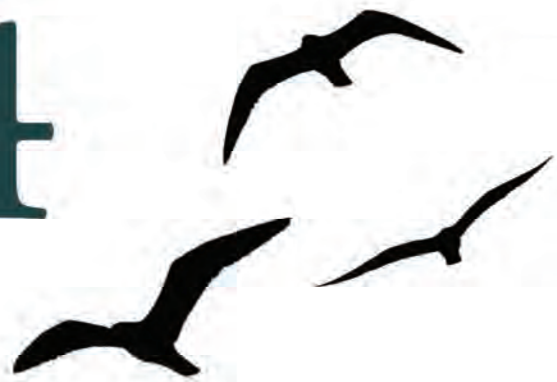
ICAPS 2014



icaps

Proceedings of the 9th Workshop on

Constraint Satisfaction Techniques for Planning and Scheduling

Edited By:
Miguel A. Salido and Roman Barták

Portsmouth, New Hampshire, USA - June 22, 2014

Organizing Committee

Miguel A. Salido
Universidad Politécnica de Valencia, Spain

Roman Barták
Charles University, Czech Republic

Program Committee

Federico Barber, Universidad Politecnica de Valencia, Spain
Roman Bartak, Charles University, The Czech Republic
Amedeo Cesta, ISTC-CNR, Italy
Minh Binh Do, NASA Ames Research Center, USA
Agostino Dovier, Universita Degli Studi di Unide, Italy
Enrico Giunchiglia, Universita di Genova, Italy
Eva Onaindia, Universidad Politecnica de Valencia, Spain
Nicola Policella, European Space Agency, Germany
Hana Rudova, Masaryk University, The Czech Republic
Migual A. Salido, Universidad Politecnica Valencia, Spain
Torsten Schaub, Technical University of Darmstadt, Germany
Dunbing Tang, Nanjing University of Aeronautics & Astronomics, China
Enrico Pontelli, New Mexico State University, USA
Ramiro Varela, Universidad de Oviedo, Spain
Gerard Verfaillie, ONERA, Centre de Toulouse, France
Vincent Vidal, CRIL-IUT, France
Petr Vilim, ILOG, France
Neil Yorke-Smith, American University of Beirut/SRI International, USA
Neng-Fa Zhou, The City University of New York, USA

Foreword

The areas of planning and scheduling in Artificial Intelligence have seen important advances thanks to the application of constraint satisfaction and optimization models and techniques. Especially solutions to real-world problems need to integrate plan synthesis capabilities with resource allocation, which can be efficiently managed by using constraint satisfaction techniques. The workshop will aim 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 will mainly focus 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.

In this edition, five papers were accepted. They represent an advance in the integration of constraint satisfaction techniques in planning and scheduling frameworks. These papers are distributed between theoretical papers such as model-based planning, partial-order planning and job-shop scheduling, and application papers to berth allocation problems and unmanned aerial vehicle activities.

Miguel A. Salido, Roman Barták
COPLAS 2014 Organizers
June 2014

Table of Contents

Revisiting Dynamic Constraint Satisfaction for Model-Based Planning

Jeremy Frank
NASA Ames Research Center
Mail Stop N269-3
Moffett Field, California 94035-1000

Abstract

As planning problems become more complex, it is increasingly useful to integrate complex constraints on time and resources into planning models, and use complex constraint reasoning approaches to help solve the resulting problems. Dynamic constraint satisfaction is a key enabler of automated planning. In this paper we identify some limitations with the previously developed theories of dynamic constraint satisfaction. We identify a minimum set of elementary transformations from which all other transformations can be constructed. We propose a new classification of dynamic constraint satisfaction transformations based on a formal criteria. This criteria can be used to evaluate elementary transformations of a CSP as well as sequences of transformations. We extend the notion of transformations to include optimization problems. We show how these transformations can inform the evolution of planning models, automated planning algorithms, and mixed initiative planning.

Introduction

Automated planning can be posed as a constraint satisfaction problem (CSP), and subsequently solved using CSP techniques (e.g. (Do and Kambhampati 2000), (Vidal and Geffner 2006), (Banerjee 2009)). Automated scheduling also makes extensive use of constraint satisfaction techniques (e.g. (Laborie 2003)). As planning problems become more complex, it is increasingly useful to integrate complex constraints on time and resources into planning models, and therefore use complex constraint reasoning approaches to help solve the resulting planning problems (Jónsson and Frank 2000), (Frank and Jónsson 2003), (Ghallab and Laurelle 1994). Similarly, many planning problems are optimization problems; one variation includes plan quality measured by the set of satisfiable goals (van den Briel et al. 2004). As an intermediate step between satisficing and optimization problems, a problem instance may be modified by the addition or subtraction not only of goals, but the changing, or waiving, of some constraints. These incremental modifications of the problem often takes place in a 'mixed-initiative' setting, where human planners use automation to solve problems, e.g. (Bresina et al. 2005).

During the search process, a plan can be translated into an underlying CSP, on which reasoning (propagation and heuristics computations) are performed. The CSP may be modified by the addition or subtraction of variables, domain values, and constraints. In order to support planning, the underlying CSP machinery must be modified to support *dynamic* constraint satisfaction (DCSP). Previous formalisms for dynamic constraint satisfaction have been developed to support automated planners. However, these formalisms are unsatisfactory for several reasons, as described below.

We propose a new formal criteria to classify dynamic constraint satisfaction transformations, and identify a minimum set of transformations from which all other transformations can be constructed. This criteria can be used to evaluate elementary transformations of a CSP as well as sequences of transformations. We extend the new formalism to include optimization problems, leading to a novel integration of dynamic constraint satisfaction and partial constraint satisfaction. We show how these transformations can inform the evolution of planning models, automated planning algorithms, and mixed initiative planning. The resulting set of simple transformations allows analysis of every modification of CSPs, with and without optimization criteria.

Dynamic Constraint Satisfaction

In this section we briefly review the Dynamic Constraint Satisfaction Problem (DCSP).

The DCSP was originally formalized by (Dechter 1988). In this formalism, all variables have identical domains, and the set of constraints and variables is allowed to vary over time, thereby creating a sequence of problems. The focus in this work was on maintaining one or a set of solutions as the problem is changed. In this formalism, the notion of *restrictions* and *relaxations* is directly associated with specific changes to the CSP. Adding variables and constraints are termed restrictions, and removing variables and constraints are termed relaxations.

A restricted variation of the DCSP was introduced by (Mittal and Falkenhainer 1990). In this formalization the set of variables and constraints is fixed, but the set of constraints limiting the solutions are activated by variable assignments, i.e. all of the variables in the scope of a constraint must be activated for the constraint to apply. The set of active variables (those that must be assigned) can be a function of other variables' values. There is also a minimum subset condition to ensure no 'extra' variables are assigned. Thus, the problem does not change via some external entity adding variable and constraints, but the act of solving 'activates' variables and constraints.

An extension to (Mittal and Falkenhainer 1990) was introduced by (Soininen, Gelle, and Niemelä 1999) to allow disjunctive activity constraints (satisfaction of activity constraints implies some subset of variables must be assigned), include 'null' variable assignment, and change the definition of solution (fixed point / closure instead of minimum subset) to reduce computational complexity.

The Disjunctive Temporal Network introduced by (Tsamardinos and Pollack 2003) is similar to the DCSP of (Mittal and Falkenhainer 1990). Here the focus is on a discrete choice of which temporal constraint to apply to a pair of temporal variables.

An extension to the DCSP to support automated planning was introduced by (Jónsson and Frank 2000) In addition to adding or removing variables and restricting or relaxing constraints, new values for existing variables may be added, or values of variables can be removed from the domains permanently. Adding new constraints and changing the scope of constraints was not considered. This approach is elaborated on in (Frank, Jónsson, and Morris 2000).

We shall now describe some limitations with the current formalisms for DCSPs.

Previously, changes to CSPs were simply labeled restrictions (or relaxations) with no formal criteria to determine which is which. Consider adding a variable without constraints to a CSP. Compared to the previous CSP (without the new variable), a decision is now required in the new CSP where no decision was required before. The resulting transformation is labeled a restriction. However, the label is somewhat unintuitive; a variable with no constraints does not strike one as obviously restrictive. Further, adding a new variable X with domain size ≥ 2 increases the *number of assignments* and the *number of solutions* but not the *percentage of assignments that are solutions*. In fact, the percentage of assignments that are solutions is left unchanged. That is because the new variable's values are *unconstrained* and therefore increase both the number of assignments and the number of solutions by the same constant factor (namely $|d(X)|$). This means classes of change to a CSP should be more carefully evaluated with respect to a formal criteria.

The formalisms of DCSPs have not been extended to problems with costs and quality bounds. How is a problem of satisfiability transformed into such a problem? Does introducing optimization restrict or relax the problem? Finally, suppose the problem is an optimization problem, and initially solutions below some cost are demanded. Over time, the cost bound is found to be too low, and the problem is changed by either increasing the cost bound, or by reducing (or eliminating) the cost of assignments associated with some constraints. Intuitively, making the cost bound higher might seem like a relaxation. Similarly, making some solutions have higher cost might seem like a restriction. If we are in an optimal setting with a cost bound defining the set of feasible solutions, then increasing the bound will increase the number of solutions, and decreasing cost will only reduce the number of solutions. However, if you reduce how good one or more solutions are, but preserve the set of solutions, is this a restriction or relaxation? If the set of all optimal solutions' costs are reduced, the set of solutions of the next lowest level of quality could be larger or smaller.

A recent line of research known as Model-Lite Planning (S.Kambhampati 2007) contemplates early-stage models for criticism as opposed to planning. A formalism such as the one proposed here could be used for such a purpose. Suppose new constraints are added to some activity as part of the modeling process. Is the change a restriction or relaxation? Is it easy to tell immediately? If the change is not obvious, or perhaps even if it is, informing the modeler of the consequences of such a change at modeling time may be valuable. As just one example, consider transforming a problem in which constraints cannot be waived into one where they can. This is a significant transformation; a new variable must be introduced, the scope of the waivable constraint must be changed, and the relations in the constraint extended. In (Frank and Jónsson 2003) such a change is not considered; while adding constraints was concieved of in (Dechter 1988), a change of scope was not, and must be synthesized from other primitives in the other formalisms. What is the ultimate impact of such a transformation?

Constraint Satisfaction Problems

Definition 1 *Let $X_1 ... X_n$ be a set of variables. The* domain *of variable X_i is denoted $d(X_i)$. Let $x_{i_j} \in d(X_i)$ be a value.*

Definition 2 *A* Constraint *C_j is a tuple S_j, R_j. The* scope *S_j of the constraint is a set of variables $X_{j_1} ... X_{j_k}$. The* relation *$R_j \subseteq d(X_{j_1}) \times ... \times d(X_{j_k})$ is a list of tuples r_{j_h} defining the allowed combinations of values to the variables in the scope of the constraint.*

Definition 3 *A* Constraint Satisfaction Problem *or CSP $\mathcal{P}$ is a set of variables $X_1 ... X_n$ with domains $d(X_1) ... d(X_n)$ and a set of constraints $C_1 ... C_m$. The* projection *of an assignment $x \in d(X_1) \times ... \times d(X_n)$ onto a set of variables $\mathcal{X}$, denoted $\pi(x, \mathcal{X})$, is the value of each variable in $\mathcal{X}$. An assignment is a solution if its projection onto the scope of each constraint C_i is in the relation R_i, that is, if $\forall C_i\, \pi(x, S_i) \in R_i$.*

Throughout the remainder of this paper, we assume CSPs are all finite discrete domain.

