to optimally nest multiple parts within a single print without inducing any build failures. "Optimal" may be defined in terms of overall print cost, overall print quality, or some other metric of optimality. It is important to note that some process decisions may induce physical conditions within the print which yield total process failures, with the onset of cracking, layer delamination, or unacceptable degrees of component warping. For this paper, we will

assume that planning and scheduling decisions are made in fully observable and deterministic world where all possibilities succeed.

Related Work

Additive manufacturing has been an emerging technology for several decades, and each advancement has provided new alternatives to traditional manufacturing methods for candidate geometries and applications. Mass production via additive manufacturing is now viable, yet the operations research techniques designed for traditional manufacturing are not directly transferable to AM. A number of cost models for AM have been recently developed in (Fera et al. 2017) and (Lutter-Günther et al. 2015), a survey of cost models was performed in (Costabile et al. 2017), and various AM products have been mapped in (Conner et al. 2014). However, only a few models for production planning of AM have been proposed. Notably, (Li, Kucukkoc, and Zhang 2017) has proposed a constraint optimization model on top of CPLEX, which does not take into account deadlines and approximates nesting by the total surface area. In (Chung, Chan, and Chan 2009), the authors implemented a model in MATLAB which takes into account nesting and scheduling metrics of earliness and tardiness, however they only consider a single machine.

Challenge Statement

Production planning in AM starts when a customer sends design specifications (CAD file) of a part that needs to be manufactured, along with a production deadline and delivery location. The design specifications of the part include geometric dimensioning and tolerancing (GD&T) information, which restricts the decision space in the production planning process. For example, the GD&T information may specify tolerances in regions of the design which are not achievable by some types of printing processes. Since machines are capable of printing multiple parts simultaneously in a single print operation, it is important to consider optimizing the constitutive members of such sets and how they fit together. Each part has its printing orientation, and they must collectively fit into the build volume of the machine – typically a regular hexahedron defined by height, width, and length. A single print operation is called a build, and we assume that the downward projections of different parts within a build cannot overlap. The duration of each build is functionally dependent on the speed that the laser beam in the machine is programmed to travel, the maximum height of the set of parts in the build, and the total path traveled by the laser beam during the build. Each machine behaves as a unary resource, and thus can execute only a single build at any moment in time.

The problem consists of a set of parts, N, including their possible orientations, configurations, and deadlines, as well as a set of machines, M. The solution to the problem is the set of builds, B, scheduled to machines, M, which maximizes the number of parts printed prior to their deadlines, while also minimizing the overall printing cost.

In this paper we are relaxing several requirements which are considered during production planning. In particular, the shipping deadlines to given delivery locations are relaxed by projecting the upper bound of shipping time into the scheduling deadline. In reality, the location of manufacturing assets around the globe would influence scheduling deadlines, but upper bounding of resource requirements for logistics allows for treating all machines as though they are in a single location. We consider 2D bin-packing (nesting) when combining different parts into a single build. While 3D bin-packing is conceptually possible, i.e. by using horizontal bridges, we do not consider it in this paper. Finally, builds may require additional preparation and post-processing steps when the machine is cleaned, the material is recycled, the machine configuration is adjusted, and small finishing operations are required on the parts before shipment. Those steps are performed by human operators that have their own shifts and schedules that vary across different time-zones (i.e. a high priority order may start to be printed in Hungary, because the operators are already up and can configure the machines, while a factory in Nevada is printing their night jobs). We are relaxing the human operators and upper bounding their work into the duration of printing a build, which also allows for relaxing the temporal reasoning to the sequence of builds on each machine.

The problem we are solving consists of multiple nested NP-hard problems (bin-packing, set cover, job-shop), and to describe the structure of our model we first encapsulate the configuration of parts. In the next section we formulate the assignment of parts into builds as a graph decomposition. Next we describe the challenges of nesting parts together, and finally we describe the complete model with experimental evaluation.

Configuration

A manufacturing configuration for a part P is a collection of rules that need to be followed to maximize the expectation of achieving the desired quality of part P. We model a manufacturing configuration O of part P as a constraint satisfaction problem $O = (V^p \cup V^m, C)$, where V^p are *persistent* variables that relate only to the part P, V^m are variables related to the manufacturing process of P that merge through composition, and constraints C then propagate across V^p variables through V^m variables. We say that the configuration O is *valid* iff there exists an assignment α of values to the variables $V^p \cup V^m$ such that all constraints in C are satisfied.

Having a set of all possible configurations Ω we define composition $\bigodot : \Omega \times \Omega \rightarrow \Omega$ as a function that composes two configurations into another configuration. Having two configurations $O_1 = (V_1^p \cup V^m, C_1)$ and $O_2 = (V_2^p \cup V^m, C_2)$ we construct a composition $O_1 \bigodot O_2 = (V_1^p \cup V_2^p \cup V^m, C_1 \cup C_2)$. As a practical example we can imagine having a part P, its configuration $O = (V^p \cup V^m, C)$, persistent variable $orientation \in \Re \times \Re \times \Re$, variable $height \in \Re$, and constraint $height >= h(O)$ (for simplicity we assume that function h computes the height of the part P given its configuration O that contains the $orientation$ variable and volumetric data). We can note that for a composition of two parts, we will determine that the printing height has to be larger than the maximum height of either part. In similar

Figure 3: Example of a compatibility graph for a problem with 20 machines and 50 parts. Nodes of the same color belong to the same clique. The clusters correspond to different materials.

fashion we can compose deadlines for different parts and all the configuration variables.

Compatibility Graph

Having a set of parts Π and their configurations Ω, we assume that all configurations are valid (invalid configurations trivially reflect a failure in the part preparation process), then we say that two configurations O_1 and O_2 are *compatible* iff $O_1 \bigodot O_2$ is valid. We can see that compatibility is a symmetrical, reflexive and non-transitive relation. Having a set of nodes $N = \Omega$ and a set of edges $E = \{(x, y)|x$ is compatible with $y\}$ we define the *compatibility graph* as $G = (V, E)$. We can observe that a build which combines together multiple parts has to be a complete subgraph (a clique) within the compatibility graph.

A compatibility graph allows us to represent relationships between individual parts, to provide a structure that can be quickly searched for build candidates, and to record structural information from previous searches, i.e. no-goods such as two parts that should not be in the same build because one of their deadlines will be violated.

Graph Decomposition

Choosing a single clique in the compatibility graph represents creating a job and scheduling it on a printer. Consequently, choosing a set of cliques that cover the whole graph without intersections on nodes represents a decomposition of the graph into complete subgraphs, and it is also equivalent to the exact set cover problem (Cover and Thomas 1991) with an additional restriction that each subset of the covering is a clique in the compatibility graph. Figure 3 illustrates a graph decomposition into cliques. Having an initial state I of the search space in which all cliques contain only a single node, we can look at the search space of the problem as a tree that merges two cliques at each branch if the parts within the two cliques can be nested together, until no two cliques can be merged. However, the size of the combinatorial space of choosing the cliques to merge one by one is $O(n^n)$, impractical for any exhaustive search algorithm.

While pairwise compatibility is a necessary condition for having a valid build, it is not a satisfactory condition, which is provided by running the nesting algorithm described in the next section.

Nesting

The nesting algorithm aims to efficiently arrange a given set of parts on a given build plate size by maximizing the total area covered with parts. Each part has an associated economic value and several possible orientations, which are considered under several (< 10) different rotations around the z-axis. The nesting algorithm maximizes the value of the build plate.

Nesting operates at three levels of fidelity with varying degrees of computational complexity - the highest fidelity nesting takes the longest to compute, while the lowest fidelity nesting is computed the fastest. Low-fidelity nesting uses 2D bounding boxes, medium-fidelity nesting uses silhouettes while packing the parts greedily, and high-fidelity nesting explores the search space using several search algorithms.

Low Fidelity At the lowest fidelity of nesting we use the 2D bounding boxes of the given parts. First these rectangles are sorted in a descending sequence by the value of the part, then by their shortest side, and finally by their longest side. We place these rectangles using the Maximal Rectangles Best Short Side Fit algorithm (Jylänki 2010). This technique keeps track of the maximal free rectangles in the bin, these are the rectangles remaining after placing the bounding boxes. It then places each rectangle in one of these free rectangles, minimizing the shortest leftover side.

Medium Fidelity This level of nesting uses a greedy algorithm to place the parts. First the silhouettes of the parts are constructed. The resulting polygons are sorted in descending sequence by the value of the object, and then by area. Then the polygons are placed one by one on the build plate. For each polygon we compare all possible rotations and positions, and choose the combination that results in the smallest bounding box. For this level of nesting only a single orientation per object is considered. Figure 4 shows an example of medium fidelity nesting.

High Fidelity The highest fidelity nesting procedure explores the solution space using several search strategies - Genetic Algorithm, Simulated Annealing and Tabu Search. A solution to the problem consists of the order in which we place the parts, and the orientation and rotation of each part. The placement of parts is done with the Bottom-Left strategy (Dowsland, Vaid, and Dowsland 2002), where we place each parts as close to the bottom-left of the build plate as possible. The nesting algorithm either maximizes the total area used area or the total value of the parts in the build.

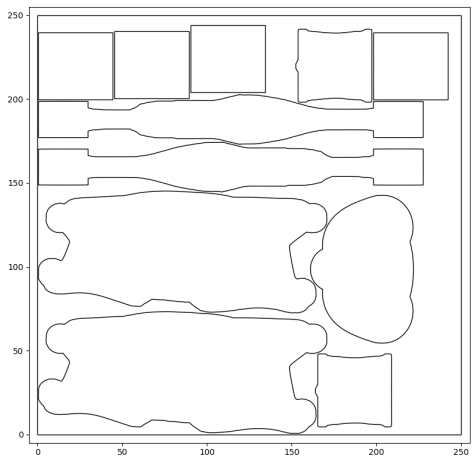


Figure 4: Example result from medium fidelity nesting.

Mathematical Model

We build up a representation of the problem as a meta constraint optimization problem (Dechter 2003) as a pair (V, C), where V is a set of variables and C is a set of constraints on top of V, where some constraints are hard and always need to be satisfied (a single machine cannot perform two activities at once), while others are soft (not every part can always be finished on time). Then the solution of the problem (V, C) is an assignment α that assigns exactly one value to each variable while all the hard constraints are satisfied. The quality of the solution is further evaluated based on the number of unsatisfied constraints, which is represented in the objective function.

We define the problem variables as follows:

- $P = \{p_1, .., p_n\}$ is a set of parts.
- $B = \{b_1, .., b_n\}$ is a set of builds to be printed.
- $M = \{m_1, .., m_m\}$ is a set of machines.
- d_{p_i} denotes the deadline for part p_i.
- d_{b_i} denotes the deadline for build b_i.
- $dur_{b_i}^{m_j}$ denotes the duration for the build b_i on machine m_j.
- c_{m_i} denotes the configuration of the machine m_i.

The decision variables of the problem are the following:

- c_{p_i} denotes the configuration assigned to the part p_i.
- $a_{p_i} = b_j$ denotes the build b_j is assigned to part p_i, we can also write $p_i \in b_j$.
- $a_{b_j} = m_k$ denotes the machine m_k assigned to build b_j, we can also write $b_j \in m_k$.

On top of the variables we are going to use multiple constraints and a strategy for ordering builds at the individual machines, described in the following sections.

Constraints

We use the following constraints to maintain the consistency of the solution and cut symmetrical parts from the search space. Note that some of the hard constraints are implicitly encoded in the representation, i.e. the constraint that a build can only be assigned to a single machine, and that a part can only be assigned to a single build. These constraints are written below for completeness:

$$\bigcup_{b \in B} b = P$$
$$\forall b_i, b_j \in B : b_i \cap b_j = \emptyset$$
$$\bigcup_{m \in M} m = B$$
$$\forall m_i, m_j \in M : m_i \cap m_j = \emptyset$$

Compatibility Constraints enforce that parts within a single build are compatible with themselves, as well as the machine which is processing the build. We define the constraint as follows:

$$\forall b_i \in B, a_{b_i} = m_j : c_{m_j} \bigodot \prod_{x \in \{y | a_y = b_i\}} x \text{ is valid.}$$

The compatibility constraint is evaluated by solving (finding a witness solution) of the inner CSP problem, where the compatibility graph provides fast evaluation and filtering of impossible candidates. Preprocessing of the compatibility graph consists of computing the compatibility relation between every pair of possible configurations coming from two different parts, likewise every possible part configuration with every machine. The cost of preprocessing of the compatibility graph is negligible, $O(n^2)$, where n is the number of parts.

Nesting Constraints are global constraints whose evaluation runs asynchronously in parallel with the main search algorithm. The main search algorithm uses the total surface area as an upper bound on how densely nested the parts can be in a build. Then those builds are sent to the nesting algorithm which provides a list of the parts that do fit into the build and the parts which are leftover from the build. We denote the set of all builds that were returned by the nester as Φ and also have $\emptyset \in \Phi$, representing that an empty build is trivially satisfied. We define the nesting constraint as follows:

$$\forall b_i \in B : \{p_j | a_{p_j} = b_j\} \in \Phi.$$

In other words, all builds need to be empty or confirmed by nester through the set Φ. We treat the nesting constraint as a soft constraint, where each build not contained in Φ represents a violated constraint. For the purpose of this paper, nesting within the nesting constraint is done at the medium fidelity level.

Deadline Constraints guarantee that the parts are finished on time. We define the deadline for a build as $d_{b_i} = min_{p_j \in b_i} d_{p_j}$, in other words, a build's deadline is the earliest deadline among its parts. Then, assuming time at the beginning of the world is 0, we define the deadline constraints as follows:

$$\forall m_j \in M, \forall b_i \in m_j : \sum_{b \in m_j | d_b <= d_{b_i}} dur_b^{m_j} <= d_{b_i}$$

In other words, on a single machine the deadlines of all the builds must occur after the sum of durations of the builds whose deadlines are earlier. It may not be always possible to fulfill all the deadline constraints, and we may instead treat them as soft constraints which need to be minimized.

The computational problem of evaluating the deadline constraints is a standard scheduling problem that we focus on in the next section.

Tardy Builds

Once a build is assigned to a machine we can consider it to be independent of the builds assigned to the other machines. Such independence would disappear if we also considered that operators can service only a single machine at a single moment and whose availability is restricted. The advantage of the independence between machines is that instead of extending the state space with precedence constraints between builds on each machine we can choose a strategy that orders the builds. This approach follows the scheme of machine-based problem decomposition (Pinedo 2008).

Having multiple builds assigned to a single machine, we are mostly interested in whether a build is either on time, or if it is late and by how much it has missed its deadline. There are several approaches to model tardiness. We are going to use the $\alpha|\beta|\gamma$ notation (Graham et al. 1979) for categorizing them as scheduling problems.

- Minimizing Total Tardiness of Builds. We minimize the sum of tardiness of all builds that missed the deadline. Categorized as $1||\sum T_j$, the problem is known to be NP-hard (Pinedo 2008).
- Minimizing the number of Tardy Builds. The number of tardy builds represents the number of builds that have at least one late part. We can solve the problem $1||\sum U_j$ with an optimal algorithm in $O(nlog(n))$.
- Minimizing the number of weighted Tardy Builds. We can further extend the representation of tardy parts by adding a weight to each of them – $1||\sum w_j U_j$, which also makes the problem NP-hard – we can imagine to have all deadlines failing and we get the knapsack problem.

For the purpose of this paper we are going to minimize the number of tardy builds $1||\sum U_j$, which is efficiently computed using an adaptation of a standard scheduling algorithm (Pinedo 2008). The algorithm has two steps:

- Order the builds into an ascending sequence L based on the earliest deadline among the parts each build contains.
- Keep adding builds into a sequence S for as long as all builds in S are on time. If S has a build that misses the deadline, remove from S the build with the longest duration.

The algorithm is optimal in minimizing the number of builds that have at least one order miss its deadline (Pinedo 2008), however it does not guarantee optimality in minimizing the total number of orders that are late. In the trivial case, when each build consists of a single order, the algorithm is optimal for minimizing the number of late parts as well. However, minimizing the number of late parts is NP-hard in general. We can show that when the deadlines for all parts are the same then the number of items in a build corresponds to a cost of an item in the knapsack problem. Consequently, we can translate a knapsack problem into minimizing the cost of parts that miss deadline.

Objective Function

The primary optimization criterion is to minimize the number of unsatisfied soft (deadline and nesting) constraints, and then minimize the makespan (maximum execution time) of the schedule. Given that the nester is run asynchronously, its output becomes integrated into the main search algorithm via the lazy evaluation of the Nesting Constraints.

Search Algorithm

The average problem size prevents the use of exhaustive search techniques such as branch and bound. Where we cannot expect solutions in reasonable time, we instead adopt local search methods aided by (meta)heuristics. We use a portfolio of local-search algorithms on top of the CSP representation of the problem combined with a range of steps that define the neighborhood for the local search. We use the following local search algorithms:

- Hill Climbing makes the move to the lowest cost state in the neighborhood.
- Tabu Search prevents Hill Climbing from being stuck in a cycle by remembering a list of recently visited states that should not be visited again.
- Simulated Annealing begins at one state, and at each step it can choose a state with the same minimal cost as HC, or it can choose a state that, with some probability, does not have the minimal cost. The probability of choosing sub-optimal states reduces in time as the algorithm progresses, hence Simulate Annealing is more likely to escape from local optima early, while behaving like HC after certain amount of time.
- Step Counting uses the cost of the current state as a bound for several future states.
- Late Acceptance makes moves to states that are at least as good as the state several steps ago.

The neighborhood in the search space is defined through four atomic steps that generate successor states:

- *emptyMove* chooses builds b_i, b_j, part p and machine m_k, where $|b_i| > 1$, $p \in b_i$, and $b_j = \emptyset$. Then it removes p from b_i, adds it to b_j, and assigns b_j to m_k. This move represents a situation, where we remove a part from an existing build to start a new build, and then move that new build to a new machine. The main advantage of this step is symmetry breaking – it does not matter which empty build has been chosen, since we assign it to a new machine right away and we only assign it to a compatible machine.
- *moveBuild* chooses a build b_i, b_i and machine m_j and assigns b_i to m_j.
- *moveOrder* chooses a build b_i, $|b_i| > 0$ and part p_j and assigns p_j to b_i.

- *changeConfiguration* chooses a part p and a possible configuration $c \in \Omega$ and performs assignment $c_{p_i} = c$.

These four steps together give access to the whole search space of the problem, and the goal of the search algorithm is to efficiently explore it using heuristics.

Heuristics

We use several problem-specific heuristics that encapsulate some knowledge about the problem and a number of problem-independent heuristics. The first heuristic run is the *construction heuristic* that creates the initial state from which the search algorithm starts to explore the search space. The construction heuristic solely assigns each order into a build and then moves it to one of the compatible machines.

We use tie-breaking heuristics for variable and value selection. In particular, we choose configurations of parts whose height is the largest among the heights that still fit into the machine. When an order is moved between builds, the builds with close mean deadlines are tried first.

Solving Approach

Combining the previously defined structure and given a set of parts P and a set of machines M, we find the solution using following steps.

1. Pre-compute the compatibility graph for all configurations of all parts and all the machines. Then filter domains of the variables.
2. Create an initial state where each part is assigned to a different build and all the builds are assigned to some machines.
3. Perform a local search over the decision variables of the problem using the *moves* to change their values until the given time is spent.
4. Run nesting in parallel with the previous step, such that the builds confirmed by the nester are available in the nesting constraint and the nester is invoked whenever a new undiscovered and non-dominated build is considered by the nesting constraint.
5. Collect the assignments of parts to builds and schedule the builds to machines in time.

The simultaneous execution of steps 3 and 4 follows a *best effort* approach that allows two solvers for hard problems to exchange the information and reach higher quality solutions than if run in a sequence.

Experimental Evaluation

To gain insight into the difficulty of finding good solutions for the stated problem in additive manufacturing, we have created two problem generators, produced a dataset, and evaluated an implementation of our proposed model.

Note that at the time when we have run the experiments we have not yet integrated the nester implementing the nesting constraint, hence all the nesting constraints are considered violated across the search space and the nesting is only approximated as the total available surface.

Problem Generators

We use two types of problem generators. One generates random problems, and the other generates problem instances with known optimal values. The random problem generator RG for a given seed of randomness D produces a random problem using a uniform random generator integrated in Java with values in the following ranges:

- Machine is generated to support
  - Material $\in M, |M| = 5$.
  - Height $\in \{20, .., 70\}$ cm.
  - Width $\in \{50, .., 100\}$ cm.
  - Length $\in \{50, .., 100\}$ cm.
- Part is generated with the following features:
  - Material $\in Materials, |Materials| = 5$.
  - Configuration $\in C, |C| = 5$ and each configuration defines the following:
    - Volume $\in \{1, .., 100\}$ liters.
    - Height $\in \{10, .., 40\}$ cm.
    - Width $\in \{10, .., 50\}$ cm.
    - Length $\in \{10, .., 50\}$ cm, where $Volume = Length * Height * Width$.

In other words, we generate a selection of machines with various sizes and materials, then we generate parts which require certain material but can be oriented through the change of their configuration, which maintains the volume but modifies dimensions. Consequently, we create a new optimal problem generator ROG using the random generator RG as follows:

- Given seed D, use the Random Generator RG to produce a random problem.
- Run 20 seconds of hill-climbing to find a sub-optimal low-makespan schedule for the problem.
- Stretch all orders such that they perfectly occupy the space of the machines and that the builds perfectly occupy the time of the machines.
- Propagate the stretching of orders into its configurations such that the volumetric constraints are satisfied.
- Record the solution cost and randomize the perfect schedule.