Dynamic Constraint Satisfaction Problems: A New Formalism

Definition 4 *A* transformation *τ is a function that maps a CSP $\mathcal{P}_i$ to a CSP $\mathcal{P}_j$ denoted $\mathcal{P}_i \xrightarrow{\tau} \mathcal{P}_j$. Let n be the number of variables in $\mathcal{P}_i$. Let m be the number of constraints in $\mathcal{P}_i$. Let $d_i = |d(X_i)|$. Let $c_j = |R_j|$. The set of classes of transformations is:*

1. *Add X_{n+1} to $\mathcal{P}_i$ with $|d(X_{n+1})| = 1$. Denote this transformation $X+$.*
2. *Remove X_j from $\mathcal{P}_i$ with $|d(X_j)| = 1$ s.t. $\forall C_k\, X_j \notin S_k$. Denote this transformation $X-$.*
3. *Add a unique value $x_{i_{d_i+1}}$ to $d(X_i)$ s.t. $\forall C_k\, X_i \notin S_k$. Denote this transformation $d+$.*

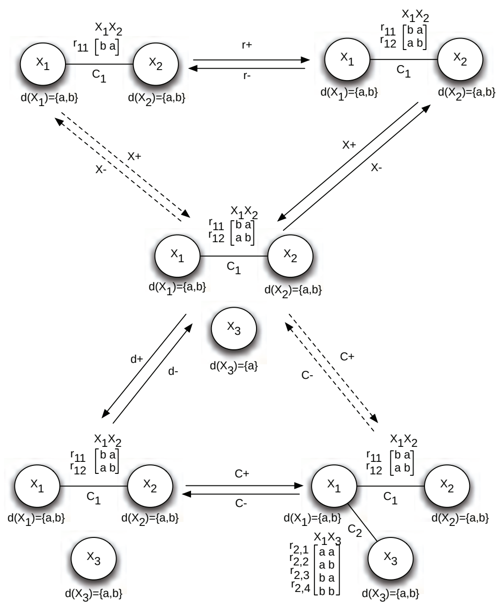


Figure 1: Transformations. The set of all valid DCSP transformations. (The dotted lines show valid transformations, but not the exact transformations of the DCSP in the figure.)

4. *Remove a value x_{i_j} from $d(X_i)$ s.t. $\forall C_k, X_i \notin S_k$. Denote this transformation $d-$.*
5. *Add a unique tuple $r_{j_{c_j+1}}$ to R_j Denote this transformation $r+$.*
6. *Remove a tuple r_{j_h} from R_j. Denote this transformation $r-$.*
7. *Add C_{m+1} with $S_{m+1} = X_{m+1_1}...X_{m+1_k}$ and relation $R_{m+1} = \times_{i=1}^{k} d(X_{m+1_k})$ to $\mathcal{P}_i$. Denote this transformation $C+$.*
8. *Remove C_j with $R_j = \times_{i=1}^{k} d(X_{j_k})$ from $\mathcal{P}_i$. Denote this transformation $C-$.*

These transformations are shown graphically in Figure 1.

Definition 5 *Let $L(\mathcal{P})$ be the number of solutions to $\mathcal{P}$. The* fraction of solutions *denoted $L_p(\mathcal{P})$ is then $\frac{L(\mathcal{P})}{\prod_{i=1}^{n} |d(X_i)|}$.*

Definition 6 *Let τ be a transformation from $\mathcal{P}_i \xrightarrow{\tau} \mathcal{P}_j$. A* relaxation *increases the fraction of solutions, i.e. $L_p(\mathcal{P}_i) < L_p(\mathcal{P}_j)$; a* restriction *decreases the fraction of solutions, i.e. $L_p(\mathcal{P}_i) > L_p(\mathcal{P}_j)$. A* neutral *transformation is neither a restriction or a relaxation.*

Unlike previous theories of transformations on DCSPs, the classes transformations on PCSPs cannot all be classified as restrictions, relaxations, or neutral:

Theorem 1 *There are classes of transformations that are restrictions, relaxations, and neutral. Furthermore, for a DCSP $\mathcal{P}_i$, there are classes of transformations that can be either restrictions or neutral, and there are classes of transformations that can be relaxations or neutral.*

The proof employs straightforward analysis of each class of transformation:

1. Since $|d(X_{n+1})| = 1$, $L(\mathcal{P}_j) = (L(\mathcal{P}_i))(|d(X_{n+1})|) = L(\mathcal{P}_i)$. Similarly, $(\prod_{i=1}^{n} |d(X_i)|)(|d(X_{n+1})|) = \prod_{i=1}^{n} |d(X_i)|$. Adding a variable with a single value but not adding this value to any relations leaves the number of assignments and solutions unchanged, so the fraction of solutions is unchanged.
2. Removing a variable with a single value that participates in no constraints leaves the number of assignments and solutions unchanged, so the fraction of solutions is unchanged.
3. Adding a value to a variable but leaving the relations unchanged increases the number of assignments and the number of solutions by the same factor, leaving the percentage of solutions unchanged. Let $d+$ change $d(X_n)$ w.l.o.g. (simplifies notation for the computation). Note $\frac{a(b+1)}{ab} = \frac{b+1}{b}$. Then

$$\frac{(\prod_{i=1}^{n-1} |d(X_i)|)((|d(X_n)|+1))}{(\prod_{i=1}^{n-1} |d(X_i)|)(|d(X_n)|)} = \frac{|d(X_n)|+1}{|d(X_n)|}$$

which means

$$(\prod_{i=1}^{n-1} |d(X_i)|)((|d(X_n)|+1|))$$

$$= (\prod_{i=1}^{n-1} |d(X_i)|)(|d(X_n)|)\left(\frac{|d(X_n)|+1}{|d(X_n)|}\right)$$

Next, recall values can only be added to variables not in the scope of any constraint. So rewrite $L(\mathcal{P}_i) = (L(\mathcal{Q}_i))(|d(X_n)|)$. Now

$$\frac{(L(\mathcal{Q}_i))(|d(X_n)+1|)}{(L(\mathcal{Q}_i)(|d(X_n)|)} = \frac{|d(X_n)|+1}{|d(X_n)|}$$

which means

$$(L(\mathcal{P}_i)\left(\frac{|d(X_n)|+1}{|d(X_n)|}\right) = L(\mathcal{P}_j)$$

Finally,

$$L_p(\mathcal{P}_j) = \frac{L(\mathcal{P}_j)}{(\prod_{i=1}^{n-1}(|d(X_i)|)(|d(X_n)|+1)}$$

$$= \frac{\left(\frac{|d(X_n)|+1}{|d(X_n)|}\right)L(\mathcal{P}_i)}{\left(\frac{|d(X_n)|+1}{|d(X_n)|}\right)\prod_{i=1}^{n}|d(X_i)|}$$

$$= \frac{L(\mathcal{P}_i)}{\prod_{i=1}^{n}|d(X_i)|} = L_p(\mathcal{P}_i)$$

.

4. Removing a value from a variable domain leaves the fraction of solutions unchanged as argued above.

5. Adding a tuple to a relation may not increase the number of solutions, as the tuple may be excluded by other relations. However, adding a unique tuple to a relation cannot decrease the number of solutions. The number of assignments does not change, so the fraction of solutions cannot decrease.

6. Removing a unique tuple from a relation cannot increase the number of solutions. The number of assignments does not change, so the fraction of solutions cannot increase.

7. Adding a constraint with the relation consisting of all assignments to the variables in the scope leaves the fraction of solutions unchanged.

8. Removing a constraint whose relation consists of all assignments to the variables in the scope leaves the fraction of solutions unchanged.

Definitions 5 and 6 and can now be used to classify a single transformation, and also to evaluate a sequence of transformations.

As an aside, the rule on the addition of variables with one value in the domain and the removal of variables with one value in the domain ensure the calculation of the number of assignments and solutions remains well-formed (i.e. no multiplications by zero for the number of assignments!) The addition and removal of constraints with no tuples in the relation has similar technical restrictions. The rule on changing domains of variables involved in no constraints ensures the CSP remains well-formed; removing a value of a variable requires removing all relations involving the value, which is cumbersome.

Let us compare this set of transformations and their classification as restrictions and relaxations to previous definitions of restrictions and relaxations. First, we have a principled definition of restriction and relaxation in terms of the fraction of assignments that solve the transformed CSP. As mentioned, this definition applies to both single transformations and sequences of transformations. Next, we have finer-grained and more precisely characterized transformations than those identified previously. We see that transformations need not be restrictions or relaxations, and that some transformations previously identified as restrictions are, in fact, not necessarily characterized this way in the new classification. For example, adding a variable is considered a *restriction* in (Dechter 1988) but is neither a restriction nor a relaxation according to the new classification.

Theorem 1 holds regardless of whether the definition of restriction or relaxation uses the number or fraction of solutions. This is summarized by the table below. Recall from defintion 5 that $L(\mathcal{P})$ is the number of solutions, and $L_p(\mathcal{P})$ is the fraction of solutions. In the table below we denote the number of assignments $\prod_{i=1}^{n} |d(X_i)|$ by $\mathcal{A}$.

τ	$\Delta L_p(\mathcal{P})$	$\Delta L(\mathcal{P})$	$\Delta\mathcal{A}$
$X+$	0	0	0
$X-$	0	0	0
$d+$	0	>0	>0
$d-$	0	<0	<0
$C+$	0	0	0
$C-$	0	0	0
$r+$	≥ 0	≥ 0	0
$r-$	≤ 0	≤ 0	0

The table shows that only a small number of transformations actually change the set of solutions in a meaningful way. Consider the transformation $\tau = d+$ (i.e. adds values to $d(X_i)$). Since $X-_i$ can't be involved in any constraints, no relations in existing constraints are modified, and no constraints are added, all of the new values participate in solutions to $\mathcal{P}_j$. However, every solution to $\mathcal{P}_i$ is a solution to $\mathcal{P}_j$. In a sense, the transformation is *trivial*, in that little work is required to keep up with this 'restriction'. Similarly, the transformation $d-$ (removing a value) is not in any relations, reduces the number of assignments, and the solutions are reduced by the same fraction. The only non-trivial transformations are those in which the set of tuples in an existing relation are modified.

It is possible to synthesize the previously defined restrictions and relaxations from a sequence of these new, primitive transformations. For instance, adding a variable X_{n+1} with domain size d_{n+1} takes d_{n+1} transformations: one addition of the variable with domain size 1, followed by $d_{n+1} - 1$ additions of values to the domain. The overall effect of the sequence is neutral, since each transformation $d+$ is neutral, and the variable addition is neutral. Next, consider changing the relation R_j of constraint C_j to R'_j such that $|R'_j| < |R_j|$. This restriction can be done by a sequence of transforms $r-, r+$. By ordering the adding of values and the changing of the relation, an existing constraint can be arbitrarily modified as proposed in (Jónsson and Frank 2000). Finally, adding new variables and an arbitrary number of constraints on these variables is accomplished by first adding the variables, then filling out their domains, then adding the trivial constraints with all elements of the relation allowed, and finally removing the invalid combinations from each relation. Such complex transformations could be relaxations, restrictions, or neither, depending on the ultimate modification of the relation.

In addition, a variety of new 'macro-transformations' can be defined. For instance, consider changing a constraint C_k to C'_k such that the scope of the constraint S_k is changed to S'_k. Assume all of the variables added to the new scope are already present in the CSP. A succession of transformations on the relation $r+$ must be performed to make C_j the trivial relation on the old scope. Then C_j can be removed. Next, the trivial relation C'_j is added on the correct scope. Finally, a series of transformations $r-$ are performed to eliminate the invalid tuples. The resulting sequence can be either a restriction or relaxation, depending on the new relation.

We provide a specific example in figure 2 of adding a variable to the scope of a constraint to support waiving the orig-

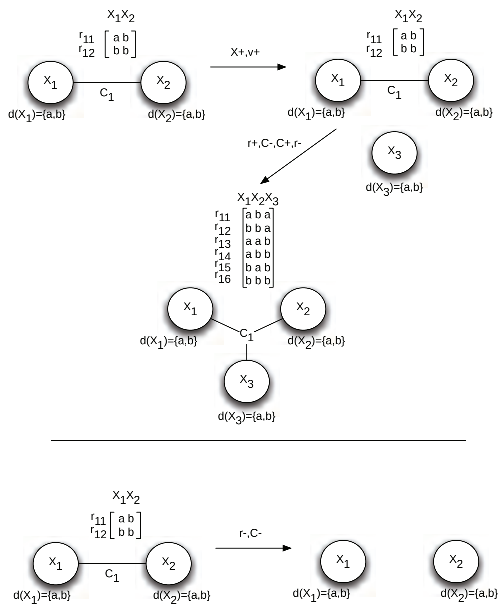


Figure 2: Extending a constraint to allow waiving violations by variable assignment during solving (top) versus removing a constraint (bottom).

inal constraint. In this figure some of the transformations have been combined for brevity. This transformation is contrasted to the transformation of simply removing a constraint from the variables.

Planning Using DCSPs

In this section we describe how this formalization of DCSPs can be used during planning. As described in the introduction, some planners use a DCSP framework during search. We consider whether this new DCSP formalization should lead to revisiting the design of those frameworks. In addition, we describe how the DCSP formalism can be used to assist in the modeling process.

During Search

Generally, each time a new action is added, there are one or more variables to represent the choices of how the action is added to the plan. For instance, there could be a single variable for each action; the values indicate whether it is added to the plan or not. There could be a variable for the next action in a sequential plan, whose domain consists of all actions whose preconditions are satisfied. In a partial order setting, or when planning with resources, action ordering may require more variables. Determining whether the resulting problem is a restriction or relaxation of the previous problem requires analyzing the constraints on those new variables.

We saw previously how to transform constraints to make them waivable as shown in Figure 2 (top). Suppose instead we constructed a planner to remove violated constraints detected during search as shown in Figure 2 (bottom). Removing the constraint from the CSP requires a series of $r+$ transformations to make the constraint trivially satisfiable, then removing it using the $C-$ transformation. When considered in isolation, the result is a net relaxation, as expected. However, the new tuples may not actually increase the number of solutions when considered in light of the rest of the constraints.

Recall that a Simple Temporal Constraint has the form $a \leq |X_i - X_j| \leq b$. A Disjunctive Temporal Constraint has scope X_i, X_j, X_k; $d(X_k)$ is discrete and $d(X_i) = d(X_j) = \mathcal{Z}$. For each $x_k \in d(X_k)$ there is a pair of constants a_x, b_x; the constraint has the form $(X_k = x_k) \Rightarrow (a_x \leq |X_i - X_j| \leq b_x)$. What happens if a new assignment $X_k = x_k$ is made in a Disjunctive Temporal Network (DTN) during search after determining that the previous value y_k resulted in a temporal constraint violation? One approach is to have the planner maintains the mapping between discrete values of the variable X_k and a Simple Temporal Network (STN) by transforms $C+, C-$ executed during search. When a violation is detected during search, the violated Simple Temporal Constraint is removed, as described in Figure 2 (bottom), then the new constraint is added.

The transformations are defined in such a way that the CSP resulting after any transformation is well-formed. That is, the resulting CSP has no unusual constructions like a relation with no tuples in it between variables that are supposed to have solutions, or tuples with values for variables not in the scope of the constraints. Thus, after any transform, any form of propagation can be performed, and the results are correct (assuming no further transforms are performed). The difficulty, of course, is that more transforms will take place. Either these transforms are the results of some known operation, e.g. as part of adding an action to a plan during search, or the transforms are possible but not yet known, e.g. some new action could be added afterwards, because the search for a plan is not yet complete. One potential value of propagation is to determine infeasability early, e.g. during construction of the plan graph level, to avoid needless work. A second alternative is to inform heuristics to make smarter decisions during search. Whether it is worthwhile to propagate ’eagerly’ or ’lazily’ remains an open question.

During Modeling

To analyze the consequences of changing actions in a planning problem model, we restrict ourselves to STRIPS models. We use the common STRIPS assumption that action preconditions are all positive, and we do not consider domain axioms. We describe the analysis in the context of Graphplan (Blum and Furst 1995), both for simplicity and because the plan graph can be transformed into a CSP, as described earlier (Do and Kambhampati 2000). Recall in this transformation that 1) variables are the propositions that hold or do not hold at levels of the plangraph; 2) $\perp$ is used to represent false propositions; and 3) the values of variables are the actions that establish propositions. Adding new preconditions

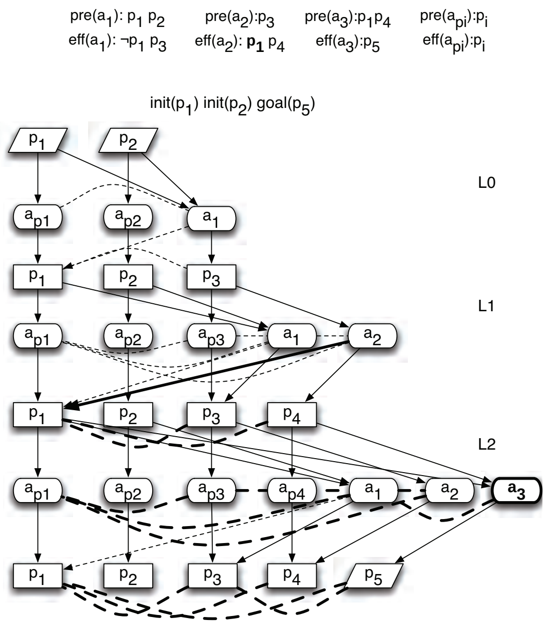


Figure 3: Adding a new positive effect. Arrows show support of actions by propositions, and also how actions lead to positive effects. Dotted lines show mutual exclusions (both action and proposition). Prior to adding positive effect p_1 to a_2, there is no plan with a_3 to reach p_5 from the initial state p_1, p_2. After adding the effect, it is now possible to add action a_3 with effect p_5 at Level 2 of the plan graph. The new positive effect and a_3 are shown in bold above. The mutual exclusions between propositions or actions at Level 2 are removed, and hence are also shown in bold.

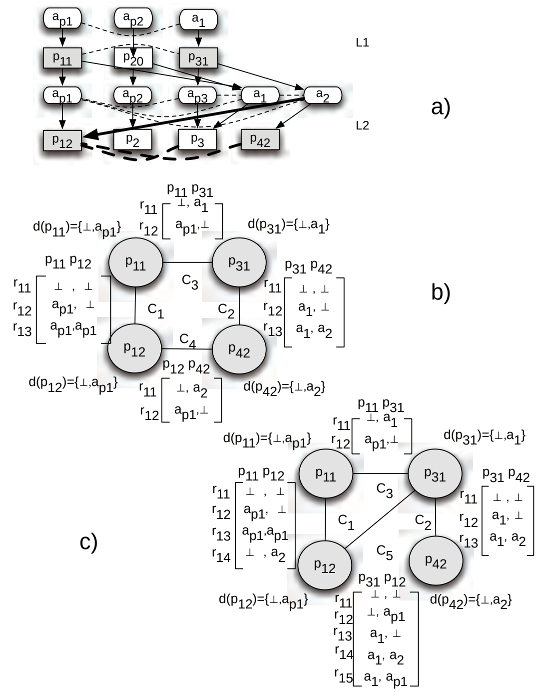


Figure 4: Adding a new positive effect; Level 1 of the plangraph. Prior to adding the positive effect p_1 to a_2, it is not possible to achieve p_1 and p_4 at the same time in level 1 (a). The figure shows the CSP for the variables representing p_1 and p_4 at level 1 of the plangraph, and p_1 and p_3 at level 0 of the plangraph (b). Adding the effect adds a_2 to the domain of p_1 at level 1, leads to elimination of the static mutual exclusion between p_1 and p_4 at level 1, and modifies the constraints whose scope contains p_1(c).

or effects are more than neutral transformations; while they add variables whose values are the existing actions, there are associated constraints that must be added as well. Similarly, adding an action is also more than a neutral transformation, adding values to the domains of the existing (state) variables. To find out more, we must do some additional analysis.

First, consider adding a precondition p to an action a_i. For every action a_j such that $\neg p \in \text{eff}(a_j)$, we add a static mutex constraint between the action variables. This results in a restriction, since it is a constraint in the plan graph and thus in the DCSP. It is accomplished by $C+$ followed by a succession of $r-$ transformations. If p is a new proposition, new variables and constraints are added at every level of the plan graph. This is accomplished by $X+$, followed by a succession of $C+$ and $r-$ transformations. Finally, the mutual exclusions between actions may propagate and lead to other mutual exclusions, leading to more $C+$ and $r-$ transformations. So on balance, any precondition addition cannot be a net relaxation, and thus removing preconditions cannot be a net restriction.

When adding effects, if p is a new proposition, new variables and constraints are added at every level of the plan graph. This is accomplished by $X+$, followed by a succession of $C+$ and $r-$

Adding a negative effect $\neg p \in \text{eff}(a_i)$ introduces a static mutex with every action for which $p \in \text{eff}(a_j)$. This is accomplished by $C+$ followed by a succession of $r-$ transformations. Adding a negative effect also introduces static mutexes with every action such that $p \in \text{pre}(a_k)$, which is also $C+$ followed by a succession of $r-$ transformations. Again, adding a negative effect cannot be a net relaxation.