Both RG and ROG generate a problem for each given seed and the numbers of machines and orders. RG generates problems that are normally distributed, and the performance of the planner upon those problems gives a realistic measure on how quickly the search algorithm can converge to some local optima, depending on the size of the problem, and where it is not helpful to invest more computational time. While the problems generated by ROG are not normally distributed, they give some estimation about the worst-case distance from the optimality when the planner converges to a local optima.

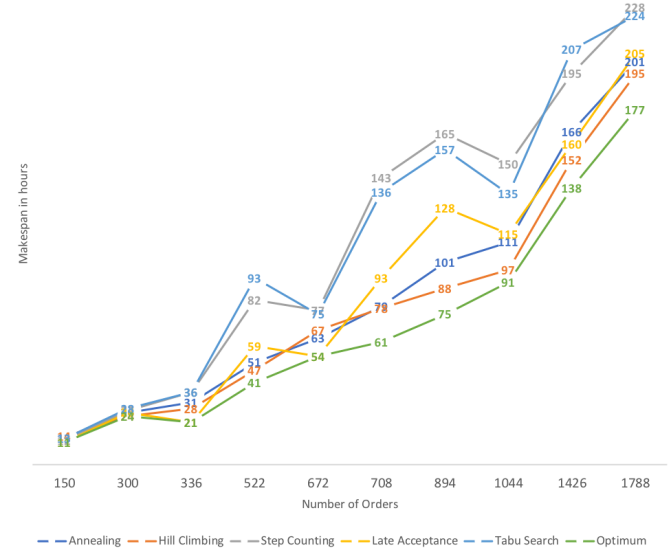


Figure 5: Shows the actual makespan in hours for how long it takes to execute the whole schedule. The graph includes the value for the precomputed optimal schedule and best schedule makespans found for other search algorithms run for 60 seconds.

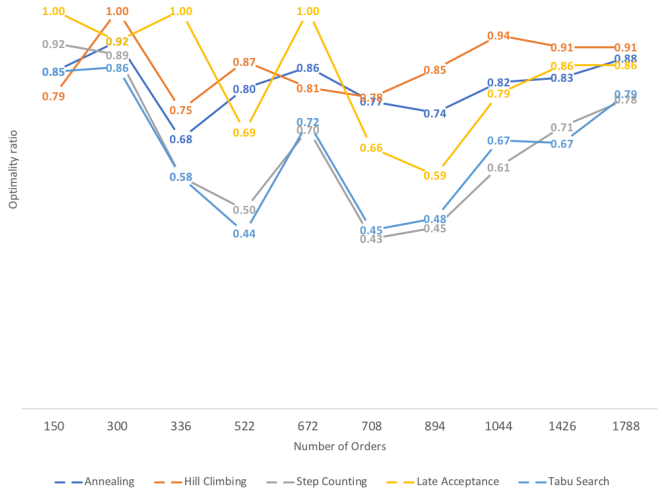


Figure 6: Shows the relative distance from the optimal makespan, computed as optimal/actual, found by different search algorithms running for 60 seconds on each problem instance. The optimum is a known value captured during the problem instance generation.

Implementation and Environment

We have implemented the model using the OptaPlanner (Smet 2018) constraint solver on top of a Java representation. To run the experiments we have used a set of homogeneous AWS EC2 instances, each with Intel CPU E5-2676@2.40GHz and 1GB of RAM, running on Ubuntu 16.04 and OpenJDK 1.8.

Preliminary Results

Using *ROG* we have generated a dataset of 10 problem instances ranging from 150 to 1788 parts. We have run each search algorithm for 60 seconds and took out the best solution found. All of the algorithms found solutions which satisfy all of the deadline constraints, at which moment the optimization turns into makespan minimization. Figure 5 shows the makespan values in hours that were required to execute all the schedules, while Figure 6 shows the relative distance from the optimal solution for each of the solvers.

The results indicate that even with an enormous search space it is possible to find reasonable solutions in a matter of minutes using simple local search algorithms and underlying constraint representations. Hill Climbing in particular seems to provide consistent performance, although it is sometimes surpassed by its variants that provide better makespans. Hill Climbing tends to get stuck in local optima traps and the other algorithms try to escape such traps through different relaxations of greediness of neighborhood exploration, leading to occasional successes such as Late Acceptance finding optimal solutions for 336 and 672 orders. The results are not conclusive with regard to which local search algorithm to choose, since the behavior can quickly change based on heuristics and neighborhood definitions, but it shows that local search is able to quickly provide reasonable solutions.

Conclusions

The focus of this paper has been to capture the fundamental difficulty involved in the newly emerging intersection between additive manufacturing, operations research, and artificial intelligence. We have introduced the field of additive manufacturing with its challenges; described and modeled the key optimization problem of nesting parts into builds and scheduling builds to machines while achieving deadlines and minimizing production time; and presented preliminary experimental results using an implementation on top of a constraint solver.

The main contribution of the paper is the new model which we hope to eventually extend into an end-to-end optimization model that captures all the discrete decision making challenges in additive manufacturing.

Future Development

This paper is one of the first attempts to describe the computational complexities faced in optimization for additive manufacturing. Multiple sub-problems have been relaxed and deserve further investigation. In particular, explicit temporal reasoning needs to be added to take into consideration operators that work with multiple machines and are required at different phases of the printing process. Then once the parts are printed, we need to take into consideration that they need to be packed and shipped to different destinations from factories at different locations around the world. Another pa-

rameter to explore is the fidelity of nesting, where we used only the medium fidelity in the experiments, but results may vary significantly for different fidelity levels and datasets. Finally, while the minimization of the number of tardy builds is a correct and satisfactory condition for achieving all deadlines, it may not accurately reflect the cost associated with the number of tardy parts when it is impossible to make all deadlines.

The experiments deserve further extension in direction of the time given to the planner per problem instance, various densities of constraints including over-constrained problems, and investigation of problems with lots of small parts, big parts, and various distributions among them.

References

ASTM International. 2018. *Standard Terminology for Additive Manufacturing Technologies*. ISO/ASTM 52900.

Chung, S. H.; Chan, F. T. S.; and Chan, H. K. 2009. A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling. *Eng. Appl. Artif. Intell.* 22(7):1005–1014.

Conner, B. P.; Manogharan, G. P.; Martof, A. N.; Rodomsky, L. M.; Rodomsky, C. M.; Jordan, D. C.; and Limperos, J. W. 2014. Making sense of 3-D printing: Creating a map of additive manufacturing products and services. *Additive Manufacturing* 1-4:64–76.

Costabile, G.; Fera, M.; Fruggiero, F.; Lambiase, A.; and Pham, D. 2017. Cost models of additive manufacturing: A literature review. 8:263–282.

Cover, T. M., and Thomas, J. A. 1991. *Elements of Information Theory*. New York, NY, USA: Wiley-Interscience.

Coykendall, J.; Cotteleer, M.; Holdowsky, J.; and Mahto, M. 2015. 3D opportunity in aerospace and defense: additive manufacturing takes flight. *A Deloitte series on additive manufacturing*.

Dechter, R. 2003. *Constraint Processing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Dowsland, K. A.; Vaid, S.; and Dowsland, W. B. 2002. An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research* 141(2):371–381.

Duoss, E. B.; Weisgraber, T.; Hearon, K.; Zhu, C.; Small, W.; R. Metz, T.; Vericella, J.; D. Barth, H.; Kuntz, J.; Maxwell, R.; M. Spadaccini, C.; and Wilson, T. 2014. Three-dimensional printing of elastomeric, cellular architectures with negative stiffness. *Advanced Functional Materials* 24(31):4905–4913.

Fera, M.; Fruggiero, F.; Costabile, G.; Lambiase, A.; and Pham, D. 2017. A new mixed production cost allocation model for additive manufacturing (miprocamam). 1–17.

Graham, R. L.; Lawler, E. L.; Lenstra, J.; and Rinnooy Kan, A. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium.* (5):287–326.

Jylänki, J. 2010. A thousand ways to pack the bin – a practical approach to two-dimensional rectangle bin packing. *retrieved from http://clb.demon.fi/files/RectangleBinPack.pdf*.

Kalpakjian, S., and Schmid, S. R. 2013. *Manufacturing Engineering and Technology*. Pearson, 7th edition.

Li, Q.; Kucukkoc, I.; and Zhang, D. Z. 2017. Production planning in additive manufacturing and 3D printing. *Computers & Operations Research* 83:157–172.

Lutter-Günther, M.; Wagner, S.; Seidel, C.; and Reinhart, G. 2015. Economic and ecological evaluation of hybrid additive manufacturing technologies based on the combination of laser metal deposition and CNC machining. In *Energy Efficiency in Strategy of Sustainable Production*, volume 805 of *Applied Mechanics and Materials*, 213–222. Trans Tech Publications.

Pinedo, M. L. 2008. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition.

Smet, G. D. 2018. *OptaPlanner User Guide*. Red Hat and the community. OptaPlanner is an open source constraint satisfaction solver in Java.

Weber, C. L.; Pena, V.; Micali, M. K.; Yglesias, E.; Rood, S. A.; Scott, J. A.; and Lal, B. 2013. The Role of the National Science Foundation in the Origin and Evolution of Additive Manufacturing in the United States. Technical report, IDA Science and Technology Policy Institute. IDA Paper P-5091.

Wikimedia Commons. 2018. https://upload.wikimedia.org/wikipedia/commons/3/33/Selective_laser_melting_system_schematic.jpg. User:Materialgeeza / CC-BY-SA-3.0.

A Large-scale Multiobjective Satellite Data Transmission Scheduling Technology Based on SVM+NSGA-II

Jiawei Zhang , Cheng Chen , YingGuo Chen and **Lining Xing**
National University of Defense Technology
Changsha, Hunan Province 410072

Abstract

The satellite data transmission traffic appears a considerable growth tendency with the increase in the number of satellites and the client requirements, so how to solve the large-scale multiobjective satellite data transmission scheduling problem (SDTSP) within a valid period of time has become more and more important. In this context, a multiobjective satellite data transmission model is developed for the practical application, and a novel SVM+NSGA-II algorithm is proposed based on the periodicity of resource confliction in satellite data transmission, the large-scale characteristic of SDTSP and the multidimensional characteristic of the optimization objectives. Experimental results show that NSGA-II has a good performance to solve the multiobjective SDTSP by comparing with the genetic algorithm (GA) , and SVM+NSGA-II can efficiently solve the large-scale multiobjective SDTSP in a very short period of time on the basis of ensuring the satisfactory optimization objectives by comparing with the single NSGA-II.

1 Introduction

Satellite data transmission scheduling is a critical stage in the process of the target information acquisition, transmission, and application. Satellites use sensors to observe targets, and then transmit data to ground stations in their visibility time windows (VTW). In order to downlink the observation data in a timely and effective way and gain more comprehensive profit for both clients and servers, it is extremely significant to study the large-scale satellite data transmission scheduling problem and the corresponding solving methods under the condition of multi-satellite and multi-ground station in the future.

As an important part of satellite range scheduling (SRS), the traditional SDTSP was proved to be NP-Hard (Barbulescu et al. 2004; Vazquez and Erwin 2014). It refers to the allocation of non-conflicting data transmission resource (DTR for short, including transmission time windows and ground station antennas) and the time windows to the satellite data transmission tasks (SDTT), which needs to meet the relevant constraints and the client requirements, and optimize the scheduling objectives. In addition, scheduling daily SDTT under a small set of ground stations is getting more difficult with an increasing number of satellites. Thus, the SDTSP is always oversubscribed, and a number of SDTT are usually unserved (Yuqing et al. 2015).

To solve the contradiction between the limited ground stations and the increasing requirements of satellite, the SDTSP and its variations have been studied for decades. As early as the 1990s, Gooley developed a mixed integer programming model to solve the SRS problem from the US Air Force Satellite Control Network (AFSCN). In 1994, Parish proposed a classical genetic algorithm approach to deal with the SRS problem. In 2002, Barbulescu et al. utilized data from AFSCN to make a comparison of a GA, a simple heuristic and local search methods. Subsequently, Barbulescu, Howe, and Whitley present the evolution of the problem during 10 years at the AFSCN. So far, many simple heuristics methods (2005; 2008; 2011) were applied to the problem, and most successful approaches to highly complex SDTSP problems are those using meta-heuristics (2004; 2011; 2012; 2013; 2015; 2014). Certainly, there appears many other new technologies (2008; 2014; 2014; 2014) to deal with the similar antenna-satellite assignment problems.

However, all above mentioned studies on SDTSP have not considered the periodicity of resource confliction in satellite data transmission, the large-scale characteristic of SDTT and the multidimensional characteristic of the optimization objectives. Firstly, the Earth observation satellites (EOS) run along the fixed orbit, the longitude and latitude of ground stations are usually constant, and the data reception range of antennas remain unchanged generally. When satellites come into the data reception range of antennas, it always appears the characteristics of revisit, which shows the periodic conflict features in contention for DTR between satellites, especially in the satellite clusters.

Subsequently, a large set of requests for satellite data must be scheduled every day. This is becoming a critical problem since the rapid increase in the number of satellites cannot be counterbalanced by installing additional and expensive ground stations. Thus, the general meta-heuristics (such as genetic algorithms, particle swarm optimization and so on) may perform worse to find a satisfactory solution for the large-scale SDTSP due to the high complexity of global search and the feature of time-consuming computation.

In addition, as the size of the problem increases, traditional single objective scheduling may lead to the unbalance of

DTR arrangement and cause a large difference in the completion of SDTT.

To tackle the large-scale multiobjective SDTSP, this paper first takes into account two objectives from both client and server: 1) maximizing the total weighted transmission time of SDTT; 2) maximizing the load balancing degree of DTR, and then develops the corresponding satellite data transmission model. Next, it combines the classical multiobjective evolutionary algorithm NSGA-II with a mature classification approach SVM to solve this problem. Finally, the analysis of computational results proves the high effectiveness of SVM+NSGA-II.

The remainder of this paper is organized as follows. In Section 2, we introduce the SDTSP and develop its mathematical model. In Section 3, SVM+NSGA-II is proposed according to the basic problem characteristics. Section 4 designs the simulation experiments, and Section 5 presents the computational results. In Section 6, we give some conclusions and remarks for further research.

2 Problem Description

The SDTSP is regarded as a combinatorial optimization problem for the allocation of DTR to the tasks in the visibility time windows, which can be divided into two phase decision subproblems: 1) to determine the allocation sequence of DTR for each task; 2) to complete DTR assignment for tasks. The main purpose of the satellite data transmission scheduling is to make a satisfactory data transmission plan that can download more data and make full use of ground station resources. In this section we formulate the SDTSP based on some practical constraints and assumptions.

In general, the SDTSP could be defined as follows:

$$SDTS = \{T, G, S, W, ST, ET\} \tag{1}$$

and the related notations are described as follows.

- $T = \{t_i | 1 \le i \le n_t\}$ is the set of data transmission tasks with $|T| = n_t$. Each task t_i is composed of a seven tuple:

  $$t_i = \{id_{t_i}, s_{t_i}, st_{t_i}, et_{t_i}, dur_{t_i}, p_{t_i}, f_{t_i}\} \tag{2}$$

  where id_{t_i} is the ID of t_i, s_{t_i} is the satellite identifier to which t_i belongs, $[st_{t_i}, et_{t_i}]$is the required transmission time window of t_i, dur_{t_i} is the minimal transmission duration of t_i, p_{t_i} is the priority of t_i, and f_{t_i} is the transmission frequency band of t_i.

- $G = \{g_k | 1 \le k \le n_g\}$ is the set of ground stations with $|G| = n_g$. Each ground station g_k is composed of a three tuple:

  $$g_k = \{id_{g_k}, loc_{g_k}, A_{g_k}\} \tag{3}$$

  where id_{g_k} is the ID of g_k, loc_{g_k} is the coordinates of g_k (including longitude, latitude and altitude), and $A_{g_k} = \{a_{g_k d} | 1 \le d \le d_{g_k}\}$ is the set of ground station antennas belonging to g_k with $|A_{g_k}| = d_{g_k}$. Each antenna $a_{g_k d}$ corresponds to a frequency band $af_{g_k d}$.

- $S = \{s_j | 1 \le j \le n_s\}$ is the set of satellites with $|S| = n_s$. Each satellite s_j is composed of a three tuple:

  $$s_j = \{id_{s_j}, orb_{s_j}, A_{s_j}\} \tag{4}$$

  where id_{s_j} is the ID of s_j, orb_{s_j} is the orbit parameters of s_j, and $A_{s_j} = \{a_{s_j m} | 1 \le m \le m_{s_j}\}$ is the set of satellite antennas belonging to s_j with $|A_{s_j}| = m_{s_j}$. Each antenna $a_{s_j m}$ corresponds to a frequency band $af_{s_j m}$.

- $W = \{w_{jk}^z | 1 \le j \le n_s, 1 \le k \le n_g, 1 \le z \le n_w\}$ is the set of visibility time windows for ground stations and satellites with $|W| = n_w$. Each visibility time window w_{jk}^z is composed of a four tuple:

  $$w_{jk}^z = \{sw_{jk}^z, ew_{jk}^z, s_j, g_k\} \tag{5}$$

  where $[sw_{jk}^z, ew_{jk}^z]$ is the access period of w_{jk}^z, s_j is the corresponding satellite, and g_k is the corresponding ground station.

- $[ST, ET]$ is the scheduling horizon.

In order to develop analyses and research easily, we make the following assumptions:

1) Without consideration of the unexpected accident in the running of satellites and ground stations, all equipment operates normally in the whole scheduling horizon.
2) Without considering the impact from technical aspects such as bit error rate, a task can be transmitted to the ground station if it meets the basic constraints.
3) The shift time of antennas is set to be a constant C_1, and no task can be transmitted during this period.
4) Once the transmission of a task starts, it cannot be preempted.
5) All the tasks are independent with each other.

Based on the above notations and assumptions, the mathematical statement of the SDTSP could be established as follows.

The decision variables of the model are as follows. Let

- $$x_{ijmkd}^z = \begin{cases} 1 & \text{if task } t_i \text{ is successfully scheduled from antenna } a_{s_j m} \text{ to } a_{j_k d} \text{ in the visibility time window } w_{jk}^z \\ 0 & \text{otherwise} \end{cases}$$

- $TW_i = [stw_i, etw_i]$ be the transmission time window of task t_i.

The objective function $F(s)$ contains two optimization objectives $f_1(s)$ and $f_2(s)$ from clients and servers where s is the scheduling scheme, respectively:

$$max\ F(s) = \{f_1(s), f_2(s)\} \tag{6}$$

- $f_1(s)$, the total weighted transmission time of SDTT, is calculated as the sum of the product of the priority and the transmission time of the task in the scheduling scheme:

  $$f_1(s) = \sum_{i=1}^{n_t} x_{ijmkd}^z \cdot (etw_i - stw_i) \cdot p_{t_i} \tag{7}$$

- $f_2(s)$, the load balancing degree of DTR, is calculated as the mean square error of the load in each ground station antenna:

$$f_2(s) = 1 - \{\frac{\sum_{k=1}^{n_g}\sum_{d=1}^{d_{g_k}}[L(a_{g_k d}) - L(\bar{A}_{g_k})]}{\sum_{k=1}^{n_g} d_{g_k}}\}^{\frac{1}{2}} \backslash L(\bar{A}_{g_k}) \tag{8}$$

where

$$L(a_{g_k d}) = \sum_{i=1}^{n_t}\sum_{j=1}^{n_s}\sum_{m=1}^{m_{s_j}} x_{ijmkd}^z \cdot (etw_i - stw_i) \tag{9}$$

$$L(\bar{A}_{g_k}) = \frac{\sum_{k=1}^{n_g}\sum_{d=1}^{d_{g_k}} L(a_{g_k d})}{\sum_{k=1}^{n_g} d_{g_k}} \tag{10}$$

Eq. (6) is subjected to the constraints as follows:

$$\forall i,j,m,k,d,z : st_{t_i} \le stw_i < etw_i \le et_{t_i} \tag{11}$$

$$\begin{gathered}\forall i,j,m,k,d,z : \\ (x_{ijmkd}^z = 1) \cap (sw_{jk}^z \le stw_i < etw_i \le ew_{jk}^z)\end{gathered} \tag{12}$$

$$\begin{gathered}\forall i,j,m,k,d,z : \\ (x_{ijmkd}^z = 1) \cap (af_{s_j m} = f_{t_i}) \cap (af_{g_k d} = f_{t_i})\end{gathered} \tag{13}$$

$$\begin{gathered}\forall j,m,k,d,z; \exists i_1,i_2 (i_1 \ne i_2) : \\ (x_{i_1 jmkd}^z \cdot x_{i_2 jmkd}^z = 0) \cup (TW_{i_1} \cap TW_{i_2} = \emptyset)\end{gathered} \tag{14}$$

$$\begin{gathered}\forall i,j,m,z; \exists k_1,k_2,d_1,d_2((k_1 \ne k_2) \cup (d_1 \ne d_2)) : \\ x_{ijmk_1 d_1}^z + x_{ijmk_2 d_2}^z \le 1\end{gathered} \tag{15}$$

$$\forall i : dur_{t_i} \le etw_i - stw_i \tag{16}$$

$$\forall i : \sum_{j=1}^{n_s}\sum_{m=1}^{m_{s_j}}\sum_{k=1}^{n_g}\sum_{d=1}^{d_{g_k}}\sum_{z=1}^{n_w} x_{ijmkd}^z \le 1 \tag{17}$$

$$\begin{gathered}\forall i_1,i_2,j_1,j_2,m_1,m_2,z_1,z_2,k,d : \\ (x_{i_1 j_1 m_1 kd}^{z_1} \cdot x_{i_2 j_2 m_2 kd}^{z_2} = 1) \cap \\ [max(stw_{i_1}, stw_{i_2}) - min(etw_{i_1}, etw_{i_2}) \ge C_1]\end{gathered} \tag{18}$$

$$\forall i,j,m,k,d,z : x_{ijmkd}^z \in \{0,1\} \tag{19}$$

$$\forall i : stw_i, etw_i \ge 0 \tag{20}$$

Constraint (11) ensures that each task is scheduled within the required transmission window. Constraint (12) guarantees that each task is scheduled within the visibility time windows for ground stations and satellites. Constraint (13) indicates that the frequency bands of both satellite antenna and ground station antenna must be identical. Constraint (14) shows the fact that one ground station antenna could not support two or more tasks simultaneously. Constraint (15) ensures that every free task is assigned to at most one ground station antenna. Constraint (16) guarantees that each transmission time window of a task satisfy the requirement of its minimal transmission duration. Constraint (17) indicates that all the tasks are scheduled at most once. Constraint (18) ensures that the spare time between two tasks must be enough for the shift time of antennas. Constraints (19) and (20) describe the domain of definitions for the decision variables.

3 Methodology

For the characteristics of large scale, high complexity and high cost in the future SDTSP, we introduce the mature SVM (Cortes and Vapnik 1995) algorithm from the field of machine learning to solve the large-scale problem, and the classical NSGA-II (Deb 2000) from the field of intelligent optimization to solve the multiobjective problem. First, through learning from the historical scheduling data, SVM builds a classification model and directly generates the initial scheduling scheme by making predictions for the new tasks. Next, NSGA-II is applied for the tasks unsuccessfully scheduled in the initial scheduling scheme. Finally, the final scheduling scheme is produced.

3.1 NSGA-II

Evolutionary algorithms (EA), which are heuristic search algorithms, have been successfully applied to multiobjective optimization problems,and multiobjective evolutionary algorithms (MOEAs) have become promising areas of research in evolutionary computation (EC). Among MOEAs, the non-dominated sorting genetic algorithm (NSGA) proposed by Srinivas and Deb in 1995 is one of the early dominance-based evolutionary algorithms. In 2000, for overcoming the drawbacks of NSGA, Deb proposed NSGA-II which is regarded as one of the most excellent MOEAs. Consequently, NSGA-II is applied to solve the SDTSP in this section by comprehensively considering the running time, accuracy and the number of objectives.

Given that solving the SDTSP is a dynamic process of dealing with the resource contention conflict for each task, it may provide more available transmission resources for the subsequent tasks by scheduling the tasks of less conflict first, which indicates that reasonable utilization of the heuristic information about conflict is an effective way to tackle the SDTSP. Thus, we first give the definition of possible conflict as follows.

Definition 1 *Let t_i and t_j be two different tasks, if there is an overlap between their required transmission time windows, we say that there exists a possible conflict between t_i and t_j.*

$$PC(t_i,t_j) = \begin{cases} true & \text{if } min(et_{t_i}, et_{t_j}) > max(st_{t_i}, st_{t_j}) \\ false & \text{otherwise} \end{cases}$$

Based on the Def. 1, the process of solving the SDTSP by means of single NSGA-II could be designed for two parts: 1) to determine the task set of possible conflict, which can limit the length of individual encoding in NSGA-II; 2) to determine the allocation sequence of DTR for tasks in the task

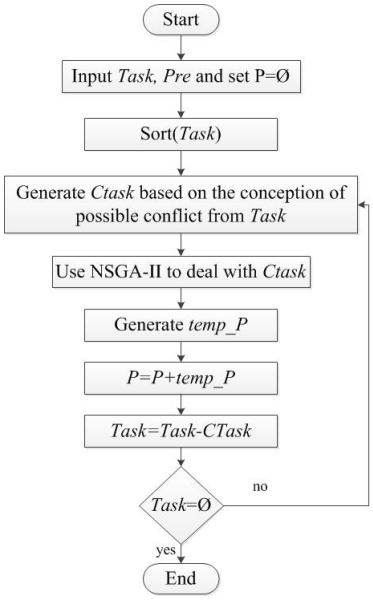


Figure 1: Illustration of the process of solving the SDTSP by means of single NSGA-II

set of possible conflict and then complete the DTR assignment by using NSGA-II. The main steps is given in Fig. 1, where $Task$ is the original set of tasks, Pre is the preference of users, P is the final scheduling scheme, $CTask$ is the current task set of possible conflict, and $temp\_P$ is the current scheduling scheme of the tasks in $CTask$. The function $sort()$ is to sort tasks in ascending order of their earliest start time. Details of the operators in NSGA-II can be seen as follows.

Individual Representation Since the characteristic of two-phase decision in SDTSP, the individual representation of a solution should contain the information of two parts: the allocation sequence of DTR and the assignment of DTR for each task, which can be denoted as resource allocation sequence and resource assignment, respectively.

1. **Resource allocation sequence representation**
   The resource allocation sequence is represented in a random number permutation way, and then the allocation sequence of DTR for tasks is determined by decoding. For each task T_i in $CTask = \{T_1, T_2, \cdots T_n\}$ which has been sorted in ascending order of the earliest start time, a random number r_i in [0,1] is generated. As shown in Fig. 2, the generated set of $\{r_1, r_2, \cdots r_n\}$ is called a random number sequence. During the running process of the algorithm, the task sequence $\{T_1, T_2, \cdots T_n\}$ of $CTask$ remains unchanged, while there exist more than one random number sequences corresponding to this fixed task sequence. Thus, one of the random number sequences can be regarded as the resource allocation sequence representation of one individual, and a random number is interpreted as the value at the corresponding bit.

   Decoding is a mapping procedure that determines the allocation sequence of DTR for tasks in CTask by means of the random number sequence: the larger the random number is, the earlier the DTR is allocated to the corresponding task. When two tasks correspond to the same random number, the DTR is firstly allocated to the front task in the task sequence of $CTask$. For example, the decoding of the representation in Fig. 2 is as follows:

$$T_4 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_5 \tag{21}$$

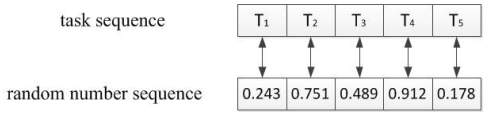


Figure 2: Simple example of the resource allocation sequence representation

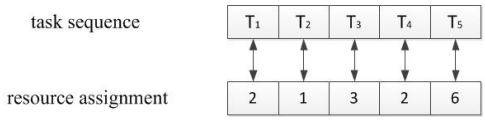


Figure 3: Simple example of the resource assignment representation

2. **Resource assignment representation**
   The resource assignment representation depends on the task sequence of $CTask$. For the value on position k in the resource assignment representation, it indicates the DTR selected to transimit the kth task in the task sequence. For example, as shown in Fig. 2, assume that the tasks $T_1,\ T_2,\ T_3,\ T_4\ and\ T_5$ has 3, 2, 5, 4 and 7 DTRs, respectively. A resource assignment representation is presented in Fig. 3, which represents that the tasks $T_1,\ T_2,\ T_3,\ T_4\ and\ T_5$ select their second, first, third, second and sixth DTR to transmit data, respectively.

Crossover and Mutation According to individual representation, one thing both the resource allocation sequence representation and the resource assignment representation have in common is the randomness of encoding, whereas the difference between them is the expression of the encoding meaning. Therefore, this section still describes the crossover and mutation operators from two parts independently.

1. **Resource allocation sequence operators**
   For the resource allocation sequence representation, the one-point crossover operator is employed. First, for two parent representations, a random point is selected to divide them into two parts, respectively. Secondly, both their left parts are exchanged and generate two offspring representations.

   Then, we apply the one-point mutation operator in our algorithm. First, a random position in the resource allocation sequence representation is selected. Secondly, a new random number in [0,1] is generated to replace the original number.

2. **Resource assignment operators**
   For the resource assignment representation, we employ

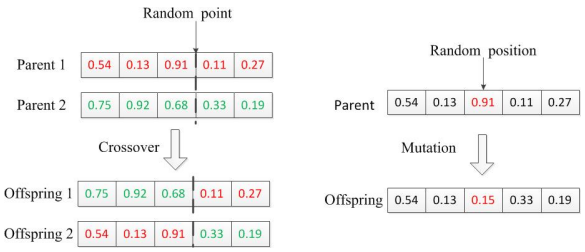


Figure 4: Simple example for illustration of one-point crossover operator and one-point mutation operator

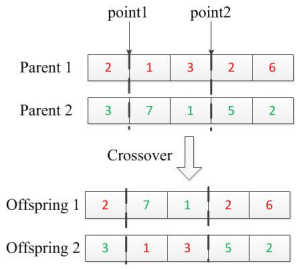


Figure 5: Simple example for illustration of two-point crossover operator

the two-point crossover operator in order to avoid generating unreasonable coding (see Fig. 5). First, two points randomly generated is used to separate two parent representations into three parts, respectively. Secondly, both their middle parts are exchanged and generate two offspring representations.

Then, considering the optimization objective of maximizing the load balancing degree of DTR proposed in Section 2, we design a mutation operator based on the resource load probability of the ground station antennas. The main steps are as follows:

1) Calculate the sum of transmission time for all tasks that have been scheduled on the ground station antenna a_l, which is regarded as the total load time of a_l and is denoted as $time_{a_l}$:

$$time_{a_l} = \sum_{a_{g_k d}=a_l} (TW_i \cdot x^z_{ijmkd}) \tag{22}$$

2) The mutation probability of ground station antenna a_l in the next stage is calculated based on the current total load time of all antennas, and is denoted as $prob_{a_l}$:

$$prob_{a_l} = \begin{cases} 0.99 & \text{if } time_{a_l} = 0 \\ 1 - \frac{time_{a_l}}{\sum time_{a_j}} & \text{otherwise} \end{cases}$$

where $\frac{time_{a_l}}{\sum time_{a_j}}$ is the resource load probability of antenna a_l.

3) The mutation operator is applied for all the positions in the resource assignment representation. And the mutation direction of each position is determined by means of the roulette-wheel method, where the probability of being selected for antenna a_l is denoted as P_{a_l}:

$$P_{a_l} = \frac{prob_{a_l}}{\sum prob_{a_i}} \tag{23}$$

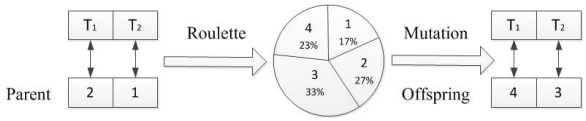


Figure 6: Simple example for illustration of the mutation operator based on the resource load probability of the ground station antennas

As shown in Fig. 6, assume that both tasks T_1 and T_2 can be scheduled on antennas a_1, a_2, a_3 and a_4, and the resource load probabilities of the four antennas are calculated to be 0.5, 0.2, 0 and 0.3, respectively. Thus, the corresponding mutation probabilities of the four antennas are 0.5, 0.8, 0.99 and 0.7, and the corresponding probability of being selected for the four antennas are 0.17, 0.27, 0.33 and 0.23, respectively. Then, Roulette will be undertaken to decide the mutation direction.

Crowding Distance Measure Due to the application of the heuristic information of possible conflict, the solution scale in algorithm is reduced, which limits the length of individual representation. Furthermore, there exists many-to-one mapping from decision space to objective space. Consequently, there are not many nondominated solutions in NSGA-II. In order to keep a good diversity of solutions, we modify the crowding distance as a measure in decision space by means of two rules:

1. Rule 1
   When the number of nondominated solutions obtained by fast nondominated sorting is less than 10, the crowding distance is measured by a function of the times of a resource assignment that occurs in the population. It is simply calculated as follows:

$$C[i]_{distance} = \frac{nPop - T[i]_{GR\_assignment}}{nPop} \tag{24}$$

   where $C[i]_{distance}$ is the crowding distance of individual i, $nPop$ is the size of population, and $T[i]_{GR\_assignment}$ represents the number of individuals in the population which have the same resource assignment with individual i.
2. Rule 2
   In addition, the traditional crowding distance measure (Deb 2000) is employed.

User Preference Design Due to the generation of a set of good compromise solutions by applying NSGA-II, there will appear a series of scheduling schemes. Thus, the selection of one satisfactory scheduling scheme is significant for clients and servers with different requirements. In this section, we design an approach of selecting the scheduling scheme on the basis of user preference:

- First, according to the optimization objectives f_1 and f_2, users give their probabilities of preference $(pre\_prob_1,\ pre\_prob_2)$ such that

$$pre\_prob_1 + pre\_prob_2 = 1 \tag{25}$$

- Then, based on the ideal point of the pareto-optimal set (f_1^*, f_2^*), the preference point of users (f_1^{pre}, f_2^{pre}) is calculated as follows:

$$f_i^{pre} = f_i^* \cdot pre\_prob_i, i = 1, 2 \tag{26}$$

where

$$f_i^* = max\{f_i | f_i \in pareto\text{-}optimal\ set\} \tag{27}$$

- Finally, compute the Euclidean distance between (f_1^{pre}, f_2^{pre}) and all solutions in the pareto-optimal set, and choose the optimal solution with minimum distance to generate the corresponding scheduling scheme.

3.2 SVM+NSGA-II

Support Vector Machine (SVM) is a machine learning algorithm based on statistical learning theory and stems from Vapnik's theory which avoids estimating probabilities on finite data (Vapnik 1995). It can automatically seek those support vectors with better classification performance, obtain a better separation between different categories, and make the generalization error of the classifier lower.

Generation of the Initial Scheduling Scheme Although SVM was originally used to separate the two classes of data, each transmission task is faced with more than two antennas in the SDTSP, which means that the multiclass SVM is required. Therefore, we introduce the classical multiclass SVM based on one-verses-one approach (Hsu and Lin 2002). Then, the libsvm toolbox developed by Chang and Lin is employed to solve the satellite data transmission classification problem to generate an initial scheduling scheme. The running process of the SVM is shown in Fig. 7, where q represents the users demanding accuracy of prediction model, and p represents the actual prediction accuracy of the test data.

The historical data contained in SVM consists of two parts: 1) the task set data T; 2) the corresponding scheduling scheme data P. As shown in Table 1, data features of each task are selected as follows: the satellite identifier s, the earliest start time st, the latest end time et, the minimal transmission duration dur, the priority p, the transmission frequency band f and the number of transmission time windows n. The data label of each task is the ground station antenna identifier ant assigned to it in the scheduling scheme.

Generation of the Final Scheduling Scheme Since the initial scheduling scheme obtained by running the SVM directly assigns the ground station antennas to each new task, it does not consider that 1) some tasks may conflict with each other and 2) some tasks can not be scheduled because of the mismatching between the prediction result and the DTRs. For example, a task T_1 has three DTRs that contain the ground station antennas $a_1,\ a_4$ and a_9 respectively, whereas the prediction result for T_1 is a_8, which will make

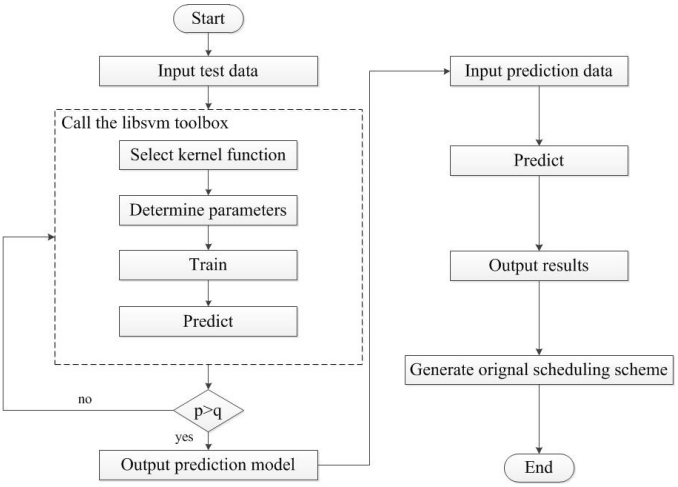


Figure 7: Illustration of the process of generating initial scheduling scheme by means of SVM

T_1 unscheduled. Thus, NSGA-II described in Section 3.1 is applied to produce the final scheduling scheme by dealing with the remaining tasks that fails to be scheduled.

4 Design of the Experiments

In order to demonstrate the validity of SVM+NSGA-II, we first configure the interface between STK and MATLAB by the STK/Connect module to realize the programming control of STK by MATLAB. Then, the STK is employed to establish the simulation scenarios. Finally, the test instances is generated based on the scenario forecast flow data from STK.

4.1 Design of Scenarios

In the simulation scenarios designed by STK, five ground stations in China participated in the scheduling, and each ground station is equipped with two S-band antennas. We designed ten sun-synchronous orbits with six satellites uniformly distributed on each orbit. The altitude difference between two neighboring orbits is 30km, and the minimum orbit altitude is 370 km. All satellites are equipped with an S-band antenna.

This section designed six scenarios, including SCEN_1 with 5 stations and 10 satellites, SCEN_2 with 5 stations and 20 satellites, SCEN_3 with 5 stations and 30 satellites, SCEN_4 with 5 stations and 40 satellites, SCEN_5 with 5 stations and 50 satellites, SCEN_6 with 5 stations and 60 satellites. The locations of the ground stations in each scenario are identical, while the satellites in each scenario consist of 1 satellite, 2 satellites, 3 satellites, 4 satellites, 5 satellites and 6 satellites on each orbit, respectively. The planning horizon is set to be 24h from 2017/06/01 00:00:00 to 2017/06/02 00:00:00 for all scenarios.

4.2 Generation of Test Instances

First, the basic properties of the transmission tasks are designed as follows.

Table 1: Computational results of performance metrics

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	label
data	s	st	et	dur	p	f	n	ant

1) The sum of the number of transmission tasks belonging to each satellite can not be more than the number of non-overlapping visibility time windows between this satellite and all the ground stations.
2) The priority of each task is randomly generated from 1 to 10, where 10 represents the highest priority.
3) The required transmission time window of each task must be inside the planning horizon.
4) If the duration of the required transmission time window of a task is less than 7min, the minimal transmission duration of this task will be the same as the length of the required transmission time window; otherwise, the minimal transmission duration of this task is set to be 7min.
5) The transmission frequency band of all tasks is set to be S-band.
6) All tasks can not be divided, that is, each task will be scheduled once at most in the planning horizon.
7) The shift time of all antennas is set to be 1min.

Then, on the basis of above task properties, the generation algorithm of test instances is described in Alg. 1.

Algorithm 1 Generation of test instances

Input: S: the satellite set; W: the visible time window set
Output: T: the task set

for s_j *in* S
  $new\_W \leftarrow sort(W(s_j))$
  $new\_W^* \leftarrow merge(new\_W)$
  for w_i *in* new_W^*
    if $dur(w_i) \geq 420$ ***then*** $q \leftarrow random(0,1)$
      if $q < 0.3$ ***then***
        repeat
        $[st_{t_i}, et_{t_i}] \leftarrow real\_generate(w_i)$
        until $et_{t_i} - st_{t_i} < 420$
        $dur_{t_i} = et_{t_i} - st_{t_i}$
      else
        $[st_{t_i}, et_{t_i}] \leftarrow store\_generate(w_i)$
        $dur_{t_i} = 420$
      Endif
    else
    $[st_{t_i}, et_{t_i}] \leftarrow real\_generate(w_i)$
    $dur_{t_i} = et_{t_i} - st_{t_i}$
    Endif
    $p_{t_i} \leftarrow int\_random(1,10)$
  Endfor
Endfor
$T \leftarrow T + t_i$

In Alg.1, the function $sort(W(s_j))$ is to sort all the visibility time windows of s_j in ascending order of their earliest start time. The function $merge(new\_W)$ is to merge the mutually overlapping visibility time windows into the new time windows which are non-overlapping with each other. The function $real\_generate(w_i)$ is to generate the required transmission time window of the task by randomly selecting two time points in the visibility time window w_i as the earliest start time and the latest end time. The function $store\_generate(w_i)$ represents that the required transmission time window of the task is the same as the visibility time window w_i. The function $int\_random(1,10)$ is to produce a random integer in [1,10] to be the task priority.

Table 2: Statistical results of test instances for each scenario

	SCEN_1	SCEN_2	SCEN_3	SCEN_4	SCEN_5	SCEN_6
scale	5×10	5×20	5×30	5×40	5×50	5×60
num_Avg	87	176	275	369	452	546

The statistical results of the average number of tasks (num_Avg) in each scenario are indicated in Table 2.

5 Computational Results and Analysis

This section evaluates SVM+NSGA-II by comparing with GA and NSGA-II. We generated 20 groups of different test instances for each scenario by the way described in Section 4.2. All the test instances run 10 times, and the average was taken as the final value for each test instance. In each scenario, the task set of the former ten test instances and their corresponding solutions obtained by employing NSGA-II will act as the training data set, and the tasks of the latter ten test instances will act as test data set to compare the performance of different algorithms.

We implemented our algorithms in MATLAB R2013a and tested them on a PC with Intel Core i5-3210 CPU (2.50 GHz).