Adding a positive effect $p \in \text{eff}(a_i)$ introduces a static mutex with every action such that $\neg p \in \text{eff}(a_j)$. Once again, this is a transformation $C+$ followed by a succession of $r-$ transformations, and cannot be a net relaxation. If p could have been added by some action a_j at this level of the plangraph, then adding the effect to a_i relaxes the constraints on p. The value representing a_i will be added to the domain of the variable p. This is accomplished by a $d+$ transformation. Recall the variable could be in the scope of one or more constraints; in our formalism, this would requite a complex series of $r+$ and a $C-$ transformation prior to the $d+$ transformation. The value also is added to one or more tuples of the existing constraints, using the $r+$ transformation. If, however, 1) p was not present in the initial state and 2) no action could have added p to a level of the plangraph where a_i could be executed, then a new variable is added to the plangraph at this level. The new variable has one action, a_i, that adds it. This action may be mutex with other actions, and hence p may be mutex with other propositions at this level. This is accomplished via the $X+$ transformation, possibly followed by $C+$ and $r-$ transformations if there are mutexes present. Finally, if $\neg p \in eff(a_j)$ earlier in the plan graph, a fact mutex could be deleted by adding p as an effect, thereby relaxing some constraints. So adding positive effects is potentially a restriction, a relaxation, or a neutral transformation.

Figures 3 and 4 show the plan graph and resulting changes to the CSPs when adding a positive effect to one action. A summary of the changes to the model in terms of restrictions and relaxations is shown in the table below. Unless otherwise stated, propositions in the change and context are at the first level of the plan graph where action a_i appears. (The inverse model changes are also permitted with the reverse transformations.)

Change	Context	τ	$Notes$
$p \in$ pre(a_i)	new p	$X+$	New p at this level
$p \in$ pre(a_i)	$p \in$ later level	$X+$	New p at this level
$p \in$ pre(a_i)	$\neg p \in$ eff(a_j)	$C+, r-$	Create static mutex
$p \in$ eff(a_i)	new p	$X+$	New p at this level
$\neg p \in$ eff(a_i)	new p	$X+$	New p at this level
$\neg p \in$ eff(a_i)	$p \in$ eff(a_j)	$C+, r-$	Create static mutex
$p \in$ eff(a_i)	$\neg p \in$ eff(a_j)	$C+, r-$	Create static mutex
$p \in$ eff(a_i)	$p \in$ later level	$X+$	New p at this level
$p \in$ eff(a_i)	$p \in$ eff(a_j)	$d+, r+, r-$	New establisher of p
$p \in$ eff(a_i)	$\neg p$ earlier level	$r+, C-$	Delete fact mutex

Extending DCSPs to Optimization Problems

We now show how to extend the notion of DCSPs to optimization problems; more precisely, we address problems in which assignments have costs, and there is a cost bound defining the set of feasible solutions. We choose the Partial Constraint Satisfaction Problems (PCSP) formalism to extend the analysis of dynamic constraint satisfaction PCSPs were originally defined by (Freuder and Wallace 1992). While more modern formalisms (such as constraints over semirings) have more expressive power and more sophisticated theoretical grounding, we will develop the theory of DCSPs using PCSPs for simplicity.

Definition 7 *A* Partial Constraint C_j *is a tuple* S_j, R_j, f_j. *The* scope S_j *of the constraint is a set of variables* $X_{j_1}...X_{j_k}$. *The* relation $R_j \subseteq d(X_{j_1}) \times ... \times d(X_{j_k})$. *Finally,* $f : R_j \rightarrow \mathcal{R}^+$.

Definition 8 *A* Partial Constraint Satisfaction Problem *or PCSP* $\mathcal{P}$ *is a set of variables* $X_1...X_n$ *and domains* $d(X_1)...d(X_n)$ *and a set of partial constraints* $C_1...C_m$ *and a real number* B. *A solution is an assignment* x *such that* $\sum_{r_{j_h} | \exists C_j \; \pi(x,S_j)=r_{j_h}} f(r_{j_h}) < B$.

We will refer to f as the cost of a tuple, since solutions must have a cumulative value below B. This definition allows the cost function of PCSPs with partially defined relations (i.e. f defined on a subset of $d(X_{j_1}) \times ... \times d(X_{j_k})$) to be well-defined; specifically, an assignment x with a projection $\pi(x, S_j)$ not equal to any r_{j_h} contributes nothing to the cost of the assignment.

Changing a PCSP requires several new transformations. First, we must be able to change the cost of any tuple:

Definition 9

Denote the transformation that increases the value of tuple r_{j_h} *in* C_j *by* $f+$.

Denote the transformation that decreases the value of tuple r_{j_h} *in* C_j *by* $f-$.

It also requires revising the definitions of some of the other transformations introduced previously. Strictly speaking, adding and removing constraints and tuples from relations are different for CSPs and PCSPs. In order to avoid confusion we assign new denotations for these transformations:

Definition 10

Denote the transformation that adds C_{m+1} *with* $S_{m+1} = X_{m+1_1}...X_{m+1_k}$ *and relation* $R_{m+1} = \emptyset$ *by* $B+$.

Denote the transformation that removes C_j *with* $S_j = X_{j_1}...X_{j_k}$ *and relation* $R_j = \emptyset$ *by* $B+$.

Denote the transformation that adds a tuple r_{j_h} *to* R_j *with* $f(r_{j_h}) = 0$ *by* $s+$.

Denote the transformation that removes a tuple r_{j_h} *from* R_j *with* $f(r_{j_h}) = 0$ *by* $s-$.

The transformations $X+, X-, d+, d-$ are identical to their DCSP counterparts.

We now are in a position to show how a CSP $\mathcal{P}_i$ can be transformed into a PCSP $\mathcal{P}_j$. Doing so requires transforming C_j, or more specifically, $R_j \in \mathcal{P}_i$, into $R'_j \in \mathcal{P}_j$. For each $r_{j_h} \in d(X_{j_1}) \times ... \times d(X_{j_k})$ such that $r_{j_h} \notin R_j$, add r_{j_h} with $f(r_{j_h}) = 1$ to R'_j. Let $B = 1$. Let x be a satisfying assignment to $\mathcal{P}_i$. Then $\pi(x, S_i) \notin R'_j$, and therefore contributes zero to the sum, $\sum_{r_{j_h} | \exists C_j \; \pi(x,S_j)=r_{j_h}} f(r_{j_h})$ by definition. The sum is therefore below the bound 1. Any other assignment has a sum of at least 1 and therefore does not satisfy the inequality. We can then replace R_j with R'_j as the relation for C_j in the new PCSP. The transformed problem $\mathcal{P}_j$ is the MAX-CSP (Wallace 1996), which is a sub-class of PCSP. We call this new transformation $v+$. Similarly, we can define its inverse transformation $v-$ which can only be performed if, $\forall C_j \, \forall r_{j_h} \in C_j$, $f(r_{j_h}) = 0$ or $f(r_{j_h}) = 1$ and $B = 1$. These transformations are neither a restriction nor a relaxation in the sense that all satisfying assignments continue to satisfy. The transformation is polynomial time, or to put it another way, it is exponential in $s = \max_{j \in C} |S_j|$ (maximum arity of any relation). Note the transformation 'inverts' the set of tuples in the relations, since we add the tuples that will exceed the cost bound and therefore 'violate' the constraints.

Definition 11

Denote the transformation from a CSP into PCSP by $v+$.
Denote the transformation from a PCSP into CSP by $v-$.

There are some equivalences between $f+, f-$ on a PCSP and $r+, r-$ transforms on the CSP from which it is derived. For instance, if $f(r_{j_h}) < 1$ and $B = 1$, and after a $f+$ transform $f'(r_{j_h}) > 1$, this is equivalent to $r-$ on the CSP variant, in which r_{j_h} is removed from R_j. The equivalence is imperfect, in the sense that we should not allow $r+, r-$ transforms on a PCSP, because the sum of costs becomes undefined for some assignments. Similarly, the transforms $f+, f-$ are not allowed on a DCSP since f is undefined for all tuples in all constraints. However, these transformations are equivalent in the sense that they preserve solutions between a CSP and its PCSP.

Theorem 2 *Let $\mathcal{P}_i \xrightarrow{v+} \mathcal{P}_j$ transform a CSP into a PCSP and Let $\mathcal{P}_i \xrightarrow{v-} \mathcal{P}_j$ transform a PCSP into a CSP.*

1. *Let $\mathcal{P}_i \xrightarrow{r+} \mathcal{P}'_i$ increase $L(\mathcal{P}'_i)$. Then there is a transform $\mathcal{P}_j \xrightarrow{f-} \mathcal{P}'_j$ such that $L(\mathcal{P}'_i) = L(\mathcal{P}'_j)$. Let $\mathcal{P}_i \xrightarrow{r-} \mathcal{P}'_i$ decrease $L(\mathcal{P}'_i)$, Then there is a transform $\mathcal{P}_j \xrightarrow{f+} \mathcal{P}'_j$ such that $L(\mathcal{P}'_i) = L(\mathcal{P}'_j)$.*
2. *Let $\mathcal{P}_i \xrightarrow{f-} \mathcal{P}'_i$ increase $L(\mathcal{P}'_i)$. Then there is a transform $\mathcal{P}_j \xrightarrow{r+} \mathcal{P}'_j$ such that $L(\mathcal{P}'_i) = L(\mathcal{P}'_j)$. Let $\mathcal{P}_i \xrightarrow{f+} \mathcal{P}'_i$ decrease $L(\mathcal{P}'_i)$, Then there is a transform $\mathcal{P}_j \xrightarrow{r-} \mathcal{P}'_j$ such that $L(\mathcal{P}'_i) = L(\mathcal{P}'_j)$.*

Finally, consider a transformation to increase or decrease B. This transformation arises naturally in branch-and-bound search, and may also arise during modeling. Is this equivalent to a series of transformations $+f, -f$? Trivially, the answer is no. Making a single change $+f, -f$ can introduce a small change to the solutions that is impossible to mimic with a change to B, as shown in Figure 5.

By contrast, can changes to the solutions due to changes to B be accomplished by a set of changes $+f, -f$? The trivial answer is yes.