5.1 Experimental Configurations

In the experiments, the parameters of the GA are set as: the generational gap $Gap = 0.9$, which indicates that 90% of the individuals in the current population are retained to the next generation and the remaining 10% will be supplied in another way. The probabilities of crossover and mutation are chosen to be $Pc = 0.8$ and $Pm = 0.1$, respectively. The objective function $f_1(s)$, the total weighted transmission time of SDTT, is employed as the fitness function of the GA. The other operators, including individual representation, crossover and mutation, are the same as those of NSGA-II.

The parameters of NSGA-II determined by means of the pilot experiments are set as: crossover percentage=0.8, crossover probability=0.8, mutation percentage=0.4, mutation probability=0.1. For the convenience of conducting experiments, we decide to set the user preference (pre_prob_1, pre_prob_2) to be (0.5, 0.5).

Table 3: The size of population and the number of generations in NSGA-II

	nP	nG
$\|CTask\| \leq 3$	10	20
$3 < \|CTask\| \leq 6$	20	40
$6 < \|CTask\| \leq 9$	30	60
$9 < \|CTask\|$	40	80

In order to further improve the efficiency of the algorithms, the size of population nP and the number of generations nG are designed based on the number of the elements contained in the task set of possible conflict $|CTask|$ (shown in Table 3).

5.2 Evaluation Metrics

Statistics based on the execution time, the objective function values, and the number of unscheduled tasks for each test instance are employed to evaluate the performance of the algorithms.

The experimental results are indicated in Table 4, whose columns of are as follows (from left to right): the identifier of scenarios, the average execution time from the latter ten groups of test instances in each scenario (T_Avg), the average objective function values from the latter ten groups of test instances in each scenario (Obj_Avg), and the average number of unscheduled tasks from the latter ten groups of test instances in each scenario ($unST\_Avg$).

From the results cast in Table 4, we can find that NSGA-II performs well to solve the multiobjective SDTSP by comparing with the GA, and SVM is an efficient technology to solve the large-scale scheduling problem within a period of limited time by comparing NSGA-II and SVM+NSGA-II. Thus, it can initially proves that the performance of the combination of SVM and NSGA-II is better.

In order to intuitively and clearly verify the performance of the algorithms, we made the corresponding statistical graphs for the results in Table 4.

5.3 Analysis of Multiobjective

As shown in Fig. 8, the objective values of the GA in $f_1(s)$ is a little larger than that of NSGA-II in all scenarios, while its value of $f_2(s)$ yielded through the second objective function is much smaller than that of NSGA-II. In Fig. 9, we can find that there is little difference in the execution time between NSGA-II and the GA. And from Fig. 10, it shows that the number of unscheduled tasks of NSGA-II in all scenarios is much less than that of the GA. Therefore, it proves that NSGA-II performs well to deal with the multiobjective SDTSP.

5.4 Analysis of Large Scale

Fig. 8 shows that the objective function values of NSGA-II and SVM+NSGA-II just have a little difference that the total weighted transmission time of SDTT $f_1(s)$ in SVM+NSGA-II is a little more than that in NSGA-II in all

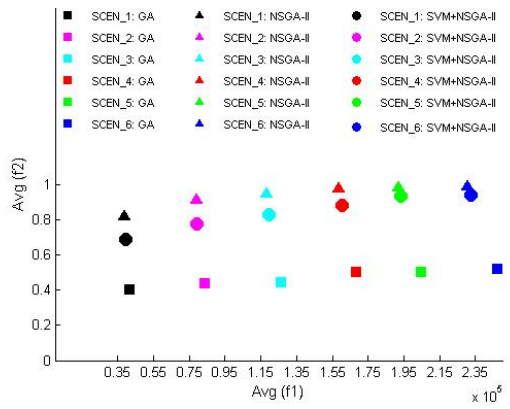


Figure 8: The Obj_Avg values with GA, NSGA-II and SVM+NSGA-II

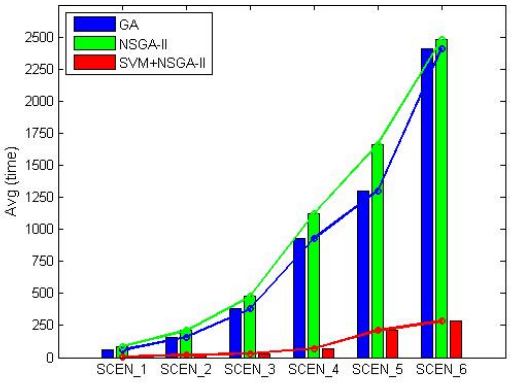


Figure 9: The T_Avg (second) values with GA, NSGA-II and SVM+NSGA-II

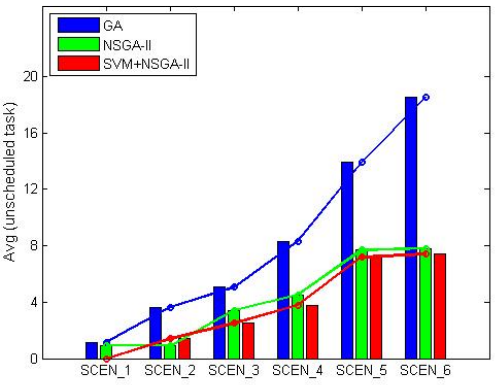


Figure 10: The $unST\_Avg$ values with GA, NSGA-II and SVM+NSGA-II

scenarios, whereas the load balancing degree of DTR $f_2(s)$ of SVM+NSGA-II is less than that of NSGA-II in all scenar-

Table 4: Computational results of performance metrics

Scenario	T_Avg (second)			Obj_Avg			$unST\_Avg$		
	GA	NSGA-II	SVM+NSGA-II	GA	NSGA-II	SVM+NSGA-II	GA	NSGA-II	SVM+NSGA-II
SCEN_1	56.976	85.412	5.728	(41806.541, 0.403)	(38643.509, 0.816)	(39384.253, 0.688)	1.2	0.9	0
SCEN_2	157.405	212.253	18.381	(83955.616, 0.436)	(79214.814, 0.909)	(79480.833, 0.772)	3.7	0.9	1.4
SCEN_3	380.647	476.119	31.108	(126414.318, 0.441)	(118516.757, 0.945)	(119464.044, 0.826)	5.1	3.4	2.5
SCEN_4	927.464	1119.623	69.093	(168724.203, 0.501)	(158584.585, 0.972)	(160173.671, 0.878)	8.3	4.5	3.8
SCEN_5	1298.919	1661.753	210.935	(204983.337, 0.501)	(191954.621, 0.977)	(193146.216, 0.932)	13.9	7.7	7.2
SCEN_6	2411.278	2476.617	284.180	(247180.303, 0.519)	(230690.426, 0.984)	(232299.949, 0.940)	18.5	7.8	7.4

ios. One reason is that the search of all solutions in NSGA-II is guided by the heuristic information from the process of mutation of the resource assignment representation for each individual.

From Fig. 9, we can see that SVM+NSGA-II exhausted much less time than NSGA-II in all scenarios. In particular, with the expansion of the problem scale, the growth of execution time of NSGA-II is much faster than that of SVM+NSGA-II.

Simultaneously, Fig. 10 indicates that the number of unscheduled tasks of NSGA-II in any other scenarios is a little more than that of SVM+NSGA-II except in Scen_2.

5.5 Conclusion

From the above experimental results we can see that SVM+NSGA-II is an excellent technology with high efficiency to solve the large-scale multiobjective satellite data transmission scheduling problem. Because it can yield a satisfactory scheduling scheme in a very short period of time on the basis of ensuring the good optimization objectives.

Theoretically, the better the prediction effect of SVM is, the less the execution time of SVM+NSGA-II is. Fig. 11 with error bars was generated by comparing the final scheduling scheme obtained from NSGA-II with that from SVM+NSGA-II in the same test instances. It shows that the average prediction accuracy for each scenario does not exceed 50%, and the standard deviation of prediction accuracy in the small-scale scenarios such as SCEN_1 and SCEN_2 is larger. Therefore, it is worthwhile to consider how to improve the prediction accuracy and the prediction stability of SVM in the future research.

6 Summary and Prospects

Satellite data transmission scheduling problem is a complex combinatorial optimization problem with highly complex and over-constrained characteristics. In this paper, we combined the SVM in the field of machine learning and the NSGA-II in the field of intelligent optimization to design an efficient technology for solving the large-scale multiobjective SDTSP. Experimental results show that this technology can greatly shorten the execution time on the basis of ensuring the satisfactory optimization objectives in all simulation experiments.

Of course, there exist many kinds of practical problems in the field of satellite range scheduling, and we just choose the SDTSP to conduct the research. It is worth mentioning

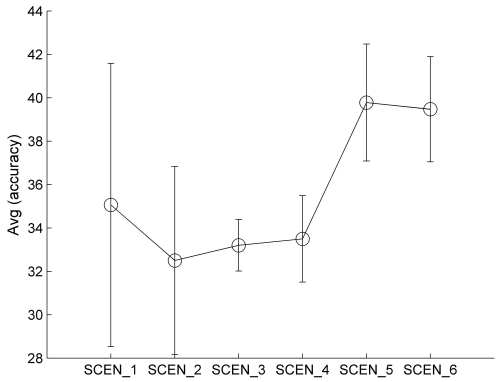


Figure 11: The average prediction accuracy and the standard deviation of prediction accuracy for SVM

that this paper is an effective try to integrate the classical technologies of intelligent optimization and machine learning, which provides an efficient idea to deal with the similar problems for researchers.

In our future work, we will continue to research the way of hybridization of different methods from different fields, and then further improve our solving technology to expand its applicability to more kinds of problems.

7 Acknowledgments

This work is supported by the National Natural Science Fund for Distinguished Young Scholars of China (61525304), National Natural Science Foundation of China (No. 71331008, 61473301, 71501180, 71501179 and 61603400).

References

Barbulescu, L.; Howe, A. E.; Watson, J. P.; and Whitley, L. D. 2002. Satellite range scheduling: A comparison of genetic, heuristic and local search. In *International Conference on Parallel Problem Solving From Nature*, 611–620.

Barbulescu, L.; Watson, J. P.; Whitley, L. D.; and Howe, A. E. 2004. Scheduling spaceground communications for the air force satellite control network. *Journal of Scheduling* 7(1):7–34.

Barbulescu, L.; Howe, A.; and Whitley, D. 2006. *AFSC-N scheduling: How the problem and solution have evolved*. Elsevier Science Publishers B. V.

Chang, C. C., and Lin, C. J. 2011. *LIBSVM: A library for support vector machines*. ACM.

Chen, H.; Zhong, Z.; Wu, J.; and Jing, N. 2015. Multi-satellite data downlink resource scheduling algorithm for incremental observation tasks based on evolutionary computation. In *Seventh International Conference on Advanced Computational Intelligence*, 251–256.

Clement, B. J., and Johnston, M. D. 2005. The deep space network scheduling problem. In *The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, Usa*, 1514–1520.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. In *Machine Learning*, 273–297.

Deb, K. 2000. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation* 6(2):182–197.

Fei, J.; Wang, J.; Jun, L. I.; Hao, C.; and Ning, J. 2011. A new scheduling method for multi-satellite data transmission based on squeaky-wheel optimization. *Journal of Astronautics* 32(4):863–870.

Gooley, T. D. 1993. Automating the satellite range scheduling process. *Masters Thesis Air Force Institute of Technology*.

Hsu, C. W., and Lin, C. J. 2002. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks* 13(4):1026.

Karapetyan, D.; Minic, S. M.; Malladi, K. T.; and Punnen, A. P. 2012. Satellite downlink scheduling problem: A case study . *Omega* 53:115–123.

Lee, S.; Jung, W. C.; and Kim, J. 2008. Task scheduling algorithm for the communication, ocean, and meteorological satellite. *ETRI Journal,30,1(2008-02-11)* 30(1):1–12.

Li, J.; Li, J.; Chen, H.; and Jing, N. 2014. A data transmission scheduling algorithm for rapid-response earth-observing operations. *Chinese Journal of Aeronautics* 27(2):349–364.

Li, L. X.; Ma, W. Z.; and Liu, X. L. 2014. Research on tsga algorithm satellite data transmission scheduling. In *International Conference on Management Science & Engineering*, 56–61.

Marinelli, F.; Nocella, S.; Rossi, F.; and Smriglio, S. 2011. A lagrangian heuristic for satellite range scheduling with resource constraints. *Computers & Operations Research* 38(11):1572–1583.

Parish, D. A. 1994. A genetic algorithm approach to automating satellite range scheduling. *Masters Thesis Air Force Institute of Technology*.

Srinivas, N., and Deb, K. 1994. *Muiltiobjective optimization using nondominated sorting in genetic algorithms*. MIT Press.

Vapnik, V. 1995. The nature of statistical learning theory. 988 – 999.

Vazquez, A. J., and Erwin, R. S. 2014. On the tractability of satellite range scheduling. *Optimization Letters* 9(2):1–17.

Vazquez, R.; Perea, F.; and Vioque, J. G. 2014. Resolution of an antennasatellite assignment problem by means of integer linear programming. *Aerospace Science & Technology* 39:567–574.

Xhafa, F.; Herrero, X.; Barolli, A.; and Takizawa, M. 2013. A simulated annealing algorithm for ground station scheduling problem. In *International Conference on Network-Based Information Systems*, 24–30.

Yuqing; Wang; Rixin; and Minqiang. 2015. Satellite range scheduling with the priority constraint: An improved genetic algorithm using a station id encoding method. *Chinese Journal of Aeronautics* 28(3):789–803.

Zufferey, N.; Amstutz, P.; and Giaccari, P. 2008. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling* 11(4):263–277.

A Constraint-based Mission Planning Approach for Reconfigurable Multi-Robot Systems

Thomas M. Roehr
DFKI GmbH Robotics Innovation Center, Bremen, Germany
thomas.roehr@dfki.de

Abstract

The application of reconfigurable multi-robot systems introduces additional degrees of freedom to design robotic missions compared to classical multi-robot systems. To allow for autonomous operation of such systems, planning approaches have to be investigated that can not only cope with the combinatorial challenge arising from the increased flexibility of combining systems, but also exploit this flexibility to improve for example the safety of operation. While the problem originates from the domain of robotics it is of general nature and significantly intersects with operations research. This paper suggests a constraint-based mission planning approach, and presents our revised definitions for reconfigurable multi-robot systems including the representation of the planning problem using spatially and temporally qualified resource constraints. Planning is performed using a multi-stage approach, and a combined use of knowledge-based reasoning, constraint-based programming and integer linear programming. The paper concludes with the illustration of the solution of a planned example mission.

1 Introduction

Flexibility is the primary feature of reconfigurable multi-robot systems, since their modularity adds an additional degree of freedom to design robotic operations compared to the application of traditional multi-robot systems. For that reason Dignum et al. (Dignum 2009) discuss the so-called strategic flexibility, which offers an exploitation of proactive and reactive adjustment in the context of reconfigurable organizations. The strategic flexibility allows to tackle a set of unforeseen tasks with a robustly equipped system that allows recovery from malfunction thanks to increased redundancy. Exploiting strategic flexibility provides a strong motivation to combine increasingly capable autonomous robotic systems with a concept for modularity.

The main benefit of reconfigurable multi-robot systems lies in the fact that resources can easily, although not arbitrarily, be (re)used by any agent being a member of the reconfigurable system. Using this flexibility allows to balance resource usage and hence to adapt dynamically to operational demands. While modularity can lead to significant operational advantages it has drawbacks: if the level of modularity is chosen arbitrarily high this can lead to less capable systems. One can observe the effect in swarm-based systems, which come with a high degree of modularity: a swarm typically consists of cheap agents with a simple design, and thus , apart from emergent high-level behaviors, these agents come with a rather limited applicability by design. Although the mentioned emergent behaviors can be exploited, these behaviours remain harder to control or will be focused on a single task only. In general, reconfigurable multi-robot systems offer a feasible solution which consists of a mix of individually capable agents, including swarm-like units that can augment the overall robotic team. This augmentation is done either by acting as a fully autonomous agent or as an extension unit. Roehr et al. (Roehr, Cordes, and Kirchner 2014) implement this idea in the context of robotic space exploration missions in order to show the general feasibility and identify critical limitations: the implemented approach validates the potential for increasing the flexibility in future robotic missions, but it also comes with increased operational demands. Thus, they suggest the introduction of a dedicated system model in order to automate operation of reconfigurable multi-robot systems and exploit the offered system capabilities to improve not only efficiency, but also safety of future robotic missions. Roehr and Kirchner (Roehr and Kirchner 2016) show how planning as essential element for automated operation for such a reconfigurable multi-robot system can be approached.

This paper details the problem definition and presents the results of the continued development of the planning approach. In Section 2 we briefly outline relevant background references for the state of the art. Section 3 introduces the planning problem, and in Section 4 we give details on the organization model and its extended use. Section 5 outlines the revised planning approach. We close with a conclusion and outlook in Section 6.

2 Background

The initial motivation for the planning problem is given by Sonsalla et al. (Sonsalla et al. 2014), where a reconfigurable multi-robot system shall establish a logistics chain operation in order to support sample-return missions as part of extraterrestrial exploration. The robotic team consists of mobile and immobile agents, which can be physically connected via a set of electro-mechanical interfaces. By connecting one or more robots, they can form a new

type of agent, comprising features none of the individual agents offers. The ability for reconfiguration offers novel ways of dealing with robotic missions, but Roehr and Kirchner (Roehr and Kirchner 2016) is to our best knowledge the only approach particularly dealing with reconfigurability. This mission planning problem can be understood as a logistic planning problem where mobile robots can transport other immobile and mobile robots. Hence, it is closely related to the Vehicle Routing Problem (VRP) (Toth and Vigo 2014): a fleet of (most often homogeneous) mobile vehicles shall serve a set of customers, e.g., by delivering and/or picking up items, while minimizing a cost function – typically the overall travelled distance. The VRP applies to transportation and logistics scenarios and comes in many variants among which Capacitated VRP (CVRP), VRP with Time Windows (VRPTW) and VRP with Pick-up and Delivery (VRPPD) are the most popular ones. The pickup-and delivery problem can be further distinguished into a many-to-many (M-M), one-to-many-to-one (1-M-1), and one-to-one (1-1) problems, where the notation can be read as cardinalities for the origin, transition point, and target of a commodity, i.e. from-to or from-via-to. The M-M variant for example accounts for multiple commodity (good) origins and destinations, while the 1-M-1 variants assume a start and end of all vehicles at a single depot. The majority of these approaches are either focusing on a single-commodity case, homogeneous vehicle capacities or optimization of routing cost, were our approach has to deal with multi-commodities, heterogeneous vehicles, multi-depots, and fleet size optimization. Hence, a closer relation can be established to more specialized VRP approaches, e.g., such as the Heterogeneous or mixed Fleet VRP (HFVRP) (Baldacci, Battarra, and Vigo 2008) which accounts for a heterogeneous fleet and optionally with unlimited vehicle availability, or Dondo et al. (Dondo and Cerdá 2007) who approach Multi-depot heterogeneous fleet VRP with time windows (MDHFVRPTW). The variant VRP with Trailers and Transshipments (VRPTT) and more generalized VRP with multiple synchronization constraints (VRPMSs) (Drexl 2013) adds synchronization constraints between vehicles, which form a special instance of a reconfigurable multi-agent system containing agents or in this case vehicles of different categories: autonomous and non-autonomous, as well as support and task vehicles. Drexl (Drexl 2013) formulates a graph-based modelling approach to account for the inter-dependence of vehicles. He does not, however, provide an implementation of a solution approach.

While much of the research in VRP originates from the area of operational research, Coltin and Veloso (Coltin and Veloso 2014a; 2014b; 2014c) investigate a pick-up and delivery variant in the context of multi-robot systems and also apply their approach to a taxi problem with ridesharing. They implement optimal approaches as well as metaheuristics, in particular simulated annealing, and Very Large Neighborhood Search (VLNS) (Ahuja, Orlin, and Sharma 2000) their application of VLNS results not only in a scalable approach, but also proves a general benefit of using transfers in a pickup and delivery scenario. In Section 3 we will outline the distinction between existing VRP and our approach, and provide additional constraints for our mission planning problem.

3 Mission Representation

The planning problem presented in the following aims to solve the problem of planning and scheduling a mission performed by a reconfigurable multi-robot system. While a mission can initially be seen as a task assignment for a multi-robot system, here it comes with an essential difference: agents are able to dynamically form physical coalitions referred to as composite agents. These composite agents are formed for three main reasons. Firstly, to perform agent transport: one carrier agent attaches one or multiple (most likely, but not necessarily) immobile systems. Secondly, to provide functionality: some functionality is only available as so-called super-additive effect and requires two or more agents to join so that this functionality becomes available only for this composite agent, but not for the individual agents. Thirdly, to increase the functional redundancy: for agents that are assigned to fulfil requirements, we assume that adding relevant resources improves the redundancy and safety of operation, and effectively the likelihood of a successful performance of an agent.

While most VRP assume homogeneous agents, the team of agents in a reconfigurable multi-robot system is formed by heterogeneous agents; agents with individual capabilities and functionalities, as well as limitations to reconfigure and constraining attributes such as an overall transport capacity. We will look at a mission as a particular (minimal) partitioning problem of an agent team to achieve a requested agent- and function-distribution over space and time.