Theorem 3 *Let $\mathcal{P}_i \xrightarrow{\tau} \mathcal{P}_j$ transform a PCSP by means of a change in bound from B to $B + b$ or $B - b$. Then there is a series of transforms $\mathcal{P}_i \xrightarrow{f+,f-} \mathcal{P}_k$ such that $L(\mathcal{P}_j) = L(\mathcal{P}_k)$.*

Recall the criteria for x to be a solution to a PCSP is $\sum_{r_{j_h}|\exists C_j\ \pi(x,S_j)=r_{j_h}} f(r_{j_h}) < B$. If we transform B to $B + b$ to admit more solutions, we could alternatively scale each $f(r_{j_h})$ by a factor of $\frac{B}{B+b}$. To show the solutions satisfy the old bound:

$$\sum_{r_{j_h}|\exists C_j\ \pi(x,S_j)=r_{j_h}} \left(\frac{B}{B+b}\right) f(r_{j_h})$$
$$= \left(\frac{B}{B+b}\right) \Big(\sum_{r_{j_h}|\exists C_j\ \pi(x,S_j)=r_{j_h}} f(r_{j_h}) \Big)$$
$$< \left(\frac{B}{B+b}\right)(B+b) = B$$

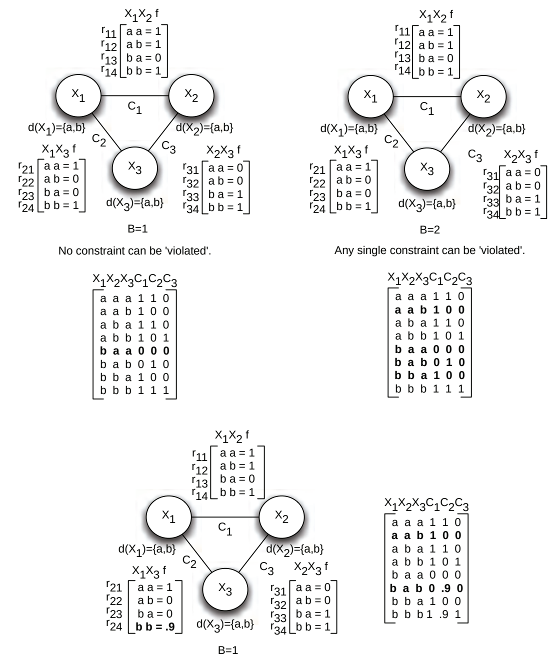


Figure 5: Changes to solutions via $f+, f-$ may not be achievable via a change to B. This PCSP (top left) is a transformed CSP allowing no constraint 'violations' (i.e. the only solution has accumulated cost 0. Changing B from 1 to 2 (top right) allows one 'violation', thereby introducing three new solutions. This is the minimum number of new solutions that can be achieved. By contrast, if $f(r_{2,4})$ of constraint C_2 is changed from 1 to $.9$, (bottom) then one new solution is introduced.

The same argument holds for assignments that are not solutions, and if we reduce B. Notice also that the ranking of optimal solutions is also preserved.

While transforming the bounds can, in fact, be simulated by transformations $f+, f-$, the fact that the required transformations impact *every* tuple in *every* constraint makes for a somewhat indiscriminate transformation. This justifies the inclusion of more concise bounds changes in our list of transformations.

Definition 12
Denote the transformation that raises the bound by $b+$.
Denote this transformation that lowers the bound by $b-$.

In figure 6 we show the transformations from CSPs to PCSPs, and between PCSPs.

Theorem 4 *There are classes of transformations on PCSPs that are restrictions, relaxations, and neither restrictions nor relaxations. Furthermore, for a PCSP $\mathcal{P}_i$, there are classes of transformations that can be either restrictions or neutral, and there are classes of transformations that can be relaxations or neutral.*

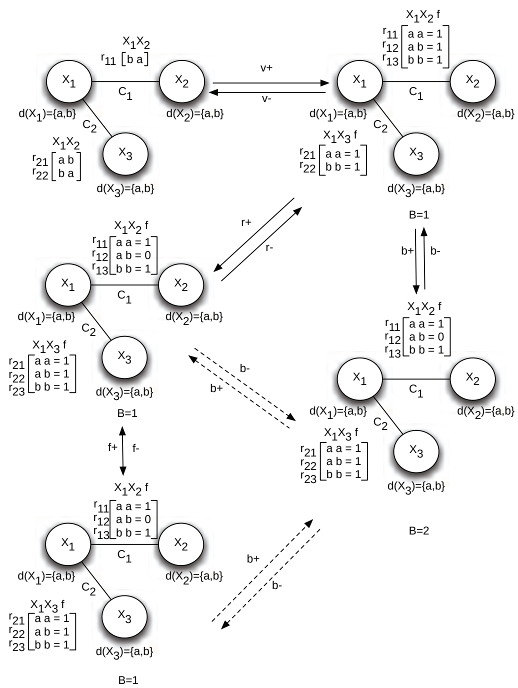


Figure 6: Transformations between CSPs and PCSPs. Set of all valid transformations. (The dotted one shows a valid transform but not the exact evolution of the DCSP in the figure.)

Increasing the cost of a tuple $f+$ or decreasing the bound $b-$ may not decrease the number of solutions or fraction of solutions, but it cannot increase the number or fraction of solutions. Vice versa, decreasing the cost of a tuple $f-$ or increasing the bound $b+$ may not increase the number of solutions or fraction of solutions, but it cannot decrease the number or fraction of solutions.

τ	$\Delta L_p(\mathcal{P})$	$\Delta L(\mathcal{P})$	$\mathcal{A}$
$X+$	0	0	0
$X-$	0	0	0
$d+$	0	>0	>0
$d-$	0	<0	<0
$B+$	0	0	0
$B-$	0	0	0
$s+$	≥ 0	≥ 0	0
$s-$	≤ 0	≤ 0	0
$f+$	≤ 0	≤ 0	0
$f-$	≥ 0	≥ 0	0
$b-$	≤ 0	≤ 0	0
$b+$	≥ 0	≥ 0	0
$v+$	0	0	0
$v-$	0	0	0

Analyzing Optimal Planning and Scheduling

We now briefly discuss optimal planning and scheduling using this new formalism. We will discuss how to transform a model that does not include optimization into a model that does include optimization. We will not spend time on the changing of an optimization problem during search.

Transforming a problem into an optimization problem directly can be done in numerous ways. When few or no solutions are available, a problem is often transformed from a satisfiability problem into an optimization problem. Typical optimization criteria for planning include minimizing plan steps, minimizing makespan for concurrent plans, and maximizing the number or value of goals achieved. The first transformation is neutral, after which either some bounds adjustments or cost adjustments are required. Most of the time, these transformations will introduce new solutions, and therefore be relaxations. Global optimization criteria like minimizing makespan may be difficult to represent explicitly as functions on the value assignments of small numbers of variables.

As mentioned above, an intermediate model change is to allow waiving some or all of the constraints. Allowing the waiving of constraints, however, introduces interesting problems. If all constraints can be waived then there are trivial solutions. Optimization criteria like minimizing the number of waived constraints are needed to prevent the introduction of trivial solutions.

Similarly, there are interesting philosophical differences in how constraints can be waived. Making the variables explicit lets the search algorithm handle it. As in Mapgen and Ensemble, the human can handle it. However, the number of variables goes up as does the representation in the constraints. Letting constraints be removed and added is already required for planners (e.g. IxTeT and EUROPA).

Conclusions and Future Work

In this paper we present a new classification of dynamic constraint satisfaction transformations based on a quantifiable criteria: the change in the fraction of solutions $\Delta L_p(\mathcal{P})$. We have broken down the transformations of DCSPs into a set of elementary transformations. The new criteria can be used to evaluate elementary transformations of a CSP as well as sequences of transformations. We identify a minimum set of transformations from which all other transformations can be constructed. We extend the notion of transformations to include optimization problems. The resulting transformations are shown to consist of restrictions, relaxations, and neutral transformations that neither restrict nor relax a problem. For optimization problems, classes of transformations may contain more than one type of transformation. We identify a complete set of transformations that can transform a problem from a satisficing problem to an optimization problem, or back. We show how these transformations can inform the evolution of planning models, automated planning algorithms, and mixed initiative planning.

The analysis of transformations of planning problems using the new framework contains few real surprises. The most complex transformation, adding or subtracting from the satisfying tuples in a relation, have the most impact on the solutions, but are the hardest to analyze. The most interesting question from the point of view of computational complexity is whether or not deconstructing a transformation into its primitive parts helps 'eager' propagation. Unfortunately there are no easy answers to this question.

From the point of view of modeling, the potential use of this new framework is to analyze proposed model changes. How would such information be provided to a modeler? What can modelers do with this information when it is provided? Consider, for instance, integrating a simple report on the consequences of adding or removing conditions and effects to a tool such as itSimple (Vaquero et al. 2007) for use in a plan or model critique phase of Model-Lite planning. This may be especially useful when satisficing problems are transformed into optimization problems, but feedback for simple situations like adding or removing conditions and effects may provide value. Different planning to CSP encodings, such as those described in (Banerjee 2009) or (van den Briel, Vossen, and Kambhampati 2005), may lead to different results than those described here. Extending the analysis for modeling to include more complex formalisms such as time, resources, domain axioms, and ADL-like constructs (disjunctive preconditions and conditional effects) will extend the power of the approach.

Acknowledgements

We acknowledge the contributions Paul Morris, Minh Do and Tristan Smith made to early drafts of this paper.

References

Banerjee, D. 2009. Integrating planning and scheduling in a cp framework: A transition-based approach. In *Proceedings of the* 19^{th} *International Conference on Automated Planning and Scheduling*.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the* 14^{th} *International Joint Conference on Artificial Intelligence*, 1636–1642.

Bresina, J.; Jonsson, A.; Morris, P.; and Rajan, K. 2005. Activity planning for the Mars Exploration Rovers. In *Proceedings of the* 15^{th} *International Conference on Automated Planning and Scheduling*, 40 – 49.

Dechter, R. & Dechter, A. 1988. Belief maintenance in dynamic constraint networks. *Proceedings of the Seventh National Conference on Artificial Intelligence* 37–42.

Do, M. B., and Kambhampati, S. 2000. Solving planning-graph by compiling it into CSP. In *Proceedings of the* 5^{th} *International Conference on Artificial Intelligence Planning Systems*, 82–91.

Frank, J., and Jónsson, A. 2003. Constraint-Based Attribute and Interval planning. *Journal of Constraints Special Issue on Constraints and Planning*.

Frank, J.; Jónsson, A.; and Morris, P. 2000. On reformulating planning as dynamic constraint satisfaction. *Proceedings of the 4th Symposium on Abstraction, Reformulation and Approximation*.

Freuder, E., and Wallace, R. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21 – 70.

Ghallab, M., and Laurelle, H. 1994. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the 4th International Conference on AI Planning and Scheduling*, 61–677.

Jónsson, A., and Frank, J. 2000. A framework for dynamic constraint reasoning using procedural constraints. *Euopean Conference on Artificial Intelligence*.

Laborie, P. 2003. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143:151–188.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence*, 25–32.

S.Kambhampati. 2007. Model-lite planning for the web-age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the* 13^{th} *National Conference on Artificial Intelligence*.

Soininen, T.; Gelle, E.; and Niemelä, I. 1999. A fixpoint definition of dynamic constraint satisfaction. *Proceedings of the 5th International Conference on the Principles and Practices of Constraint Programming*.