Definitions & Assumptions

In the following we introduce the basic notation, definitions and assumptions regarding reconfigurable multi-robot systems. The provided definitions describe a modular multi-agent system, which can form composite agents from a set of available agents:

Definition 3.1. *An **atomic agent** a represents a monolithic physical robotic system, where $A = \{a_1, \ldots, |A|\}$, is the set of all atomic agents, and $a \in A$ or equivalently $\{a\} \subseteq A$.*

Definition 3.2. *A mechanically coupled system of two or more atomic agents is denoted a **composite agent** CA, where $CA \subseteq A$, and $|CA| > 1$.*

Definition 3.3. *The type of an atomic agent a is denoted $\widehat{a}$ and equivalently for a composite agent CA the type is denoted $\widehat{CA}$. The set of all agent types is denoted $\theta(A) = \{1, \ldots, |\theta(A)|\}$, with the corresponding type-partitioned sets $A^1, \ldots, A^{|\theta(A)|}$, where $A = A^1 \cup \cdots \cup A^{|\theta(A)|}$.*

Definition 3.4. *A (general) agent is denoted GA, where $GA \subseteq A$, and $GA \neq \emptyset$. A (general) agent represents the wrapping concept for atomic and composite agents, with the corresponding type-partitioned sets $GA^1, \ldots, GA^{|\theta(A)|}$, where $GA = GA^1 \cup \cdots \cup GA^{|\theta(A)|}$*

Definition 3.5. *A (general) agent type $\widehat{GA}$ will be represented as a function $\gamma_{\widehat{GA}} : \theta(A) \rightarrow \mathbb{N}_0$, which maps an*

atomic agent type $\widehat{a}$ to the cardinality $c_{\widehat{a}}$ of the type partition, such that $c_{\widehat{a}} = |GA^{\widehat{a}}|$. The set of all constructible general agent types from a set of atomic agents A is denoted $\theta(\widehat{A})$; it represents the collection of all general agent types that are found in the powerset of all agents $\mathcal{P}^A$.

Note, that a general agent type can equivalently be represented as tuple set of agent type and type cardinality: $\widehat{GA} = \{(\widehat{a}_1, c_1), \ldots, (\widehat{a}_n, c_n)\}$, where $a_i \in A$ and $c_i = |GA^i|$. $\widehat{GA} \supseteq \widehat{GA'} \iff \forall (a_i, c_i) \in \widehat{GA}, (\widehat{a}_i, c'_i) \in \widehat{GA'} : c_i \geq c'_i$, where $i = 1 \ldots |A|$.

Definition 3.6. *A set of atomic agents A is denoted an **agent pool** and it can be represented by a general agent type $\widehat{GA}$, such that $\forall a \in A : \gamma_{\widehat{GA}}(\widehat{a}) = |A^{\widehat{a}}|$.*

Definition 3.7. *An **atomic agent role** $r^{\widehat{a}}$ represents an anonymous agent instance of an atomic agent type $\widehat{a}$.*

Definition 3.8. *A coalition structure of an agent set A is denoted CS^A and is represented by a set of disjunct general agents $CS^A = \{GA_0, \ldots, GA_n\}$, where $GA_0 \cup \cdots \cup GA_n = A$, and $\forall i, j \wedge i \neq j : GA_i \cap GA_j = \emptyset$.*

Assumptions

Our design of the organization model and planning system for a reconfigurable multi-robot system, which both will be detailed in the following section, is based on a set of assumptions to simplify the modelling approach.

Assumption 3.1. *Each atomic and composite agent can be mapped to a single agent type only.*

A reconfigurable multi-robot system requires coupling interfaces, e.g., an electro-mechanical interface (Dettmann et al. 2011), to create physical linking between atomic agents to establish a composite system. Although multiple links could be considered between any two agents, interfaces cannot be arbitrarily coupled and the following assumption holds:

Assumption 3.2. *A mechanical coupling between two atomic agents can only be established through a single link and two and only two compatible physical coupling interfaces.*

Mission specification

The mission specification is a temporal database description, and it defines the initial, intermediate and goal state for a reconfigurable multi-robot system. A valid mission specification is described by the following two definitions:

Definition 3.9. *A spatio-temporal requirement is a spatio-temporally qualified expression (stqe) s which describes the functional requirements and agent instance requirements for a given time-interval and a particular location: $s = (\mathcal{F}, \widehat{GA}_r)@(l, [t_s, t_e])$, where $\mathcal{F}$ is a set of functionality constants, $\widehat{GA}_r$ is the general agent type representing the required atomic agent type cardinalities, $l \in L$ is a location variable, and $t_s, t_e \in T$ are temporal variables describing a temporal interval with the implicit constraint $t_s < t_e$. Variables associated with s will also be referred to using the following notation: $\mathcal{F}^s$,$\widehat{GA}^s_r$,l^s,t^s_s, and t^s_e.*

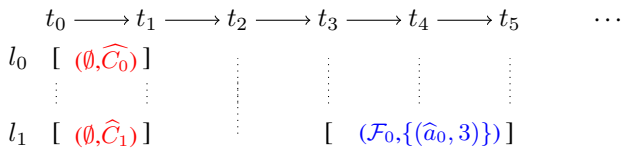


Figure 1: A mission specification example based on a space-time representation

Definition 3.10. *The robotic mission is a tuple $\mathcal{M} = \langle \widehat{GA}, STR, \mathcal{X}, \mathcal{OM} \rangle$, where the agent pool $\widehat{GA}$ describes the available set of agents, STR is a set of spatio-temporally qualified expressions, $\mathcal{X}$ is a set of constraints, and $\mathcal{OM}$ represents the organization model.*

The initial state is defined by the earliest timepoint and binds available agents to their starting depot. The earliest timepoint is $t_0 \in T$ and $\forall t \in T, t \neq t_0 : t > t_0$. Figure 1 illustrates a mission specification, where

$$\widehat{GA} = \{(\widehat{a}_0, 3), (\widehat{a}_1, 2)\},$$
$$STR = \{(\emptyset, \widehat{C_0})@(l_0, [t_0, t_1]), (\emptyset, \widehat{C_1})@(l_1, [t_0, t_1]), (\mathcal{F}_0, \{(\widehat{a}_0, 3)\})@(l_1, [t_3, t_5])\}$$
$$\mathcal{X} = \{t_0 < t_1, \ldots, t_4 < t_5\}$$
$$\mathcal{OM} = \{mobile(\widehat{a}_0), \neg mobile(\widehat{a}_1), \ldots\}$$

$\widehat{C}_0 = \{(\widehat{a}_0, 2), (\widehat{a}_1, 1)\}$, $\widehat{C}_1 = \{(\widehat{a}_0, 1), (\widehat{a}_1, 1)\}$, l_0, l_1 are location variables and $t_0, \ldots, t_5$ are timepoint variables. Two general agents $\widehat{C}_0$ and $\widehat{C}_1$ are assigned to location l_0 and l_1 respectively. Two stqes related to the interval $[t_0, t_1]$ define the initial agent assignments; no functional requirements are part of the initial state description. The goal state is defined over the interval $[t_3, t_5]$ and requires a functionality set $\mathcal{F}_0$ in combination of least 3 agents of type $\widehat{a}_0$ at location l_1.

Mission constraints

A mission can be detailed by constraints in the constraint set $\mathcal{X}$. The only initially required constraints are temporal ones to describe the starting state, e.g., in the presented example all stqes relating to a start at t_0, e.g., cardinality constraints allow to set upper and lower bounds on the usage of agents and functionalities to reduce the combinatorial challenge. Other optional constraints can be added to detail and constrain the evolution of a mission. The following list describes the available constraint types; minimum constraints come with a corresponding max constraint implementation:

temporal qualitative timepoints describe time intervals, where timepoint constraints are provided using point algebra ($<, >, =$) (Dechter 2003).

duration $minDuration(s, t)$, $s \in STR$: sets a lower bound of time t for the duration of the time interval associated with the stqe s.

min cardinality $minCard(s, \widehat{a}, c_{min})$, $s \in STR$: represents a minimum cardinality constraint so that $|GA^{\widehat{a}}| \geq c_{min}$

all distinct $allDistinct(S, \widehat{a})$ describes the constraint: $\forall s \in S : \bigcap A^{\widehat{a},s} = \emptyset$, where $S \subseteq STR$, and $A^{\widehat{a},s}$ represents the subset of agents of type $\widehat{a}$ which are associated with the stqe s.

min distinct $minDistinct(S, \widehat{a}, n)$ describes the constraint: $\forall s_i, s_j \in S, i \neq j : \left||A^{\widehat{a},s_i}| - |A^{\widehat{a},s_j}|\right| \geq n$, where $n \geq 0$, $S \subseteq STR$, and $A^{\widehat{a},s}$ represents the partition of A which contains only agents of type $\widehat{a}$ which are associated with the stqe s.

all equal $allEqual(S, A_e)$ describes the constraint: $\forall s \in S \exists A_e : A_e = A_r^s$, where $A_e \subseteq A$, $S \subseteq STR$.

min equal $minEqual(S, A_e)$ describes the constraint: $\forall s \in S \exists A_e : A_e \subset A_e^s$, where $A_e \subset A$, $S \subset STR$.

min-function $minFunc(s, f)$: requirement for a functionality f to be available at stqe s: $f \in \mathcal{F}^s$

min-property $minProp(s, f, p, n)$ constrains the property p_f of a functionality f to be $p_f \geq n$, where the constraint implies $minFunc(s, f)$

To handle service preferences within this representation, e.g., when a particular agent should visit two distinct locations, equality constraints are required. An equality constraints can define partial or full paths for the same instances of agents, e.g., to control that the same agent visiting location l_0 at timepoint t_0 will also visit location l_1 at t_1. Detailing functionality request with min and max property constraints are motivated by informed repair strategies, e.g., a property constraint can demand a mobile agent with a particular transport capacity. In Section 4 we will detail this reasoning further.

Distinction & Observation

Existing VRP based approaches most often only consider a subset of the presented constraints, while the mission planning problem formulation embeds the following VRP properties: time windows, capacity constraints, heterogeneous agents, fleet size minimization and vehicle synchronization. Furthermore, additional special features are introduced: (i) it is not only accounted for commodity demand, but rather a combination of commodities and vehicles that provide certain functional properties; (ii) the use of qualitative temporal constraints (in contrast to hard or soft quantitative time windows), which enables partially ordered requirements and increase the flexibility to synchronize agent activities; (iii) the mix-in of a multi-pickup multi-delivery problem in contrast to a single drop-off.

4 Organization Modeling

To reason upon a reconfigurable multi-robot system a special so-called organization model is introduced which describes all resources that can be part of a reconfigurable multi-robot team: atomic agents as well as their functionalities and properties thereof. As detailed in (Roehr and Kirchner 2016) the organization model builds upon an ontological description, which: (a) encodes information about resources that are associated with agent types, (b) associates interfaces with agent types, (c) defines compatibility between interfaces, (d) allows the identification of feasible, and (e) allows inferencing functionality of composite agents.

In combination of all features the organization model serves as main reasoner to identify composite agents and coalition structures, which are suitable to support a set of time and location bounded functional requirements.

The following sections will describe agent properties, and the details of identifying feasible composite agents, and subsequently suitable agent with respect to a given functionality.

Atomic agent type

Each agent type is associated with the following essential attributes:

mobility $mobile(\widehat{a})$ defines whether an agent of type $\widehat{a}$ is mobile or not.

transport capacity $tcap(\widehat{a})$ defines the maximum total capacity (measured in storage units) of an agent of type $\widehat{a}$ to transport others, and $tcap(\widehat{a_i}, \widehat{a_j})$ defines the maximum capacity of an agent type $\widehat{a_i}$ to transport an agent type $\widehat{a_j}$.

capacity consumption $tcon(\widehat{a})$ defines the number of storage units an agent of type $\widehat{a}$ consumes temporarily when being transported (currently this is set to 1 by default);

velocity $v_{nom}(\widehat{a})$ defines the nominal velocity of an agent type $\widehat{a}$, $v_{nom} \geq 0$ for mobile atomic agent types and $v_{nom} = 0$ for immobile

power $pw(\widehat{a})$ defines the nominal required power to operate an agent of type $\widehat{a}$

mass $mass(\widehat{a})$ defines the mass of an agent

energy $energy(\widehat{a})$ defines the available electrical energy that initially comes with an atomic agent

General and composite agent type

Some properties of composite agents can be inferred from their compositing atomic agents: Avella et al. (Avella, Boccia, and Sforza 2004) (though in the context of route constraints) label these as 'numerical totalisable', e.g., here mass and energy, which can be easily represented as sum of the property values of each atomic agent forming the composite agent. Inferring the capacity, in contrast, can be complex due to geometrical packaging constraints. The present model, however, currently ignores geometrical packing constraints and checks only connectivity based on interface compatibility.

Feasible agents

The main feature of a reconfigurable multi-robot system is the possibility for physical interconnection, but the not all composite agents are feasible. The compatibility and availability of connecting interfaces can restrict the design of a fully connected composite agent. Interfaces can come in different variants, e.g., for the reference system in (Roehr and Hartanto 2014) a male and female (also referred to as *EmiPassive* and *EmiActive*). But only one male and one female interface can be coupled. Atomic agents can comprise any number of interfaces, but based on Assumption 3.2 exactly one interface can be used for the connection to another agent's interface. For a successful connection, both interfaces need to be compatible.

Checking feasibility is a matching problem for graph $G = (V, E)$, with constraints for the existence of edges, where a vertex $v \in V$ represents a single interface. We denote I^A as the set of all interfaces of a set of agent A, so that $V = I^A$ with the corresponding partitioning $I^A = I^0 \cup I^1 \cup \ldots I^n$, where $n = |A| - 1$ and the set of interfaces of an agent a_0 is represented as $I^0 = \{i_{0,0}, i_{0,1}, \ldots, i_{0,|I^0|-1}\}$. The adjacency matrix is an $m \times m$ Matrix C, where $m = |I^A|$, and $\forall i, j \in I^A : c_{i,j} = 0, 1$ (rows and columns are annotated with the interface):

$$\begin{array}{c} \\ i_{0,0} \\ i_{0,1} \\ \vdots \\ i_{n,|I^n|} \end{array} \begin{array}{c} \begin{array}{cccc} i_{0,0} & i_{0,1} & \cdots & i_{n,|I^n|} \end{array} \\ \begin{pmatrix} c_{i_{0,0},i_{0,0}} & c_{i_{0,0},i_{0,1}} & \cdots & c_{i_{0,0},i_{n,|I^n|}} \\ c_{i_{0,1},i_{0,0}} & c_{i_{0,1},i_{0,1}} & \cdots & c_{i_{0,1},i_{n,|I^n|}} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i_{n,|I^n|},i_{0,0}} & c_{i_{n,|I^n|},i_{0,0}} & \cdots & c_{i_{n,|I^n|},i_{n,|I^n|}} \end{pmatrix} \end{array}$$

Checking connectivity means search for a valid assignment for the adjacency matrix C, while the following constraints hold for this symmetric matrix, where $c_{p,q} = c_{q,p}$, $p, q \in I^A$:

$$\forall a_k \in A, p, q \in I^k : c_{p,q} = 0 \quad (1)$$

$$\forall a_k \in A, p \in I^k : \sum_{q \in I^A} c_{p,q} \leq 1 \quad (2)$$

$$\forall a_k, a_l \in A : \sum_{p \in I^k} \sum_{q \in I^l} c_{p,q} \leq 1 \quad (3)$$

Constraint 1 defines that no self links are allowed for an atomic agent, while Constraint 2 restricts each interface to be part of maximum one link only. Finally, Constraint 3 enforces Assumption 3.2, so that two atomic agents have to be connected by one link.

The assignment problem is solved using constraint-based programming and implemented using Generic constraint development environment (Gecode) (Schulte and Tack 2012), where the matrix entries represent the constraint-satisfaction problem (CSP) variables, each with the domain $D_c = \{0, 1\}$. Since a single agent might have multiple interfaces of the same type, the corresponding column assignments in the adjacency matrix are interchangeable, and create redundant solutions. We use symmetry breaking to reduce the number of redundant solutions, and to further speed the assignment process up, variable assignments are done in order of the least constrained agents, i.e.

$$a^* = \underset{a^k \in A}{\operatorname{argmin}} \frac{1}{|I^k|} \sum_{q \in I^A} \sum_{p \in I^k} c^*_{p,q} \quad (4)$$

, where

$$c^*_{p,q} = \begin{cases} 1 & \text{if } c_{p,q} \text{ is already assigned} \\ 0 & \text{otherwise} \end{cases}$$

In practice, we will also add a small fractional random bias which serves as tie breaker when between variables with equally constrained agents.

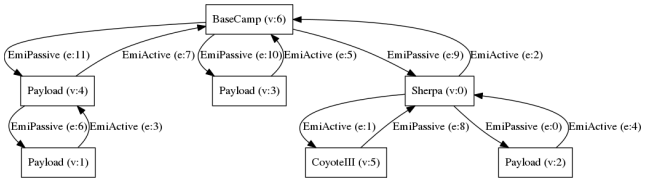


Figure 2: A feasible link structure for a composite agent after solving the assignment problem. Edges are annotated with the interface corresponding to the source vertex. Agent models and interfaces are related to the reference system described in (Roehr, Cordes, and Kirchner 2014).

Figure 2 shows the result of a successful assignment procedure, for a set of seven agents, where the agent Sherpa comprises four male and two female interfaces, Payload one of each, CoyoteIII two male and BaseCamp five male.

Suitable agents

An atomic agent is associated with a set of resources, being either physical components or virtual ones such as capabilities and functionalities it can offer; the same holds for composite agents. Additionally, virtual resources can depend upon other resources, leading to a hierarchical dependency structure. In order to resolve the functional requirements of the mission specification to actual suitable agent type which support the requirements, the organization model provides a mapping function: $\mu : \mathcal{P}^{\mathcal{F}} \rightarrow \mathcal{P}^{\theta(\mathcal{A})}$, where $\mathcal{P}^{\mathcal{F}}$ represents the powerset of all functionalities, and $\mathcal{P}^{\theta(\widehat{A})}$ denotes the powerset of all general agent types. The function μ thus maps a set of functions to a set of general agent types which support this set of functions and forms feasible agents. The organization model encodes functionality based on resource availability, where resources can be physical devices and capabilities belonging to an agent. Thus, the organization model allows to infer functionality from a given agent type and its associated resource structure: $\mu^{-1} : \mathcal{P}^{\theta(\widehat{A})} \rightarrow \mathcal{P}^{\mathcal{F}}$. Thereby, the organization allows to map from agents to functionalities and back. An additional generalization can be achieved, when the mapping does not only account for a set of functionalities, but a set of arbitrary resource types which can be associated with a general agent. Currently, however, we have restricted the mapping to functionality.

Each agent type is associated with a maximum cardinality for a resource type, which reflects its initial and original state. Note, that setting the maximum cardinality still allows to lower the bound, in contrast to defining the exact cardinality. Therefore, the current modeling approach is prepared to consider resource failure or removal in future extensions.

Support is defined for an agent type and a single resource concept c as follows (cf. Roehr and Kirchner (Roehr and Kirchner 2016)):

$$support(\widehat{a}, c, f) = \frac{card_{max}(c, \widehat{a})}{card_{min}(c, f)} \quad (5)$$

, where $card_{min}$ and $card_{max}$ return the minimum and maximum required cardinality of resource instances. Accordingly, support of a function f with respect to a resource

class c can be categorized as follows:

$$support(\widehat{a}, c, f) = \begin{cases} 0 & \text{no support} \\ \geq 1 & \text{full support} \\ > 0 \text{ and } < 1 & \text{partial support} \end{cases} \quad (6)$$

Since composite agents might comprise a high level of redundancy, the introduction of a saturation bound shall reduce the number of agents which have to be considered when a given set of functionalities is demanded. We define the *functional saturation bound* for an atomic agent type $\widehat{a}$ with respect to functionality f using the inverse of $support$:

$$FSB(\widehat{a}, f) = \max_{c \in \mathcal{C}} \frac{1}{support(\widehat{a}, c, f)}, \quad (7)$$

where $\mathcal{C}$ is a set of resource classes and $\forall c \in \mathcal{C} : card_{min}(c, f) \geq 1$ to account only for relevant resource classes. If there is no support for a $c \in \mathcal{C}$ such that $support(\widehat{a}, c, f)$ then $FSB(\widehat{a}, f) = \infty$. Similarly, the bound for a set of functions $\mathcal{F}$ is defined as:

$$FSB(\widehat{a}, \mathcal{F}) = \max_{f \in \mathcal{F}} FSB(\widehat{a}, f) \quad (8)$$

Identifying functionality support for a general agent type is equivalent to an atomic agent type, but to compute the maximum resource cardinalities the following holds:

$$card_{max}(c, \widehat{GA}) = \sum_{\widehat{a} \in \widehat{GA}} \gamma_{\widehat{GA}}(\widehat{a}) card_{max}(c, \widehat{a}) \quad (9)$$

, where $c \in \mathcal{C}$. Minimum resource cardinalities will be computed equivalently using $card_{min}(c, \widehat{a})$.

The number of general agent types that can support some functionality can be large, but it can be observed that for a supported set of functionalities a set of minimal general agent types $\mathcal{G}_{min}$ exists.

Definition 4.1. *A general agent type which supports a given set of functionalities and whose agent type cardinalities cannot be further reduced is denoted* ***minimal*** *with respect to the given set of functionalities.*

Hence, a minimal general agent type represents a lower bound to satisfy functionality requirements with a given combination of agent types.

5 Mission planning

The primary goal is to provide a valid assignment for the provided mission specification (cf. Section 3), while fleet size minimization and total cost minimization are secondary. The actual planning process is based on several stages in order to generate solutions:

(1) temporal ordering of all timepoints using a temporal constraint network
(2) upper and lower bounding of agent type cardinality for each spatio-temporal requirement
(3) generation of agent role timelines according to unification constraints and agent type cardinalities
(4) flow optimization to transfer immobile agents with mobile agents
(5) quantification of timepoints, based on transition times

Stage (1) creates a sequence of ordered timepoints, which is a necessary precondition for all next stages in order to identify concurrent resource usage and creating a commodity flow network. Stage (2) identifies the minimally required set of agent roles, and is for this reason a key element for minimization of the resources in use. Stage (3) takes all mission constraints into account in order to suggest a feasible agent role assignment. This assignment is handed toa optimized in stage (4).

Constraint-based programming is involved in the reasoning of the organization model, and the planning stages (1),(2), and (3). Each of these stages involves the definition of appropriate branching strategies, and symmetry breaking conditions, and (3) uses of special implemented constraint propagator. If at any listed stage the search process fails, backtracking will be performed to the previous stage. The following sections will describe the details of the individual stages:

Temporal Ordering of Timepoints To generate valid timelines and identify resource conflicts the approach requires a fully ordered set of timepoints. The generation of a fully constrained set of timepoints is based on qualitative temporal reasoning using point algebra with the set of relations $REL = \{>, <, =\}$ (Rina Detcher 2003). Consistency of the Temporal Constraint Network (TCN) is checked using a CSP which is defined by a set timepoint variables $T = \{t_1, t_2, \ldots, t_{|T|}\}$, a set $D = \{D_1, D_2, \ldots, D_{|T|}\}$ to represent the domain values for each timepoint $t \in T$, and a constraint set C with constraints of the form $C = \langle t_n, rel_i, t_m \rangle$, where $n, m = 1, \ldots, |T|$, and $rel_i \in REL$. A constraint is fulfilled if the relation described by $c \in C$ between two timepoint variables is fulfilled.

The final domain for each variable is restricted to a singleton: $|D_i| = 1$ and permitted values are $D_i \subseteq \{1, 2, \ldots, |T|\}$. If a full assignment of values can be found, the TCN is consistent, and the ordering of timepoints corresponds to the ordering of the assigned values. The qualitative temporal reasoning is sufficient to synchronize tasks, but only the quantification of time in the last stage of the planning approach will verify the temporal consistency of a solution.

Bounding agent type cardinality To perform an upper and lower bounding of agent type cardinality a matrix based representation for spatio-temporal requirements and agent types is used, where $x_{i,j}$ represents the cardinality of agent type $\widehat{a}_j \in \widehat{A}$ and $s_i \in STR$. The following matrix representation with annotated rows and columns illustrates the meaning of each related CSP variable:

$$\begin{array}{c} \\ s_0 \\ s_1 \\ \vdots \\ s_m \end{array} \begin{array}{c} \begin{array}{cccc} \widehat{a}_0 & \widehat{a}_1 & \cdots & \widehat{a}_n \end{array} \\ \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,n} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,0} & x_{m,1} & \cdots & x_{m,n} \end{pmatrix} \end{array} \quad (10)$$

, where $n = |\widehat{A}| - 1$, and $m = |STR| - 1$. Each variable x has an initial domain of positive integers $D_x = \{0, 1, \ldots\}$.

Since resource availability is restricted, the general agent type $\widehat{GA}$ which is part of the mission specification defines an upper bound for all agent type cardinalities, which will be referred to as $\widehat{GA}_{UB}$ for readability.

Spatio-temporal requirements, however, can overlap, i.e. when they refer to the same location and their time intervals overlap. For a set of overlapping spatio-temporal requirements $\Omega = \{s_i, \ldots, s_j\}$, $s_i, s_j \in STR$ the upper bound is enforced as follows:

$$\forall \widehat{a_j} \in \widehat{A} : \quad \sum_{s_i \in \Omega} x_{i,j} \leq \gamma_{\widehat{GA}_{UB}}(\widehat{a_j}) \tag{11}$$

The organization model is required to translate the requirements for functionalities into requirements for (suitable) general agent types, and apply the functional saturation bound. Lower bounds for each spatio-temporal requirement result from the combination of demanded functionalities and the given minimum agent type cardinalities. This lower bound represents a set of minimal general agents which is translated into the spatio-temporal requirement's CSP variable domain. This domain is considered in the CSP by using extensional constraints for the assignment of model cardinalities, thus restricting model combination to minimal general agents. The extensional constraints enforce an exact assignment, but any full assignment of model cardinalities is only a lower bound for the subsequent agent role assignment stage. If no assignment can be found, too few resources are available to fulfil the mission requirements; the planning continues with another assignment of the temporal constraint network if possible or fails otherwise.

Agent roles Subsequent to the CSP branching on bounded agent type cardinalities, a candidate assignment of agent roles to spatio-temporal constraints can be computed using a set of integer variables $y_{i,k,j}$, for $s_i \in STR$, $\widehat{a}_k \in \widehat{A}$, and $0 \leq j \leq \gamma_{\widehat{GA}_{UB}}(\widehat{a}_k)$, which have the domain $D = \{0, 1\}$:

$$\begin{array}{c} \\ s_0 \\ s_1 \\ \vdots \\ s_m \end{array} \begin{array}{c} \begin{array}{cccccc} r_0^{\widehat{a}_0} & \cdots & r_0^{\widehat{a}_k} & \cdots & r_l^{\widehat{a}_n} \end{array} \\ \begin{pmatrix} y_{0,0,0} & \cdots & y_{0,k,0} & \cdots & y_{0,n,l} \\ y_{1,0,0} & \cdots & y_{1,k,0} & \cdots & y_{1,n,l} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ y_{m,0,0} & \cdots & y_{m,k,0} & \cdots & y_{m,n,l} \end{pmatrix} \end{array} \tag{12}$$

, where $l = \gamma_{\widehat{GA}_{UB}}(\widehat{a_n}) - 1$, $m = |STR| - 1$, and $n = |\widehat{A}| - 1$.

Additional constraints are applied to guarantee unary agent role usage for time-overlapping constraints, and the general mission constraints described in Section 3 can directly be translated into low level CSP constraints, e.g., such as equality constraints minEqual, maxEqual as well as distinction constraints. Since agent roles of the same agent type are interchangeable symmetry breaking is applied to reduce the number of redundant solutions. While constraint propagation will reduce the corresponding domain and will lead to value assignment, full assignment of variables will only be performed for agent roles that (a) have an assignment apart from the single starting location, and (b) are mobile. To the first kind of agent roles we will also refer to as *active* agent roles. This partial assignment allows to extract full timelines for active mobile agents and partial timelines for active immobile agents. Both form the basis for a multi-commodity flow problem which is solved using integer linear programming.

Timeline Generation Variable assignment for a single agent role variable assignment have to fulfill another important property: they have form a path in a temporal-expanded network. Ford and Fulkerson (Ford and Fulkerson 1963) have shown that networks can represent flow over time, and we similarly rely on what we call a temporal-expanded network to compute a flow-based representation for the mission planning problem. The temporal-expanded network has a bound on the number of edges by allowing only edges between vertices which are related to neighbouring timepoints and point forward in time:

Definition 5.1. *A time-expanded network for a set of timepoints T and a set of locations L is a graph $G = (V, E)$ with the following properties: Each vertex in V corresponds to a unique location timepoint tuple $v_{l,t} = (l, t)$, where $l \in L$, and $t \in T$. The set of edges is restricted: $e \in E \implies e = (v_{t_n,l_i}, v_{t_{n+1},l_j})$, where $n = 0, \ldots, |T| - 1$ and $i, j = 1, \ldots, |L|$. Without loss of generality $t_0 \leq t_1 \leq \cdots \leq t_{|T|-1}$.*

A custom (path) propagator has been implemented to exploit the structure of the network and enforce a constrained path in the network. This leads to a faster assignment process of agent role variables.

Multi-commodity flow When the role assignment process is completed (fully for the mobile agents, and partially for the immobile ones), it is straightforward to translate the agent role timelines into a multi-commodity min-cost flow problem (Ahuja, Magnanti, and Orlin 1993): mobile agents represent transport providers, while immobile agents will be treated as individual commodities. Thus, edges in the network are either 'local' connections since they refer to the same location, or they are part of mobile agent routes. While we assume that local connections have infinite capacity, edges created as result of a mobile agent transition have an upper capacity bound defined by the transport capacity of the corresponding mobile agent. All available mobile agents span a flow network over which commodities, or here immobile agents, can be routed to their target destinations. But agents are not restricted to a single target destination, so that requirements partially define a route for each agent. Therefore, the flow network represents all immobile agent requirements by minimum trans-flow requirements. Although bundling all agent types into one commodity would lead to a compact representation, route requirements for individual agents could not be set properly. Hence, each immobile agent role corresponds to a commodity, and role usage requirement are translated in to minimum transition requirements as already mentioned.

Mobile and immobile agent routes are transformed into a multi-commodity min-cost flow problem with unit com-

modity cost (Ahuja, Magnanti, and Orlin 1993):

$$\min \sum_{k,m} x_m^k$$

$$s.t. \sum_{e_m \in A_n} x_m^k - \sum_{e_m \in B_n} x_m^k = \begin{cases} S_k^+ & \text{if } n = s_k \\ -S_k^- & \text{if } n = t_k \\ 0 & \text{otherwise} \end{cases}, \forall\, n, k$$

$$x_m^k \geq l_m^k \wedge x_m^k \leq u_m^k$$

, where

G	=	(V, E)
K	=	number of commodities, $k = \{1, \ldots, K\}$
m	=	$\{1, \ldots, M\}, M = \lvert E \rvert$
e_m	=	edge between node i and node j, i.e. (i,j)
x_m^k	=	flow for commodity k in arc e_m
c_m^k	=	unit cost for commodity k in arc e_m
u_m^k, l_m^k	=	upper/lower bound for commodity k flow through edge m
s_k, t_k	=	source/target of commodity k, $s_k \in V$
S_k^+, S_k^-	=	supply/demand of $s_k \in V$
B_n	=	set of incoming edges of node n
A_n	=	set of outgoing edges of node n

To solve the network flow problem, the problem is first translated into a standard representation (CPLEX LP) so that different LP solvers can be used to solve the optimization problem (here: SCIP (Achterberg et al. 2008) and GLPK (Free Software Foundation 2015)). Any feasible and optimal solution of the network flow problem is also a feasible, but not necessarily an optimal solution for the mission assignment problem.

Quantification of time A full solution still requires the quantification of temporal intervals: the qualified temporal network is therefore converted into a quantitative simple temporal network where the transitions between locations (and stqes) are based on the time required for the mobile systems to perform the location transitions and to form composite agents. Any min and max duration constraints will also apply at this planning stage.

Search & Solution repair

The previously described constraints lead to the generation of role timelines, and the CSP framework Gecode (Schulte and Tack 2012) has been used for the implementation. All role timelines are not only checked for feasibility via the multi-commodity min-cost flow optimization, but at the same time locally optimized. Still, finding a feasible solution for a highly restricted set of resources can be a significant challenge. Several strategies can be considering for search, and our initial approach interprets lower agent type cardinality bounds as exact bounds - with the intention to keep the fleet size minimal and enlarge only when necessary. Hence, in the case when no optimal solution can be found, the infeasible (LP) solution is analysed to identify open flaws, i.e. unfulfilled minimum commodity transflow requirements. Upon identification of all flaws, a repair heuristic can be applied which injects additional transport provider requirements, thereby triggering either the change of existing mobile agent routes, or an increase of the lower agent type cardinality. The min-property constraint is used to augment the mission and restart the search after the local repair. For highly constrained missions, the repair process can reduce the number of flaws, but is slow at finding feasible solutions, hence showing that the heuristic is currently insufficient for complex setups.

An alternative is offered by the relaxation of cardinality bounds. In order to speed up finding an initial feasible solution, it is beneficial not to interpret the lower agent type cardinalities as exact bounds. Allowing an additional set of mobile systems (still within the number of the available ones) can reduce the time to find a feasible solution, but leads to higher redundancies and therefore less efficient solutions, since more agents will be required.

Figure 3 shows a computed feasible solution. The general agents available for the mission are 3 Sherpa, 2 CREX, 3 Coyote II, 16 Payload, and 5 BaseCamp, where some agent interfaces are listed in the upper left corner of the figure. The assignment at the location lander shows, that only a subset of atomic agent is required for the solution. Fulfilled atomic agent requirements are highlighted as green squares, while the presence of systems without requirements is shown in green. These requirements, however, only represent one feasible set of atomic agent requirements which has been inferred from required functionalities. This solution has been computed within few seconds but only for a relaxed cardinality bound with two additional mobile agent roles (per mobile agent type).

Mission solution & cost function

The overall state of the agent organization, i.e. current connection state of atomic and composite agents is reflected by the coalition structure. In order to cost factor the dynamics in an agent organization two related concepts have to be used: policies and heuristics. Policies are required to define rules for selection and attribution. For example in the case of a transport multiple mobile robot may be available to perform this transport. To decide which one to take, a transport policy has been introduced, which chooses the agent with the largest transport capacity. For attribution energy consumption in a composite agent serves as main example. Since multiple power sources might exist is such system, a consumption policy has to distribute the consumption to all energy providers. Here, for the default policy each provider takes a share relative to its contribution to the overall energy capacity of the composite agent. Heuristics serve to interpolate a organizational state and estimate final mission costs: a duration heuristic for moving between locations relies on the information about the distance and the nominal speed of the transporting agent. Energy cost are depending upon the duration heuristic by relating duration to the power consumption of a composite system. Any reconfiguration changes this coalition structure, but requires a transition time, so that

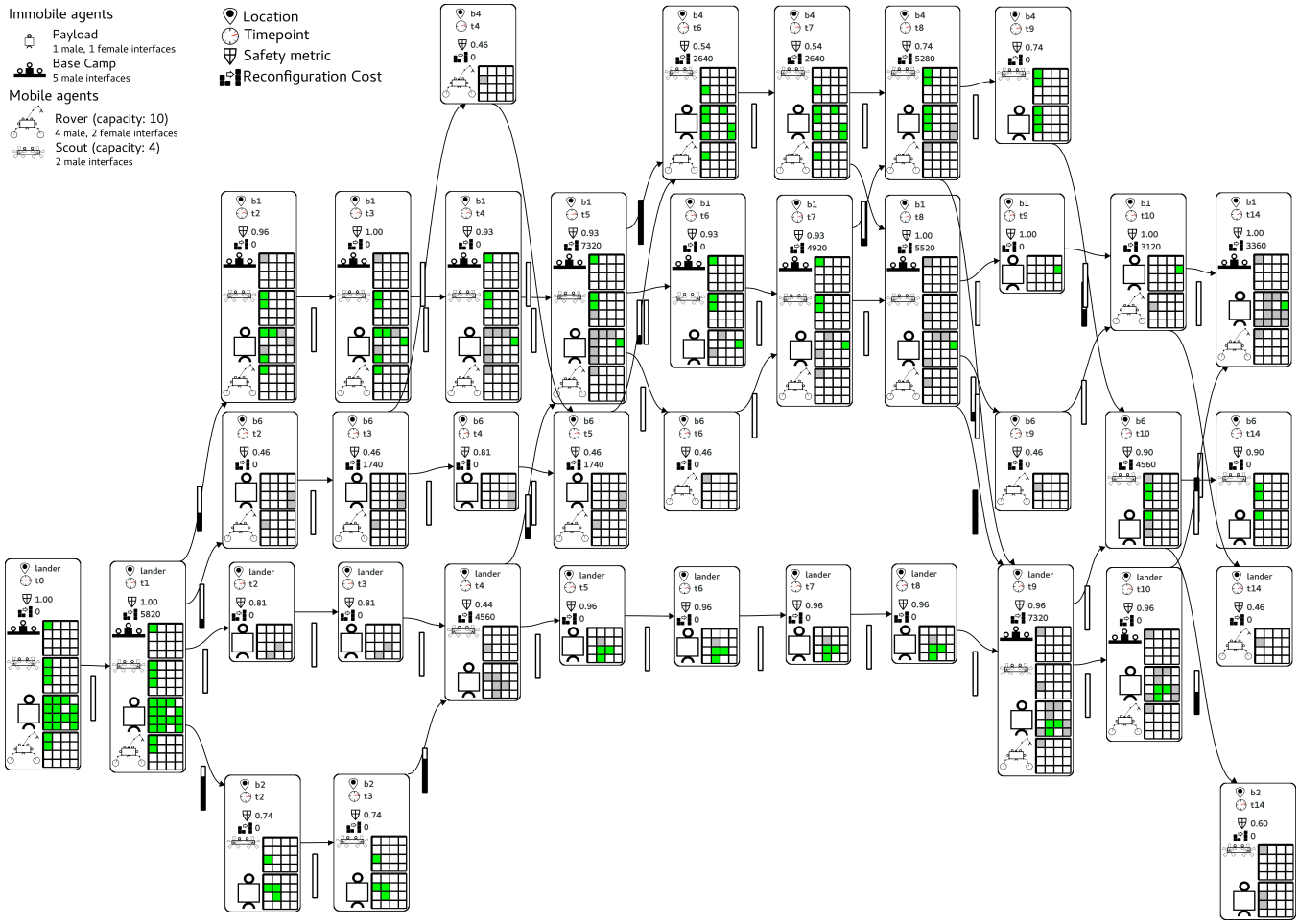


Figure 3: Example of a feasible solution for a full mission with locations = {lander,b1,b2,b4,b6}, timepoints = {t0,t1,...,t10,t14}. Fillbars indicate the consumed capacity of mobile agents. Color-coded boxes represent unique agent roles (only for better visualization limited to 16 per agent type): green = fulfilled requirement, gray = presence without a requirement.

$\rho(CS_i^A, CS_j^A)$ defines the time to transition from one coalition structure CS_i^A to another CS_j^A. This cost heuristic assumes the same location of all agents in A.

The objectives of the planner is to find a solution that balances the overall energy consumed with the level of safety:

distance $d(a, \mathcal{M}_s)$ travelled distance of an agent a in mission $\mathcal{M}_s$

operation time $op(a, \mathcal{M}_s) = d(a, \mathcal{M}_s)/v_{nom}(a)$ duration of operation of an agent a

energy $E(a, \mathcal{M})$, where $E(a, \mathcal{M}_s) = op(a, \mathcal{M}_s) \cdot pw(\widehat{a})$ overall consumed energy by agent a to perform $\mathcal{M}_s$; $E(\mathcal{M}) = \sum_{a \in A}$ overall consumed energy per mission

safety $SAF(\mathcal{M}_s) = \min_{s \in STR} saf(s)$ represents the minimal safety level (here: redundancy) of the mission, where $saf(s)$ defines the safety metric associated with an stqe s based on the available (general) agent and with respect to the required set of resources; currently a redundancy based model is used to estimate the probability of survival based on an agent's set of component required to provide the functionalities in $\mathcal{F}$ (cf. (Roehr and Kirchner 2016) for details), such that $0 \leq saf(s) \leq 1$.

fulfillment $SAT(\mathcal{M}) = \frac{1}{|STR|} \sum_{s \in STR} sat(s)$ represents the ratio of fulfilled requirements, where

$$sat(s) = \begin{cases} 0 & \text{, unfulfilled} \\ 1 & \text{, fulfilled} \end{cases}$$

This following cost function reflects a balancing of three general mission aspects: efficiency through the energy cost function, efficacy through checking the level of fulfillment, and safety as redundancy dependant survival metric; for balancing the parameters α, β and $gamma$ can be used:

$$cost(\mathcal{M}_s) = \alpha E(\mathcal{M}_s) + \beta SAT(\mathcal{M}_s) + \gamma SAF(\mathcal{M}_s)$$

Figure 3 shows an example of a feasible solution. Each such solution can be translated into action plans for individual agent roles. Each vertex of the solution network serves as synchronization point and assumes reconfiguration operation to account for necessary coalition structure changes; the reconfiguration cost are annotated accordingly, along with

the safety metric. Overall cost for the provided solution network are computed by constructing a simple temporal constraint network (Dechter 2003) where the bounds are defined by the transition times of the mobile agents.

6 Conclusion & Future Work

This paper presents the continued work for developing a planning system for a reconfigurable multi-robot system. The planner relies on constraint-based programming to specify and solve missions involving reconfigurable multi-robot systems, which is combined with multi-commodity flow optimization as local search. Furthermore, it suggests a multi-objective optimization target involving efficacy, efficiency and safety. The approach presented in this paper does not only result in a planning system for reconfigurable multi-robot system, but also in a tool which allows to analyse the effects of using reconfigurable multi-robot systems in robotics missions. Future work will firstly focus on introducing better plan repair heuristics, and the extended use of meta-heuristic search strategies to improve the performance and scalability of the embedded local search approach. Secondly, a resource augmentation stage will be added in order to use previously unused resources to raise the level of safety.

Acknowledgments This work was supported by the German Space Agency (DLR) under grant agreement 50RA1301 and 50RA1701, and the project Hi-Digit Pro 4.0.

References

Achterberg, T.; Berthold, T.; Koch, T.; and Wolter, K. 2008. Constraint Integer Programming: A New Approach to Integrate CP and MIP. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5015 LNCS. Berlin, Heidelberg: Springer Berlin Heidelberg. 6–20.