Tsamardinos, I., and Pollack, M. 2003. Efficient so- lution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence* 151(1-2):43–90.

van den Briel, M.; Nigenda, R. S.; Do, M. B.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (oversubscription) planning. In *Proceedings of the* 19^{th} *National Conference on Artificial Intelligence*, 562 – 569.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming for ai planning: A branch and cut framework. In *Proceedings of the* 15^{th} *International Conference on Automated Planning and Scheduling*.

Vaquero, T. S.; Romero, V. M. C.; Tonidanel, F.; and Silva, J. R. 2007. Itsimple 2.0: An integrated tool for designing planning domains. In *Proceedings of the* 17^{th} *International Conference on Automated Planning and Scheduling*.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal POCL planner based on constraint programming. *Artificial Intelligence* 170(3):298 – 335.

Wallace, R. 1996. Enhancement of branch and bound methods for the maximal constraint satisfaction problem. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* 188 – 195.

Decentralized Cooperative Metaheuristic for the Dynamic Berth Allocation Problem

Eduardo Lalla-Ruiz and **Belén Melián-Batista,** and **J. Marcos Moreno-Vega**
Department of Computer Engineering
University of La Laguna, Spain
{elalla, mbmelian, jmmoreno}@ull.es

Abstract

The increasing demand of maritime transport and the great competition among port terminals force their managers to reduce costs by exploiting its resources accurately. In this environment, the Berth Allocation Problem, which aims to allocate and schedule incoming vessels along the quay, plays a relevant role in improving the overall terminal productivity. In order to address this problem, we propose Decentralized Cooperative Metaheuristic (DCM), which is a population-based approach that exploits the concepts of communication and grouping. In DCM, the individuals are organized into groups, where each member shares information with its group partners. This grouping strategy allows to diversify as well as intensify the search in some regions by means of information shared among the individuals of each group. Moreover, the constrained relation for sharing information among individuals through the proposed grouping strategy allows to reduce computational resources in comparison to the 'all to all' communication strategy. The computational experiments for this problem reveal that DCM reports high-quality solutions and identifies promising regions within the search space in short computational times.

Introduction

Maritime container terminals are infrastructures built with the goal of facing the technical requirements arising from the increasing volume of containers in the international sea freight trade. They are aimed at transferring and storing containers within multimodal transportation networks. The main transport modes found at a maritime container terminal are container vessels, trucks, and trains. In this regard, according to the UNCTAD [1], the international maritime container trade has greatly grown over the last decades. One of the most widespread indicators for assessing the competitiveness of a maritime container terminal is the time required to serve the container vessels arriving to the port (Yeo 2010). For this reason, an inefficient utilization of some key resources, like berths, could produce delays of yard-side and land-side operations, giving rise to a poor overall productivity of the container terminal.

The aforementioned issue leads to the definition of the Berth Allocation Problem (BAP). Its main goal is to assign berthing positions along the quay to incoming vessels. In this process, container terminal managers must consider several factors such as the vessels and berth time windows, number of loaded/unloaded containers, water depth, and tide conditions. In this paper, we study the Dynamic Berth Allocation Problem (DBAP) introduced by (Cordeau et al. 2005), which considers berth and vessel time windows as well as heterogeneous vessel service times stemming from the assigned berth.

In order to solve the DBAP, this work proposes Decentralized Cooperative Metaheuristic (DCM). This algorithm is a population-based approach in which a set of individuals is organized into groups that exchange information among them while the search is performed. In this regard, as indicated by (Gutiérrez-Castro et al. 2008), the 'all to all' communication in working systems is not appropriate because it demands too many computational resources. Therefore, the way the information is shared in DCM pursuits a decentralized grouping strategy. Namely, during the search, the individuals only share information with their group partners. Each group has its own leader and rules regarding how to exchange information.

The goals of this work are, on the one hand, to assess the behaviour of DCM as well as provide high-quality solutions by means of short computational times for the berth allocation at maritime container terminals. On the other hand, we seek to evaluate the effectiveness of DCM by comparing its computational results with those reported by the mathematical model proposed by (Christensen and Holst 2008) and the results obtained by the best algorithms from the related literature for the DBAP. In this regard, as discussed in the relevant section, the computational results provided by DCM indicate that it requires less computational time than the best solution approach recently proposed in the literature for the DBAP.

The remainder of this paper is organized as follows. A short literature review of the BAP is presented in the following section. Then, the mathematical formulation of the DBAP used in this work is described. In the next section, the algorithm proposed for addressing the BAP is described.

[1] United Nations Conference on Trade And Development, http://unctad.org

Later, the computational experience carried out and a comparative summary are presented. Finally, some conclusions and several lines for further research are drawn in the last section.

Literature Review

The Berth Allocation Problem (BAP) has been extensively studied in the literature. In this regard, due to the large variety of maritime terminal layouts, research has produced multitude of variants for this problem. Depending on how the quay is modelled, the BAP can be referred to as discrete (the quay is divided into segments called berths) or continuous (the quay is not divided, thus the vessels can berth at any position in the quay). Moreover, in some related works (Cordeau et al. 2005), (Umang, Bierlaire, and Vacca 2013)) there is also a hybrid consideration of the quay (the quay is divided into a set of berths and a vessel can occupy more than one berth at a time or share its assigned berth with other container vessels). Depending on the arrival time, the BAP can be classified into static (the vessels are already in port when the berths become available) or dynamic (the vessels arrive during the planning horizon). For detailed descriptions, the reader is referred to (Bierwirth and Meisel 2010) and (Christiansen et al. 2007).

One of the most relevant approaches is the Dynamic Berth Allocation Problem (DBAP). It was first formulated by (Imai, Nishimura, and Papadimitriou 2001) as an extension of the model proposed in (Imai, Nagaiwa, and Chan 1997) for the Static Berth Allocation Problem. Alternative formulations for the dynamic problem have been proposed and studied by (Monaco and Sammarra 2007), (Cordeau et al. 2005) and (Christensen and Holst 2008). These models are described and compared in (Buhrkal et al. 2011). The main conclusion extracted from the latter work is that the model presented by Christensen and Holst is superior to the other models when considering the temporal behaviour. In this regard, it is able to reach the optimal solutions within short computational time for the set of instances used by all the previous authors.

Recently, (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012) presented an efficient Tabu Search metaheuristic with Path-Relinking for solving the DBAP. They also proposed a benchmark suite of instances for which the model by (Christensen and Holst 2008) does not provide feasible solutions within a time limit. (de Oliveira, Mauri, and Lorena 2012) presents a Clustering Search (CS-SA) with Simulated Annealing for solving the DBAP. This algorithm provides the optimal solutions for all the largest instances proposed by (Cordeau et al. 2005). In this regard, (Ting, Wu, and Chou 2013) propose a Particle Swarm Optimization algorithm for addressing the DBAP, which reports optimal solutions within shorter computational times than CS-SA.

Dynamic Berth Allocation Problem

In this work, we address the Dynamic Berth Allocation Problem (DBAP) proposed by (Cordeau et al. 2005), which is modeled as a Multi-Depot Vehicle Routing Problem with Time-Windows (MDVRPTW). The vessels are seen as customers and the berths as depots at which one vehicle is located. The goal of the DBAP is to determine the berthing position and berthing time of $|N|$ incoming vessels along the quay, which is divided into $|M|$ berths. In order to make this paper self-contained, the description of the model proposed by (Cordeau et al. 2005) is included. The following parameters are defined in the problem:

- N, set of vessels
- M, set of berths
- t_i^k, handling time of vessel $i \in N$ at berth $k \in M$
- a_i, b_i, arrival, departure time of vessel $i \in N$
- l^k, e^k, start, end of the availability of the berth $k \in M$
- v_i, the service priority of each vessel $i \in N$

Let us define a graph, $G^k = (V^k, A^k)\ \forall\ k \in M$, where $V^k = N \cup \{o(k), d(k)\}$ contains a vertex for each vessel as well as the vertices $o(k)$ and $d(k)$, which are the origin and destination nodes for any route in the graph. The set of arcs is defined as $A^k \subseteq V^k \times V^k$, where each one represent the handling time of the vessel. The decision variables are as follows:

- $x_{ij}^k \in \{0,1\}$, $\forall\ k \in M$, $\forall\ (i,j) \in A^k$, set to 1 if vessel j is scheduled after vessel i at berth k, and 0 otherwise.
- T_i^k, $\forall\ k \in M, \forall\ i \in N$, the berthing time of vessel i at berth k, i.e., the time when the vessel berth.
- $T_{o(k)}^k$, $\forall\ k \in M$, starting operation time of berth k, i.e., the time when the first vessel berths at the berth.
- $T_{d(k)}^k$, $\forall\ k \in M$, ending operation time of berth k, i.e., the time when the last vessel departs at the berth.

The assumptions considered in the mathematical model are the following:

(a) Each berth $k \in M$ can only handle one vessel at a time.

(b) The service time of each vessel $i \in N$ is determined by the assigned berth $k \in M$.

(c) Each vessel $i \in N$ can be served only after its arrival time a_i.

(d) Each vessel $i \in N$ has to be served until its departure time b_i.

(e) Each vessel $i \in N$ can only be berthed at berth $k \in M$ after k becomes available at time step l^k.

(f) Each vessel $i \in N$ can only be berthed at berth $k \in M$ until k becomes unavailable at time step e^k.

The time windows of the vessels and berths are defined by (c)-(f). The objective function (1) aims to minimize the total (weighted) service time for all the vessels, defined as the time elapsed between their arrival to the port and the completion of their handling. When i is not assigned to berth k, the corresponding term in the objective function is zero because $\sum_{j \in N \cup d(k)} x_{ij}^k = 0$ and $T_i^k = a_i$. A detailed mathematical formalization of the model can be consulted in (Cordeau et al. 2005).

$$minimize \sum_{i \in N} \sum_{k \in M} v_i \left[T_i^k - a_i + t_i^k \sum_{j \in N \cup d(k)} x_{ij}^k \right] \quad (1)$$

In order to improve the understanding of the DBAP, we provide in Figure 1 an example of a berth scheduling. In the figure, a schedule and an assignment plan are shown for 6 vessels within 3 berths. The rectangles indicate the vessels and inside each rectangle we display its corresponding service priority (v_i), used for establishing vessels priorities. The time windows of the vessels are represented by the lines at the bottom of the figure. In this case, for example, vessel 1 arrives at time step 4 and it should be served before time step 14. Moreover, the time window of each berth is limited by the not hatched areas. Table 1 reports the different handling times for each vessel depending on the assigned berth. For example, if vessel 1 is assigned to berth 1, its handling time would be equal to 6, which is shorter than the handling time of 8 that it would have at berth 2. As can be seen in the example, vessels 5 and 6 would have to wait for berthing in their respective assigned berths. In this regard, since their service priorities value are 1, their wait for berthing will have less impact in the objective function value than delaying other vessels, like vessels 3 and 4, for which the service priorities are 6 and 4, respectively. That is, if their berthing time are delayed, the waiting time step of each vessel is multiplied for 6 and 4, respectively. The objective function value of this solution example is 101.

Table 1: Vessels handling times depending on the allocated berth

	Berth 1	Berth 2	Berth 3
Vessel 1	6	8	5
Vessel 2	2	3	4
Vessel 3	5	5	4
Vessel 4	4	6	5
Vessel 5	5	8	7
Vessel 6	4	4	5

Decentralized Cooperative Metaheuristic

In this work, we propose Decentralized Cooperative Metaheuristic (DCM), which is a population-based approach that exploits the concepts of communication and grouping. It is inspired by the work by (Duman, Uysal, and Alkaya 2012). In that work, the authors propose a nature-inspired metaheuristic based on the V-formation flight of migrating birds. The method is called Migrating Birds Optimization (MBO) and consists of a set of individuals called *birds* that are centred around a leader bird. The way they perform this communication is by considering a V-formation structure. Figure 2(b) shows an illustrative scheme of the V-formation, in which each circle corresponds to a bird. The leader is represented by the circle at the top, whereas the remaining circles represent the rest of the flock. The arrows in the figure represent how the information is shared among the birds. In MBO the shared information corresponds to the best discarded neighbour solutions. The initial positions of the individuals along the V-formation depend on the generation order. That is, the first individual generated will be the bird 1 and, therefore, the leader of the flock, the second and third will be its followers, and so forth. Figure 2(a) shows an example of an initial population: id represents the identifier of each bird derived from the generation order and obj indicates the objective function value associated to each individual. After generating the initial population, the individuals are organized into a V-formation, as shown in Figure 2(b). It should be noted that the individuals are positioned regardless the objective function value associated to them.

In MBO, during the search process, the leader bird randomly generates a number n_{on} of neighbour solutions. The remaining birds of the flock generate n_{on} neighbour solutions minus the number of solutions to be shared δ. For each bird, if the best generated neighbour leads to an improvement, the current individual is replaced by that neighbour solution. The δ neighbour solutions that are not used to replace the existing bird are shared with its followers. For instance, in the example shown in Figure 2(b), the bird 2 will share its best discarded neighbour solutions with bird 4. This search process is repeated until a number of prefixed iterations, $iter_l$ is reached. Once, the leader bird becomes the last solution, one of its direct followers becomes the new leader. Considering the example shown in Figure 2(b), the bird 1 would occupy the place of bird 6 and all the birds of that wing of the V-formation will move forward one position. That is, bird 2 would become the leader and bird 4 will be its next follower, and so on. Then, the search process is re-started until $iter_l$ iterations are reached. Once that, the next bird to take the leader role will be bird 3 and bird 2 will occupy the place of bird 7. This process is executed until a number of neighbour solutions, max_N, has been generated through the search process. For a more detailed description of the MBO algorithm, the reader is referred to (Duman, Uysal, and Alkaya 2012).

The MBO algorithm has been successfully applied to the Quadratic Assignment Problem (Duman, Uysal, and Alkaya 2012). Although it provides good quality solutions by means of short computational time, one of its main drawbacks is that the search can easily converge to a local optimum. This is due to the fact that the V-formation is centred on a leader and the way the communication among the individuals is performed. In this regard, the convergence to a local optimum depends on the number of shared solutions, δ, and the current leader. For example, when a leader reaches a local optimum, its δ discarded solutions are shared with the rest of the flock. Thus, these birds could likely become neighbour solutions of that local optimum. In this regard, once the leader reaches the $iter_l$ number of iterations being the head of the formation, one of its direct followers would take over the lead role and it could likely be a neighbour solution of its past leader, which was a local optimum solution.

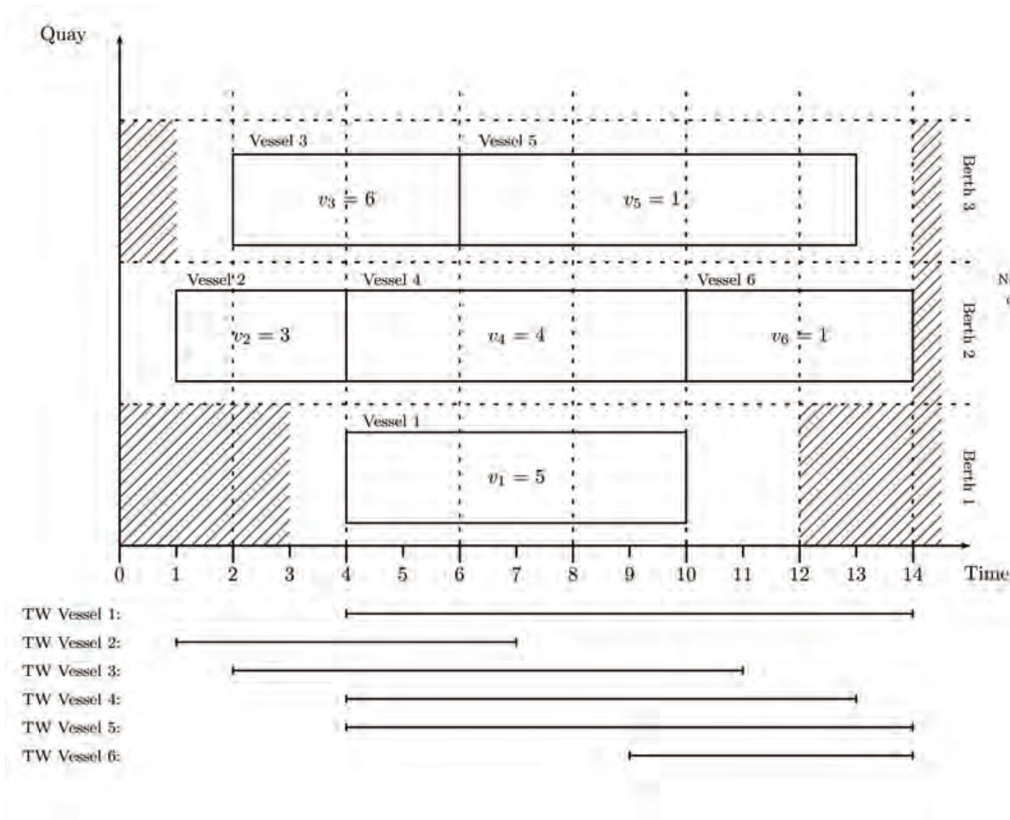
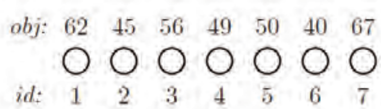
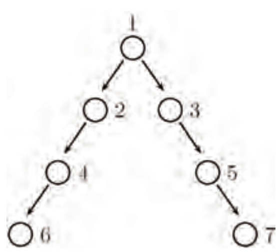


Figure 1: Example of solution for the DBAP with 6 vessels and 3 berths



(a) Initial population



(b) Population V-formation

Figure 2: MBO scheme example

Other minor issues concerning the MBO algorithm are the initial position of each bird in the formation and the way in which the birds are relieved. The initial position of the individuals, as explained above, are determined by the generation order. This could produce that, depending on the values of $iter_l$ and the δ number of shared neighbours, some birds (the lower-medium ones in the formation) would lose their identity since they can become a neighbour solution of a bird in front of them and thus, some promising regions may be ruled out. It makes sense then to include a decentralized strategy that would allow to diversify the search and reduce the likelihood to converge to local optima. Moreover, as indicated by (Klotsman and Tal 2012), the initial formation of the birds when starting a flight is not the V-formation. Thus, a V-formation from the beginning of the flight as described in the MBO would not be the accurate behaviour of the migratory birds. Moreover, as described by (Bajec, Zimic, and Mraz 2005), the migratory birds can also be organized in groups or present a different flight formation besides the V-formation.

To address the above mentioned details, the DCM is developed. In this approach, a population of individuals, S, is organized into groups. This distribution into groups is called *formation*. An example of a formation composed of three groups and two not grouped solutions is shown in Figure 3(b). The organization of the population into groups allows to recognize:

- A set of leader individuals ($S_{leaders}$): This set is made up by the best solutions of each group.
- A set of independent individuals (S_{ind}): This set includes all the individuals that are not grouped. Thus, they do not exchange information with any other individuals.
- A set of follower individuals (S_{fol}): This set contains the individuals that are neither leaders nor independents.

Moreover, in DCM, as it is done in MBO, each individual of a group performs a search procedure and exchanges information with other while the search is being developed. This characteristic gives rise to a cooperative scheme. The way the information is shared is almost constrained to the group members, in the sense that only at most two members of a group can share information with other groups. Nevertheless, some individuals may belong to more than one group and, therefore, share information with more than one group. DCM consists basically of three main components that are summarized in the following items:

(i) *Grouping*. The individuals are organized into groups. The criterion for establishing the groups is a decision of the designer since it can be performed according to different criteria such as objective function value, solution structure, frequency, etc. In the context of the DBAP, we use the objective function value of each individual calculated according to Eq. 1. Moreover, the adjacency of each individual is determined by their creation order. That is, considering the individuals of Figure 3(a), the first individual created, 1, will have only one adjacent solution 2. The second individual created, 2, will have two adjacent solutions, 1 and 3, and so on. The adjacency among individuals is never altered during the execution of the algorithm.

   Once the individuals are created and their objective function values are calculated, a comparison among the individuals and their adjacent ones with respect to the objective function value is performed. When an individual presents a worse objective function value than its adjacent individual, then it will directly form part of its adjacent group. However, if both of them have the same objective function value, there will no exist any communication. Figure 3 shows an example where 15 individuals are organized into groups according to their objective function value. In this case, individual 1 has a worse objective function value (50) than its adjacent 2 (46) so it will form part of the individual 2 group.

(ii) *Sharing*. The individuals of each group share information with their adjacent group partners. There is no direct exchange between individuals in different groups, but indirect exchanges may arise due to individuals appearing in more than one group. This is shown in Figure 3(b), where the individual 8 belongs to two groups.

   The information shared in the DCM approach applied to the DBAP consists of the best discarded solutions. The way they exchange information depends on their objective function value. That is, if an individual presents a better objective function value than its adjacent solution, it will directly change information with it by providing its best discarded solutions. However, if both of them have the same objective function value, there will not be any information exchanged. Finally, if the individual has a worse objective function value, it will receive information from its adjacent. In the example shown in Figure 3, the individual 1 will receive information by means of the best discarded neighbours solutions from individual 2. Individuals 5, 6 and 7 will not exchange information among them.

(iii) *Formation*. The formation consists of the division of the population into groups and the way the information is exchanged among them. In DCM, the formation is not always the same, it can be re-determined if a given formation stopping condition is met. That is, when a certain criterion is met, the distribution of the population is re-designed. In that case, all the individuals are compared again and a new division of the population into groups is performed. For the proposed solution approach, the formation stopping condition is met when the best solution found is improved or all group leader individuals could not improve their objective function value in the current iteration.