Ahuja, R. K.; Magnanti, T. L.; and Orlin, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Michigan, US: Prentice Hall.

Ahuja, R. K.; Orlin, J. B.; and Sharma, D. 2000. Very large-scale neighborhood search. *International Transactions in Operational Research* 7(4-5):301–317.

Avella, P.; Boccia, M.; and Sforza, A. 2004. Resource constrained shortest path problems in path planning for fleet management. *Journal of Mathematical Modelling and Algorithms* 3(1):1–17.

Baldacci, R.; Battarra, M.; and Vigo, D. 2008. Routing a heterogeneous fleet of vehicles. In *Operations Research/ Computer Science Interfaces Series*, volume 43. Springer. 3–27.

Coltin, B., and Veloso, M. 2014a. Online pickup and delivery planning with transfers for mobile robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 5786–5791. IEEE.

Coltin, B., and Veloso, M. 2014b. Ridesharing with passenger transfers. In *Intelligent Robots and Systems (IROS 2014)*.

Coltin, B., and Veloso, M. 2014c. Scheduling for Transfers in Pickup and Delivery Problems with Very Large Neighborhood Search. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2250–2256.

Dechter, R. 2003. Temporal Constraint Networks. In Dechter, R., ed., *Constraint Processing*, The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann. chapter 12, 333–362.

Dettmann, A.; Wang, Z.; Wenzel, W.; Cordes, F.; and Kirchner, F. 2011. Heterogeneous Modules with a Homogeneous Electromechanical Interface in Multi-Module Systems for Space Exploration. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA '11)*, 1964–1969. Shanghai, China: ESA.

Dignum, V., ed. 2009. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global.

Dondo, R., and Cerdá, J. 2007. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research* 176(3):1478–1507.

Drexl, M. 2013. Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research* 227(2):275–283.

Ford, L. R., and Fulkerson, D. R. 1963. Flows in networks. Technical report, The RAND Corporation, Santa Monica, California.

Free Software Foundation. 2015. GLPK (GNU Linear Programming Kit). https://www.gnu.org/software/glpk;1 October 2015.

Rina Detcher. 2003. *Constraint Processing*, volume 33. Morgan Kaufmann Publishers Inc.

Roehr, T. M., and Hartanto, R. 2014. Towards safe autonomy in space exploration using reconfigurable multi-robot systems. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2014)*. ESA.

Roehr, T. M., and Kirchner, F. 2016. Spatio-Temporal Planning for a Reconfigurable Multi-Robot System. In Finzi, A., and Karpas, E., eds., *Proceedings of the 4th Workshop on Planning and Robotics (PlanRob)*, o.A.

Roehr, T. M.; Cordes, F.; and Kirchner, F. 2014. Reconfigurable Integrated Multirobot Exploration System (RIMRES): Heterogeneous Modular Reconfigurable Robots for Space Exploration. *Journal of Field Robotics* 31(1):3–34.

Schulte, C., and Tack, G. 2012. View-based propagator derivation. *Constraints* 18(1):75–107.

Sonsalla, R.; Cordes, F.; Christensen, L.; Planthaber, S.; Albiez, J.; Scholz, I.; and Kirchner, F. 2014. Towards a Heterogeneous Modular Robotic Team in a Logistic Chain for Extraterrestrial Exploration. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*. Montreal, Canada: ESA.

Toth, P., and Vigo, D., eds. 2014. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. MOS-SIAM, 2 edition.

Dynamic rescheduling in energy-aware unrelated parallel machine problems

Sergio Ferrer, Giancarlo Nicolò, Miguel A. Salido, Adriana Giret and Federico Barber
Instituto de Automática e Informática Industrial, Valencia, Spain
Universitat Politècnica de València, Valencia, Spain
{serfers2, giani1, msalido, agiret, fbarber}@dsic.upv.es

Abstract

Advance in applied scheduling is a source of innovation in the manufacturing field, where new results help industrial practitioners in improving manufacturing line performances. In this domain, the manufacturing line environment is usually assumed to be static, in contrast to real life scenarios where disruptions are frequent and the original solution can drastically change. To address this intrinsic characteristic of a real environment, the scheduling system accounted to provide the solution for the production schedule needs to be able of monitoring possible disruptions and, through a *rescheduling* process, to produce a new scheduling solution that considers the unforeseen. The literature of rescheduling problems in the case of single-objective optimization is a well study field, while there is a lack of extensive studies for the case of rescheduling for multi-objective optimization, especially when one of the objective represents energy measures (energy aware scheduling). To improve the field, this paper extends a previous work over an energy aware scheduling problem, modeled from a real industrial case study, extending the manufacturing environment as a dynamic one and introducing specific rescheduling techniques tackling the machines unavailabilities through new rescheduling techniques. Two new rescheduling techniques are developed (greedy-heuristic and meta-heuristic) and compared to existing approach thought manufacturing environment simulations. The results give insight over which techniques better performs in terms of rescheduling quality and computational time.

1 Introduction

Applied scheduling is the field of study that aims to fill in the gap between the scheduling theory and its application to real scenario, giving scheduling practitioners new and innovative solutions to face current trend in their domain of work. Of particular interest for the applied scheduling is the manufacturing domain, where one of the main application of the scheduling problems is in the context of optimizing the manufacturing line production. In this scheduling environment, the usual assumption is to analyze the problem as static, meaning that all the information of the scheduling problem are not going to change over the time. While this assumption can be useful for theoretical results and research simulations, when dealing with real scenario this assumption limits the possibility to concretely apply the developed static scheduling technique, because the only way to face environment disruption is to solve a new scheduling problem with substantial computational cost and lack of system responsiveness. To improve this situation, the environment has to be considered as dynamic and faced with predictive-reactive scheduling strategies. In this way, from the initial environment, a (predictive) scheduling solution is produced and when a disruption occurs a reactive strategy (rescheduling) tackles this issue producing a new feasible scheduling solution.

Current literature of rescheduling methods for dynamic scheduling environment is well developed. To understand the concepts of rescheduling strategies, policies, and methods in rescheduling manufacturing systems, the work of (Vieira, Herrmann, and Lin 2003) is of great helps thanks to the proposed framework to classify the rescheduling problem under analysis. There exist many works on concrete rescheduling problems, those mostly differs in terms of the kind of disruption they can manage. (Hall and Potts 2004) studies a scheduling problem where a set of original jobs has already been scheduled in order to minimize a cost objective, and then, a new set of jobs arrives and creates a disruption. The decision-maker needs to insert the new jobs into the existing schedule without excessively disrupting. The authors provide either an efficient algorithm or a proof that such an algorithm is unlikely to exist. (Qi, Bard, and Yu 2006) propose the problem of updating a machine schedule when either a random or an anticipated disruption occurs affecting a subset of the jobs. The proposed approach differs from most rescheduling analysis in that the cost associated with the deviation between the original and the new schedule is included in the model. In (Nouiri et al. 2018) a meta-heuristic rescheduling strategies for the dynamic flexible job-shop environment under machine disruption is presented and extensively tested against usual approach. The work outlines how machines disruption scenario significantly affects initial solution and the tradeoff between the evaluated approaches.

Most of the literature works over rescheduling problems address scheduling environment in which the main objectives are related to production objectives (completion time, tardiness, lateness, etc.), while current trends in manufacturing are taking in to consideration more complex objectives,

where environmental issues have to be addressed through the consideration of energy consumption in the optimization process (i.e. energy-aware scheduling problem) leading to the creation of the sustainable scheduling field of study. Even though the importance of sustainable scheduling is broadly affirmed through the scientific community (Bruzzone et al. 2012; Dai et al. 2013; Plitsos et al. 2017), most of the current literature in this field is addressing static problems (predictive scheduling), while literature works related to dynamic environments are very few in comparison to the variety of possible dynamic rescheduling environments determined by the combinatorial combination between scheduling environments, optimization functions, constraints and analyzed disruptions. In (Wang et al. 2016), authors tackle a bi-objective single machine batch scheduling problem where the first objective is to minimize the makespan and the second is to minimize the total energy costs, by considering both the machine utilization and the economic cost. An integer programming model is proposed and then, an exact ϵ-constraint method is adapted to obtain the exact Pareto front. In (Le and Pang 2013), the need to deal with uncertainties in energy optimization of flexible manufacturing systems is faced. It considers a dynamic scheduling problem which minimizes the sum of energy cost and tardiness penalty under power consumption uncertainties. In (Tonelli et al. 2016), a simple multi-agent system model is proposed to decompose an energy-aware scheduling problem into smaller subproblems. In this approach, each agent solves a subproblem by using a Mixed Integer Linear Programming (MILP) model and the results are combined to obtain a global solution. A similar idea is proposed in (Nicolo et al. 2017) where a set of solving techniques are compared and job features are studied in order to tackle energy-aware scheduling problems.

The proposed work is an extension of a well studied scheduling scenario derived from analyzing a manufacturing real case of an injection molding plastic industry (Paolucci, Anghinolfi, and Tonelli 2015). This problem can be considered as an energy-aware unrelated parallel machine scheduling problem with machine-dependent energy consumption and sequence-dependent setup time. The current literature that works over this problem, considered it as a static environment (predictive scheduling), and no previous works have been developed to manage this problem as a dynamic one. For the best of our knowledge, only one work can be considered slightly close to our proposal. The work of (Arnaout 2014) tackles a similar scheduling environment where a bi-objective function is optimized, but the main difference is the absence of energy information, leading to a interesting approach that cannot be applied in the energy-aware configuration. From above considerations, this work pretends to fill this lack in literature proposing new methodologies to solve a dynamic energy-aware scheduling problem.

The remainder of the paper is organizes as follow. Section 2 reports the problem under analysis introducing formal notation and a mathematical model for solving the predictive scheduling problem. Section 3 introduces the reactive rescheduling problem describing the category of disruptions taken in consideration. Section 4 describes the proposed approach to face the analyzed rescheduling problem introducing two new techniques. Section 5 compares the proposed techniques with a usual approach in the rescheduling literature (right-shift rescheduling). Section 6 summarizes the results and insights from the current work outlining possible future directions for improving and extending the current results.

2 Description of the Scheduling Problem

The scheduling problem under analysis is the energy-aware unrelated parallel machine scheduling problem with machine-dependent energy consumption and sequence-dependent setup time. It was firstly introduced in (Paolucci, Anghinolfi, and Tonelli 2015) and better studied in (Tonelli et al. 2016; Nicolò et al. 2016; Nicolo et al. 2017). In this problem, a set of orders, represented by jobs, has to be scheduled on a set of unrelated parallel machines. Each job has associated to its specific temporal and cost features: release date, due date and penalty cost. A job can be processed by one or more machines, where processing time and energy consumption of the job depends by the machine assigned to process it. Between the execution of two consecutive jobs over the same machine a setup time is needed, where its values depend on the jobs sequence and machine assigned to them. In the next section, the problem under observation is formally described and its mathematical formulation, introduced in (Paolucci, Anghinolfi, and Tonelli 2015) as Mixed Integer Linear Programming model, is reported.

2.1 Mathematical formulation

The following scheduling notation (Graham et al. 1979) formally represent the problem under study:

$$R_m|M_j, p_{jk}, E_{jk}, r_j, s_{ijk}|\sum w_j T_j, \sum E_{jk}, \sum S_{ijk}$$

The problem is multi-objective since there are three measures to be minimized, which are expressed as objective functions: the total weighted tardiness of the jobs $TT(s)$, the total energy consumption $EN(s)$, and the total setup time $ST(s)$. The solution s^* can be obtained by minimizing a 3-dimensional objective function:

$$s^* = \arg\min_{s \in S}[TT(s), EN(s), ST(s)] \quad (1)$$

where S denotes the feasibility space for the problem solution space. To represent the problem model and the objective function components to be optimized, a list of notations extracted from (Paolucci, Anghinolfi, and Tonelli 2015) is presented below.

2.2 Mixed integer programming model

In order to evaluate a solution of the problem, the three objective functions must be aggregated. The mixed integer linear programming (MILP) model (Paolucci, Anghinolfi, and Tonelli 2015) combines the three factors into a scalar function with a minimum deviation method, resulting in the fol-

Sets	
Notation	Definition
$J = \{1, \ldots, n\}$	the set of jobs
j_0, j_{n+1}	two fictitious jobs as first and last job on each machine
$M = \{1, \ldots, m\}$	the set of machines
$M_j, \forall j \in J$	the set of machines that can execute job j
$J_k, \forall k \in M$	the set of jobs that can be executed by machine k

Parameters	
Notation	Definition
B	a sufficiently large constant
$D_j, \forall j \in J$	the due date of job j
$R_j, \forall j \in J$	the release date of job j
$W_j, \forall j \in J$	the tardiness penalty of job j
$P_{jk}, \forall j \in J, \forall k \in M_j$	the processing time of job j on the eligible machine k
$E_{jk}, \forall j \in J, \forall k \in M_j$	the energy consumption of job j on the eligible machine k
$S_{ijk}, \forall i, j \in J$ $\forall k \in M_j \cap M_i\ , i \neq j$	the setup time on machine k between the completion of job i and the start of the subsequent job j
$\Pi_g, g = 1, 2, 3$	the weights of the objective function components

lowing scalar objective function F to be minimized:

$$\min\ \Pi_1 \cdot \frac{\sum_{j \in J} W_j \cdot t_j - f_1^-}{f_1^+ - f_1^-} + \Pi_2 \cdot \frac{\sum_{j \in J} \sum_{k \in M_j} E_{jk} \sum_{\substack{i \in J_k \\ i \neq j}} x_{ijk} - f_2^-}{f_2^+ - f_2^-} + \Pi_3 \cdot \frac{\sum_{k \in M} \sum_{i \in J_k} \sum_{\substack{j \in J_k \\ i \neq j}} S_{ijk} \cdot x_{ijk} - f_3^-}{f_3^+ - f_3^-} \tag{2}$$

The quantity f_g^-, g $\in \{1, 2, 3\}$ in (2) represents the best (i.e., minimum) value for the g-th component when this is optimized individually; f_g^+ is an estimation of the worst value for $f_g(s)$ that can be fixed as $f_g^+ = \max_{h \neq g} f_g(s_h^*)$, where (s_h^*) is the optimal solution found when the objective $f_h(s)$ is individually optimized. The weights Π_g, $g \in \{1, 2, 3\}$ in (2) express the relative importance given by the decision maker to the different objective components and are selected

Variables	
Notation	Definition
$c_j, \forall j \in J \cup \{0\}$	the completion time of job j
$st_j, \forall j \in J \cup \{0\}$	the starting time of job j
$t_j, \forall j \in J$	tardiness of job j with respect to its due date
$x_{ijk} \in \{0, 1\}$, $\forall i, j \in J \cup \{0, n+1\}$ $k \in M_i \cap M_j, i \neq j$	binary sequencing variables (i.e., $x_{ijk} = 1$ denotes that job i immediately precedes job j on machine k)
$y_{jk} \in \{0, 1\}$, $\forall j \in J$, $k \in M_j$,	binary assignment variables (i.e., $y_{jk} = 1$ denotes that j is processed by k)

such that $\sum_g \Pi_g = 1$. The above function (2) is subject to:

$$\sum_{\substack{i \in J_k \\ i \neq j}} x_{ijk} = y_{jk} \qquad \forall j \in J, k \in M_j \tag{3}$$

$$\sum_{\substack{j \in J_k \\ j \neq i}} x_{ijk} = y_{ik} \qquad \forall i \in J, k \in M_i \tag{4}$$

$$\sum_{k \in J_k} y_{jk} = 1 \qquad \forall j \in J \tag{5}$$

$$\sum_{j \in J_k} x_{0jk} \leq 1 \qquad \forall k \in M \tag{6}$$

$$c_j \geq R_j + \sum_{k \in M_j} P_{jk} y_{jk} \qquad \forall j \in J \tag{7}$$

$$t_j \geq c_j - D_j \qquad \forall j \in J \tag{8}$$

$$c_j \geq c_i + P_{jk} + S_{ijk} - B \cdot (1 - x_{ijk}) \quad \forall k \in M, \forall i, j \in J_k, i \neq j \tag{9}$$

$$c_0 = 0 \tag{10}$$

$$c_j \geq 0, t_j \geq 0 \tag{11}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in J, i \neq j, k \in M_i \cap M_j, \tag{12}$$

$$y_{jk} \in \{0, 1\} \qquad \forall j \in J, k \in M_j \tag{13}$$

Constraints (3) and (4) impose that each job assigned to a machine must be sequenced on that machine. Specifically, it must have a predecessor and a successor on the machine. Constraint (5) guarantees that each job is assigned to a single machine among the ones eligible to process it. Constraint (6) imposes that, at most, a single job is the first one scheduled on each machine. Constraint (7) defines the lower bound for the job completion time, and Constraint (8) defines the job tardiness. Constraint (9) controls the job completion times, ensuring that each machine processes one job at a time and the setup time between two successive jobs is satisfied. Constraint (10) fixes the completion time for the dummy job_0, and Constraints (11), (12) and (13) define the problem decision variables.

3 Definition of Incidence

An "incidence" is an unpredictable event that transforms the original schedule into a non-executable plan, such as: breakdowns on machines, power cuts, work accidents, etc. The

representation for an incidence is independent of its origin and nature. This means that the representation of an incidence only has information about its technical characteristics ignoring the reason for the incidence. An incidence can be formally defined as:

$$i = [id_i, st_i, m_i, tts_i]$$

where

- id_i: an identification number for the incidence
- st_i (starting time): the time point when the incidence *i* starts
- m_i (machine): an integer representing the ID of the affected machine by *i*
- tts_i(time to solve): the necessary time to solve the incidence *i*. It has to be estimated by a human expert. From st_i to $st_i + tts_i$, the machine *m* will be inoperative.

Notice that, with this formalization, an incidence can only affect a single machine, because its representation has only one integer to indicate the machine id. Thus, if there exists an incident that affects more than one machine (e.g: a global power cut) it has to be represented as a set of incidences, one per each affected machine. This representation linked to one single machine allows to model an incidence as a normal job with the constraint that has to be scheduled in the exact moment that it happens.

4 System Proposal

4.1 Assumptions

Some considerations must be taken into account when a incidence is inserted into the system and a rescheduling is carried out. These considerations define the rules (or constraints) to be satisfied during the execution of the rescheduling techniques. They model some real-world conditions related to the specific production environment (air injection presses) that cannot be changed. Concretely, the considerations to take into account are:

- Preemptive tasks: all jobs can be interrupted during its execution by an incidence. When the incidence is solved and the interrupted job recovers its execution it can continue the process from the point it was interrupted. This means that the process carried out by a job will not be lost if the job is force to stop by an incidence.
- Static interrupted-jobs: once an incidence is solved, the affected job must continue its execution in the same machine it was interrupted. Furthermore, the interrupted job must be the first job to be executed after the recovery. The remaining jobs must be rescheduled without these constraints.

4.2 Example problem

Let's analyze an example with the parameters shown in tables 1 and 2. A possible solution for this instance is shown in figure 1. Let's also suppose, as shown on figure 1, that during execution of the job j2 an incidence (specifically, incidence i = [0, 6, 1, 4]) occurs on the machine 1 and it will be inoperative during a period of 4 units of time.

Table 1: Example: possible machines

Job j	M_j		Job j	M_j
1	{1,3}		7	{2}
2	{1,2}		8	{3}
3	{1,2,3}		9	{1,3}
4	{1}		10	{1,2,3}
5	{2,3}		11	{2,3}
6	{1,2,3}		12	{1,2,3}

In this situation, we propose 3 different rescheduling techniques to be analyzed and compared: a baseline that consists on the propagation of the incidence without interaction with other machines, a Greedy-Rescheduling (GR) algorithm based on the Worst Local Job strategy presented in (Nicolo et al. 2018) and, a Genetic Algorithm (GA) extracted from (Nicolo et al. 2017) that is combined with the GR algorithm by initializing the population from the GA with the GR strategy in order to improve its results.

4.3 Proposed baseline

The proposed baseline is based on the propagation of the incidence along the schedule on the machine affected by the incidence using a Right-Shift technique (Vieira, Herrmann, and Lin 2003). This baseline models the behavior of waiting until broken machine is repaired without any other actions nor intelligent response that manage this situation. As shown in figure 2, this technique causes that the whole set of jobs scheduled after the incidence to be delayed, worsening system performance and delaying the end of the affected machine the same time needed to repair the incidence. Notice that the delay introduced by the incidence may increase the tardiness of the delayed jobs affecting directly to the multi-objective function (eq. 2). We will use this baseline to compare the performance of our proposals, which implement a more intelligent way of tackling these situations.

Table 2: Example: processing times

	Job j											
	1	2	3	4	5	6	7	8	9	10	11	12
P_{j1}	2	4	5	2	-	3	4	-	4	3	-	4
P_{j2}	-	3	5	-	4	2	-	-	-	5	4	4
P_{j3}	4	-	8	-	3	4	-	3	2	4	3	5

4.4 Greedy rescheduling (GR) algorithm

In contrast to the baseline, a new reallocation approach is proposed. It analyzes every affected job to determinate if it has to be moved to another machine and, if applicable, what is the best machine and in which position to introduce this job. An "affected job by an incidence" is defined as "Every job scheduled in the same machine in which the incidence occurs and its execution time is scheduled after the

Figure 1: Possible given solution before the incidence

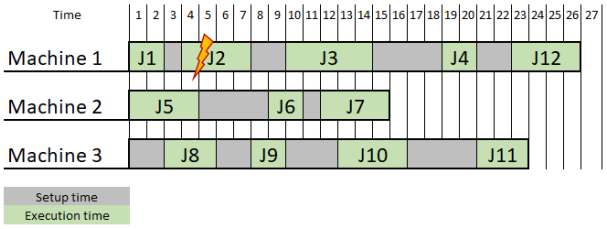
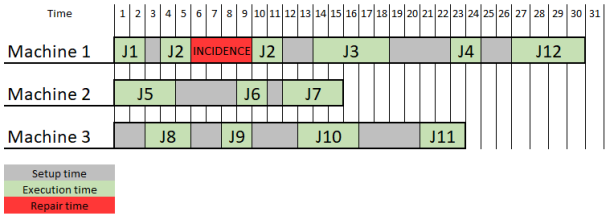


Figure 2: Proposed baseline



incidence starts". Formally, the set of affected jobs (AJ) by an incidence *i* is defined as:

$$AJ = \{j : y_{jm_i} = 1 \wedge st_j > st_i\} \quad (14)$$

Thus, for example, the affected jobs by the incidence in figure 2 are {J3, J4, J12}. Notice that the job interrupted by the incidence (j2 in the example) is not considered to be an affected job since it cannot be rescheduled or moved to another machine by any technique and it must be the first job to execute after the recovery (section 4.1).Also notice that, probably exist some jobs that, in practice, were not delayed by the incidence but they are being considered as *affected-Jobs*. The reason is that the main objective is always to minimize the objective function (eq. 2) and not to recover the original schedule since the original schedules are not supposed to be optimal.

For each affected job, it has to be analyzed if it is convenient to move it to another machine or is more productive to wait until the machine is repaired. Thus, the Best Available Machine (BAM) for each job is determined. If the BAM of a job is the machine in which it is previously scheduled, the job will not be moved, otherwise the job will be moved to its BAM.

Given a job j already scheduled in a machine k', we need to do the next calculations to determinate its BAM. First we need to study how the rest of machines improve or worsen their performance when receiving job j:

$\forall j \in AJ$ do

$$\Delta_k = F(S_k + j) - F(S_k), \forall k \neq k' \in M_j \quad (15)$$

where:

- M_j: is the set of machines that can execute job j (see section 2).
- S_k: is the schedule of machine k. S_k is a sorted list including all the jobs assigned to machine k. Formally:

$$S_k = \{j : y_{jk} = 1\} \quad (16)$$

- $F(S_k)$: is the multi-objective value resulting from the application of the evaluation function (eq. 2) on the schedule S_k.
- $S_k + j$: is the resulting schedule from inserting the job j into schedule S_k in the best position inside S_k (determined by brute force). In this case, the symbol '+' represents the operator of adding a new job into an existing schedule into the best position inside it.
- Δ_k: represents how much the evaluation of the schedule S_k is worsened when job j is added into it.

Each Δ_k stores how much each machine k is deteriorated when receiving the job j, so it can be selected the best machine k* with lower value:

$$k^* = argmin_{k \neq k'} \Delta_k \quad (17)$$

In this way, k^* is selected as the best machine to receive job j, but j is already in a machine k' that could event be better than k^*. To evaluate how job j fits on k', j is taken out from S'_k and an evaluation of how the machine k' performances without j is carried out by:

$$\Delta_{k'} = F(S_{k'}) - F(S_{k'} - j) \quad (18)$$

where:

- $S_{k'} - j$: is the resulting schedule from extracting the job j from schedule S_k.

So, $\Delta_{k'}$ stores how much machine k' deteriorates its performance due to job j. So, finally, the BAM for a given job j can be defined as:

$$BAM = k \in \{k', k^*\} : k = argmin \Delta_k \quad (19)$$

If BAM is *k'* means that *j* is already in its best machine and it does not need to be reallocated in other machine to produce better results. By contrast, if BAM is k^* means that moving job *j* into k^* will produce a better performance ($\Delta_{k^*} < \Delta_{k'}$).

Given this BAM definition, algorithm 1 presents the algorithm used by the system to manage the incoming incidences into an existing schedule:

Algorithm 1: Rescheduling algorithm

INPUT: solution S and incidence I
OUTPUT: solution S with I inside and affected jobs by I reallocated

1: id ← I[0]; st ← I[1]; m ← I[2]; tts ← I[3]; # see section 3
2: affectedJobs ← $\{j : y_{jm} = 1 \wedge st_j > st_{id}\}$
3: Right-Shift(I,S) # Inserts incidence. See figure 2
4: **for** `job ∈ affectedJobs` **do** {
5: $\Delta_k = F(S_k + j) - F(S_k), \forall k \neq m \in M_j$
6: $k^* = argmin_{k \neq m} \Delta_k$
7: $\Delta_m = F(S_m) - F(S_m - j)$
8: $BAM = k \in \{m, k^*\} : k = argmin \Delta_k$
9: if (BAM ≠ m) {moveJob(job, BAM) }
10: }

Line 3 implements the Right-Shift repairing technique (Vieira, Herrmann, and Lin 2003). That means to apply the baseline before rescheduling starts in order to take into account the delay introduced by the incidence when evaluating the reallocation possibilities for each job. In line 9, the *moveJob* function moves the job to its BAM, both elements received as parameters. The position for the job inside its BAM is selected by brute-force: trying one by one all the possible places. Formally, the index *i* to introduce the job *j* into its BAM is selected as:

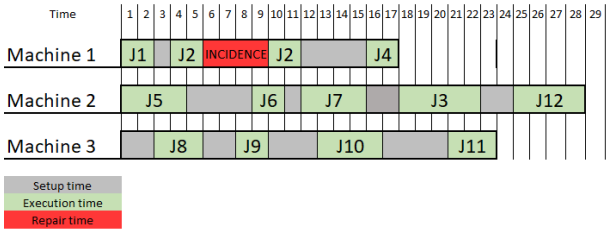
$$i = argmin_{0 \leq i < |S_{BAM}|} F(S_{BAM} + j, i) \quad (20)$$

where:

- S_{BAM} is the current schedule for machine BAM.
- $F(S_{BAM} + j, i)$ is the application of the evaluation function (eq. 2) to the schedule S_{BAM} with the job *j* introduced in the position *i* by Right-Sift technique.

Figure 3 shows a possible solution for the incidence presented in figure 1. As shown, using a reallocation strategy (e.g.:GR) the system only needs to use 28 units of time (UoT) instead of 30 UoT that needs the Right-Shift strategy showed in figure 2.

Figure 3: Proposed solution



4.5 A Genetic Algorithm with GR initialization (GA+GR)

In this section, a Genetic Algorithm (GA) for rescheduling a solution after an incidence is proposed. The main idea is to run a GA only with the affected jobs (AJ) (eq. 21) in order to preserve the solution before the incidence and trying to reschedule all the jobs after the incidence with the objective of minimizing the impact of the disruption. Notice that incidences are supposed to appear in the production environment while the scheduling is being already executed by the machines, therefore the decision to keep the previous jobs without rescheduling them is not really a decision but a constraint of the environment. A competitive GA to tackle this problem has been used in (Nicolo et al. 2017), where the GA is embebed in a multi-agent system. Taking this GA as a starting point and our definition of AJ, some modifications to them have been carried out:

- In order to reschedule the whole set of jobs that can be moved after the incidence, a new definition of AJ is presented:

$$AJ = \{j : st_j > st_i\} \quad (21)$$

With this new definition, the whole set of jobs executed after the incidence is selected for the rescheduling, independently of the machine in which the job is scheduled, which allows the GA to explore a larger search space than the GR algorithm. The main idea behind this choice is to be able to compare two options of general rescheduling approaches: to preserve as much as possible the previous solution (stability) by rescheduling only the jobs of the affected machine (done by GR) or to reschedule the whole solution after the incidence (done by this modification of GA).

- The *fitness* function of the GA is modified to contemplate the previous schedule to the solutions given by the GA in order to calculate the multi-objective function (eq. 2) with the complete schedule. Notice that GA is executed only with the AJ as input so the schedule given as solution only contains the AJ. Nevertheless, the evaluation of the solution needs the previous schedule (the one containing the *non affected jobs*) in order to decide the quality of a solution. In this way, it is achieved that the GA solves a sub-problem of scheduling (only the problem containing the AJ) but it evaluates its solutions with respect the complete schedule
- The initialization of the GA is modified to create an initial population which includes the solution given by the GR algorithm. The incidences will be solved accumulatively (see section 5), which implies that as incidences appear, the GR and the GA may have different solutions for each incidence. However, in the first disruption, both algorithms starts in the same state. Only in this first incidence, the GA includes the solution given by the GR as an individual of its initial population. This decision is given to guide the GA search with the previous solutions which allows to study if this solutions is replaced during the process by another better solutions or it is preserved during the whole process.

5 Evaluation

To evaluate the proposed system, a set of incidences was generated. The incidences were randomly generated following some rules:

- m_i: The machine affected by the incidence was randomly selected among all the machines involved in the problem.
- st_i: The starting time of the incidence was randomly selected before the 75% of the total time in the schedule S_{m_i}. Formally:

$$st_i = random(0, 0.75 * |S_{m_i}|)$$

This decision avoids the generation of incidences at the end of the schedule because when an incidence is introduced at the end of the schedule the set *AJ* may be empty or composed of few jobs. This situation does not provide an effective environment to carry out an objective comparison of the different techniques.

- tts_i: The necessary time to solve the incidence was randomly generated between 10% of the longest job LJ as-

signed to m_i and its total processing time P_{jm_i}. Formally:

$$LJ = argmax_{j \in J} P_{jm_i} : y_{jm_i} = 1$$

$$tts_i = random(0.1 * P_{jm_i}, P_{jm_i})$$

where *random(a,b)* is a function that returns a random number in the interval [a,b).

The incidences were solved accumulatively, which means that the input solution in which the incidence *i+1* was introduced was the schedule with the incidence *i* already introduced in it. By accumulating the incidences in the schedule it can be compared how the different techniques absorb the incoming incidences as new disruptions arrive into the system.

Algorithm 2 shows the algorithm to compare all techniques. Notice that the proposed techniques are rescheduling techniques but not solving techniques so they need an initial solution and an incoming incidence to operate. The initial solutions were taken from the system proposed in (Nicolo et al. 2018) which present a good set of solutions for the given problem but it can not be used with rescheduling purposes for two reasons:

- It can not deal with the hard constraint associated to the starting time of the incidence.
- It is not prepared to receive a solution as input and carry out from this solution a local search. Instead of this, the system receives a specification input of the problem and executes a global search with all the combinatorial possibilities.

These limitations are solved in the proposed modification (section 4.5) resulting on the GA+GR algorithm that is evaluated in this section.

Algorithm 2: Rescheduling algorithm

INPUT: a solution S and set of incidences I
OUTPUT: solution S with I solved by 4 different techniques

```
1: GA_solution ← S
2: baseline_solution ← S
3: GR_solution ← S
4: GA+GR_solution ← S
5: for i ∈ I do {
6: GA_solution ← GA(i, GA_solution)
7: baseline_solution ← BL(i, baseline_solution)
8: GR_solution ← GR(i, GR_solution)
9: GA+GR_solution ← GA+GR(i, GA+GR_solution)
10: }
11: return baseline_solution, GA_solution, GR_solution, GA+GR_solution
```

Lines 1 to 4 copy the original solution to be used for each technique. Line 5 declares the loop that will accumulatively solve the incidences. Function $GA(i, s)$ in line 6 is a function that solves the incidence i on solution s by applying the GA proposed in (Nicolo et al. 2017) without the GR technique proposed in this paper. Functions BL, GR and $GA + GR$ in lines 7-9 are functions that implements the rescheduling techniques proposed in sections 4.3, 4.4 and 4.5 respectively.

The instances of the problem were taken from the benchmark proposed in (Tonelli, Evans, and Taticchi 2013) which presents 4 classes of instances depending on the number of jobs and machines per incidence. Table 3 shows the different descriptions for each class of instances.

Table 3: Benchmark instances

Instance	No. machines	No. jobs
j30	4	30
j50	6	50
j100	10	100
j250	20	250

To test and compare the four techniques, a subset of the 10 largest instances per class were selected (40 instances). They were solved with the system proposed in (Nicolo et al. 2018) and the solutions given by this system were used as the input for our system, described in algorithm 2. Table 4 shows the average needed time per instance by each technique to be executed (table shows values from j250; smaller instances needed proportional time to their size measured in terms of No.jobs*No.machines). GA (Nicolo et al. 2017) and GA+GR(section 4.5) have been both fixed to 30 seconds timeout, which presents a reasonable time in the environment context of the problem.

Table 4: Execution times

Algorithm	Time (s)
baseline	0.2
GR	2.3
GA	30
GA+GR	30

Figures 4, 5, 6, 7 show the results of applying the four techniques to each class of the problem. In order to combine the multi-objective nature of the problem with the performance of our system, all solutions are evaluated by using the multi-objective function (eq. 2) to minimize. For this, it is necessary to set the values of weights π_1, π_2 and π_3 from eq. (2). These values where fixed to 0.6, 0.35 and 0.05 respectively, values that come from (Nicolo et al. 2018), where authors carry out a study of what values can provide better results. As shown on figure 4, for small instances the GR algorithm obtained the best results, but as problem gets larger, the GA+GR algorithm got a better behavior than the rest of the techniques. It is also interesting to compare GA vs GA+GR since GA+GR maintained a better performance that GA in all cases, although both were very similar. This is due to the fact that the only difference between them is that GA+GR includes an initialization with the greedy algorithm GR. This difference could came from the fact that GA was designed in (Nicolo et al. 2017) to have a good performance in solving (not rescheduling) the instance and with a larger timeout (10 minutes) than the timeout fixed for the

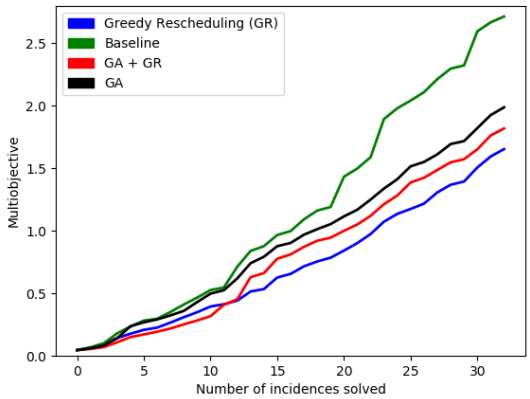


Figure 4: Results on j30

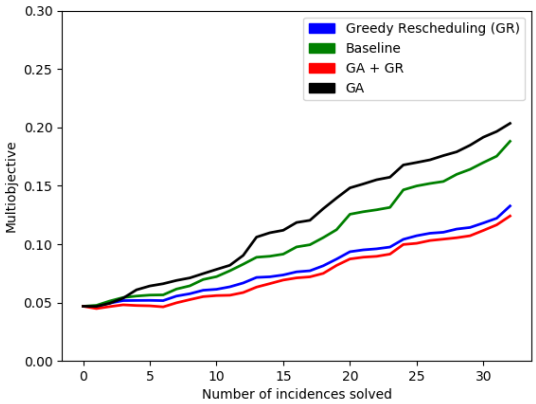


Figure 6: Results on j100

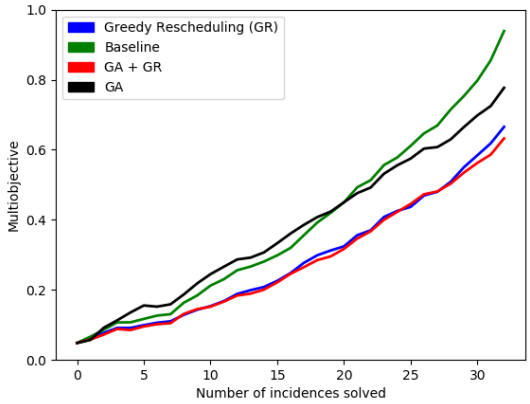


Figure 5: Results on j50

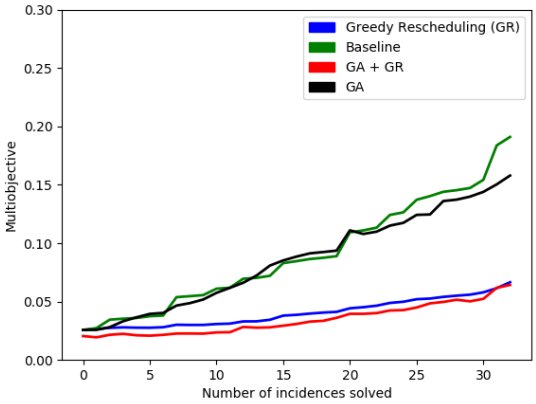


Figure 7: Results on j250

rescheduling tasks. It can also be observed that GA was in some cases even worse than the baseline (figures 5 and 6). This effect could be explained taking into account that the baseline technique try to maintain the previous solution (robustness), meanwhile GA does not respect the previous solution but executes the complete GA algorithm with random initialization.

6 Conclusions and Future Works

In this paper, two different techniques for solving the rescheduling on the unrelated parallel machine problem have been proposed. The proposed GR algorithm obtained the best results in small instances of the problem. However, as the instances were larger, the GA+GR algorithm obtained a better behavior than the rest of the techniques. Indeed, high quality improvements have been achieved to the existing algorithm (Nicolo et al. 2017) designed to solve this problem, due to the GA+GR algorithm proposed in this paper got better results in large instances than GA. As future work, some improvements to the GA+GR algorithm can be carried out, such as executing the GR algorithm on each incidence and not only in the first incidence. Furthermore, it is proposed to manage the incidences by a match-up technique to recover the original solution as soon as possible in order to improve stability of solutions.

Acknowledgements

The paper has been partially supported by the Spanish research project TIN2016-80856-R and TIN2015-65515-C4-1-R, and the ACIF/2017/061 Program of the Conselleria d'Educació (Generalitat Valenciana).

References

Arnaout, J.-P. 2014. Rescheduling of parallel machines with stochastic processing and setup times. *Journal of Manufacturing Systems* 33(3):376–384.

Bruzzone, A.; Anghinolfi, D.; Paolucci, M.; and Tonelli, F. 2012. Energy-aware scheduling for improving manufacturing process sustainability: a mathematical model for flex-

ible flow shops. *CIRP Annals-Manufacturing Technology* 61(1):459–462.

Dai, M.; Tang, D.; Giret, A.; Salido, M. A.; and Li, W. D. 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing* 29(5):418–429.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K.; and Kan, A. R. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics* 5:287–326.

Hall, N. G., and Potts, C. N. 2004. Rescheduling for new orders. *Operations Research* 52(3):440–453.

Le, C. V., and Pang, C. K. 2013. Fast reactive scheduling to minimize tardiness penalty and energy cost under power consumption uncertainties. *Computers & Industrial Engineering* 66(2):406–417.

Nicolò, G.; Salido, M.; Giret, A.; and Barber, F. 2016. Assessment of a multi agent system for energy aware offline scheduling from a real case manufacturing data set. *COPLAS 2016* 35.

Nicolo, G.; Salido, M. A.; Ferrer, S.; Giret, A.; and Barber, F. 2017. A multi-agent approach using dynamic constraints to solve energy-aware unrelated parallel machine scheduling problem with energy-dependent and sequence-dependent setup time. In *Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems. 27th International Conference on Automated Planning and Scheduling June 19-23, 2017, Pittsburgh, USA*, 31–37.

Nicolo, G.; Ferrer, S.; Salido, M. A.; Giret, A.; and Barber, F. 2018. A multi-agent framework to solve energy-aware unrelated parallel machine scheduling problems with machine-dependent energy consumption and sequence-dependent setup time. *Main track from 28th International Conference on Automated Planning and Scheduling June 24-29, 2018, Delft, The Netherlands*.

Nouiri, M.; Bekrar, A.; Jemai, A.; Ammari, A. C.; and Niar, S. 2018. A new rescheduling heuristic for flexible job shop problem with machine disruption. In *Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer. 461–476.

Paolucci, M.; Anghinolfi, D.; and Tonelli, F. 2015. Facing energy-aware scheduling: a multi-objective extension of a scheduling support system for improving energy efficiency in a moulding industry. *Soft Computing* 1–12.

Plitsos, S.; Repoussis, P. P.; Mourtos, I.; and Tarantilis, C. D. 2017. Energy-aware decision support for production scheduling. *Decision Support Systems* 93(Supplement C):88 – 97.

Qi, X.; Bard, J. F.; and Yu, G. 2006. Disruption management for machine scheduling: the case of spt schedules. *International Journal of Production Economics* 103(1):166–184.

Tonelli, F.; Bruzzone, A.; Paolucci, M.; Carpanzano, E.; Nicolo, G.; Giret, A.; Salido, M.; and Trentesaux, D. 2016. Assessment of mathematical programming and agent-based modelling for off-line scheduling: Application to energy aware manufacturing. *CIRP Annals - Manufacturing Technology* 65:405–408.

Tonelli, F.; Evans, S.; and Taticchi, P. 2013. Industrial sustainability: challenges, perspectives, actions. *International Journal of Business Innovation and Research* 7(2):143–163.

Vieira, G. E.; Herrmann, J. W.; and Lin, E. 2003. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of scheduling* 6(1):39–62.

Wang, S.; Liu, M.; Chu, F.; and Chu, C. 2016. Bi-objective optimization of a single machine batch scheduling problem with energy cost consideration. *Journal of cleaner production* 137:1205–1215.