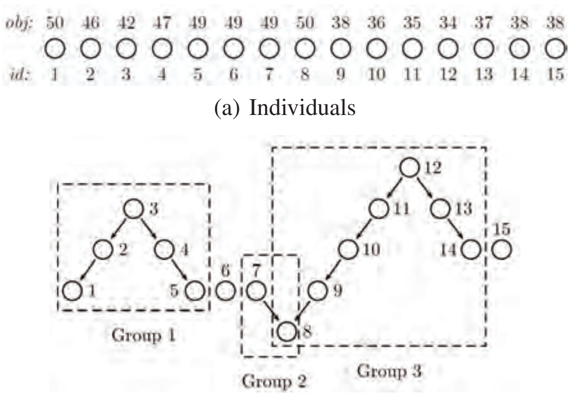
The pseudocode of DCM is depicted in Algorithm 1. The initial population composed of n_s individuals is randomly generated (line 1). The best solution is initialized to the best individual (line 2). The formation is determined by considering and comparing the objective function value of each individual and its adjacent ones (line 4). Once the individuals are organized into groups, the search process for that formation is performed (lines $5-15$) until the formation stopping condition is met. In this case, the formation stopping criteria used for the DBAP is set until the best solution known, s_{best}, is improved or any leader solution, $s \in S_{leaders}$, is able to improve. In the search process, n_{on} random neighbour solutions are generated for each group leader and independent individual (line 7). If the best neighbour random solution leads to an improvement, the current solution is replaced by that one (line 8). Then, each individual $s \in S_{fol}$ generates $n_{on}-\delta$ neighbours and adds the δ best discarded neighbours received from its adjacent individual (lines $11-12$). In the special case that an individual belongs to two groups it will receive $2 \cdot \delta$ solutions. If the best solution (generated by the individual or received from an individual in front of it) leads to an improvement, the solution is replaced by that one (line 13). The DCM search process is carried out while a stopping criterion is not met (line 3). For the DBAP, the search is performed until a maximum number of neighbours equal to $|N|^3$ has been generated by the individuals, where $|N|$ is the number of vessels, or a number n_{imp} of consecutive iterations without improvement of any individual has been performed.

Figure 3 shows an example of a formation for DCM. That is, the individuals are organized into three different groups and two independent solutions that do not cooperate with other individuals. These ‘freelance’ individuals generate the same number of neighbours as a group leader. They can be seen as part of a diversification strategy since they do not communicate with other individuals. Thus, they are not influenced to move to other regions of the search space by other individuals because they do not receive any neighbour solution. Furthermore, it should be highlighted that some groups can influence other ones if they have common individuals. This is the case of the groups 2 and 3.

The relationship among individuals is based on sharing their best discarded neighbours with their adjacent individuals. Therefore, the individuals of a group are able to inten-

Algorithm 1: Decentralized Cooperative Metaheuristic

Generate a set S of n_s individuals at random
$s_{best} \leftarrow$ best solution $\in S$
while *(stopping criterion is not met)* **do**
    Determine the groups according to Eq. 1
    while *(formation stopping condition is not met)* **do**
        for *($\forall s \in S_{leader} \cup S_{ind}$)* **do**
            Generate n_{on} neighbour solutions for each s
            Move each individual to its best solution if leads to an improvement
        end
        for *($\forall s \in S_{fol}$)* **do**
            Generate $n_{on} - \delta$ neighbour solutions for each s
            Each s obtains δ unused best neighbours from the solution in the front
            Move each individual to its best solution if leads to an improvement
        end
    end
end
return s_{best}



(b) Formation

Figure 3: Example of the DCM scheme

sify the search on those promising search space regions by increasing the number of generated neighbours. Table 2 and Figure 4 show an example of an iteration within the DCM search process. In this example, we have a population of individuals $S = \{1, ..., 6\}$ organized into 2 groups according to their objective value. The leaders of the groups are individuals 2 and 5. For this example, we consider that each individual manages $n_{on} = 3$ random neighbour solutions and each follower receives $\delta = 1$ solution. Table 2 reports the objective value of each individual, obj. Under the heading *Generated solutions*, shows the neighbour solutions generated, ϕ_i, by each individual $i \in S$ and $j \in \{a, b, c\}$. Column *Received solutions* shows the solutions received from individual j according to the formation. Figure 4 illustrates the group division and the way the individuals exchange information. In this example, the leader individual 2 generates

Table 2: Example of the search process within DCM. Underlined indicates that the individual will move to that solution at the next iteration

Index	*obj*	*Generated solutions* $(obj.)$			*Received solutions* $(obj.)$
1	50	ϕ_{1a} (51)	ϕ_{1b}(49)	-	$\underline{\phi_{2a}(44)}$
2	$\underline{42}$	ϕ_{2a} (44)	ϕ_{2b}(46)	ϕ_{2c}(47)	-
3	55	ϕ_{3a} (53)	ϕ_{3b}(50)	-	ϕ_{4a} (50) , $\underline{\phi_{2b}(46)}$
4	52	ϕ_{4a} (50)	ϕ_{4b} (54)	-	$\underline{\phi_{5a}\ (49)}$
5	48	ϕ_{5a} (49)	$\underline{\phi_{5b}\ (46)}$	ϕ_{5c}(51)	-
6	54	ϕ_{6a} (53)	$\underline{\phi_{6b}\ (52)}$	-	$\underline{\phi_{5b}\ (46)}$

three random neighbour solutions ϕ_{2a}, ϕ_{2b}, and ϕ_{2c}. The objective value of those solutions will not lead to an improvement of its objective value. Thus, it will no move to other solution but it shares its best discarded solutions with its followers. In this case, according to Figure 4, individual 1 receives from individual 2 the solution, ϕ_{2a}. This solution is the one that allows the greatest improvement of objective function value of individual 1. Therefore, the individual 1 moves to that solution. Moreover, as can be seen in Table 2, individual 3 receives two neighbour solutions since it belongs to two groups. In this case, individual 3 will move to solution ϕ_{2b} because it allows the greatest improvement. Hence, at the next iteration individuals 1 and 3 will move to the same region as their leader 2. This allows to intensify the search in that region.

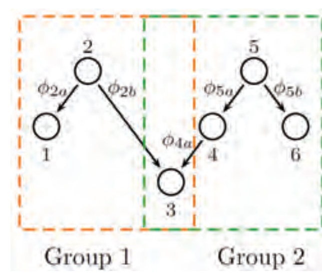


Figure 4: DCM information exchange

DCM for the DBAP

In the context of the DBAP, the DCM implementation for this problem considers a solution s as a sequence composed by features, where a *feature*, is defined as indicated below:

$$features(s) = \{(i, j) : \text{vessel } j \text{ is assigned to berth } i\} .$$

Figure 5 shows an example of the solution structure for the planning example shown in Figure 1. Each berth is delimited by a 0. Thus, there will be M sub-sequences. The service order of each vessel is determined by its position in the subsequence. As can be seen in Figure 5, only vessel 1 is allocated at berth 1. At berth 2, the vessel 2 is the first vessel to be allocated. Once it departs from the berth, the next vessel to be allocated is vessel 4, and so on.

The neighbourhoods used in this approach are the following:

Figure 5: Solution structure for the BAP

1	0	2	4	6	0	3	5

(a) Reinsertion-move, $N_1(s, \lambda)$: λ vessels are removed from a berth i and reinserted into another berth i' ($\forall i, i' \in M, i \neq i'$).

(b) Interchange-move, $N_2(s)$: It consists of exchanging a vessel j assigned to berth i with a vessel j' assigned to berth i' ($\forall j, j' \in N, j \neq j', \forall i, i' \in M, i \neq i'$).

The leaders and independent individuals produce n_{on} random neighbour solutions using the reinsertion movement, whereas the other individuals use the interchange-move. The DCM approach for the DBAP is performed until a maximum number of neighbours equals to $|N|^3$ has been generated, where $|N|$ is the number of vessels, or a number n_{imp} of consecutive iterations without improvement of any individual has been performed.

Computational Results

This section is devoted to present the computational experiments carried out in order to assess the performance of the Decentralized Cooperative Metaheuristic. All the reported computational experiments were conducted on a computer equipped with an Intel 3.16 GHz and 4 GB of RAM. By taking into account the experiments carried out in this work, we identified the following parameter values for DCM: $n_{on} = 20$, $\delta = 3$, number of individuals $n_s = 30$, and stopping criteria of $max_N = |N|^3$ number of generated neighbour solutions by the individuals or $n_{imp} = 20$ consecutive iterations without improvement of reached by some individual of DCM.

The problem instances used for evaluating the proposed algorithm are provided by (Cordeau et al. 2005) and (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012). The instances from (Cordeau et al. 2005) were generated by taking into account a statistical analysis of the traffic and berth allocation data at the maritime container terminal of Gioia Tauro (Italy). The problem instances from (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012) were generated according to the work by (Cordeau et al. 2005) and address other realistic scenarios arising at container terminals. Moreover, with the aim of comparing DCM with MBO, an approach of MBO for the DBAP is implemented in the same way as proposed by (Duman, Uysal, and Alkaya 2012). Therefore, the comparison among the following algorithmic methods for the DBAP is provided along the remainder of this section.

– Generalised Set-Partitioning Problem mathematical model (GSPP) (Christensen and Holst 2008)

– Clustering Search with Simulated Annealing for generating initial solutions (CS-SA) (de Oliveira, Mauri, and Lorena 2012)

– Particle Swarm Optimization (PSO) (Ting, Wu, and Chou 2013)

– Migrating Birds Optimization approach for the DBAP developed in this work (MBO) (Duman, Uysal, and Alkaya 2012)

– Decentralized Cooperative Metaheuristic (DCM)

Table 3 shows the computational results obtained by applying these solution approaches. The mathematical formulation GSPP implemented in CPLEX[2] by (Buhrkal et al. 2011) provides the optimal solution in 17.92 seconds in the worst case. However, as highlighted by (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012), GSPP can require large amounts of memory and computational time, depending on the complexity of the instances. In this regard, a Clustering Search with Simulated Annealing (CS-SA) that is able to provide the optimal solutions in all the cases and outperforms the GSPP time behaviour is presented in (de Oliveira, Mauri, and Lorena 2012). The results shown in the table related to this algorithm correspond to the best objective function values obtained and the average computational time required for 5 tests. Recently, (Ting, Wu, and Chou 2013) have proposed a Particle Swarm Optimization (PSO), which finds the optimal solutions with less computational effort. The results shown in the table correspond to the best objective function values provided by PSO and the computational time is the average time required for the 30 executions.

The comparison of DCM with the two different population-based solution approaches, CS-SA and PSO, reported in Table 3, shows that DCM presents a similar behaviour regarding the quality of the solutions within less computational time. In this regard, the comparison with PSO, which is a metaheuristic that follows a decentralized strategy inspired by the social behaviour of individuals inside swarms, would highlight the benefits of applying a cooperative structure within a decentralized scheme. Moreover, the comparison with CS-SA can give us an idea of the capability of DCM for identifying high promising regions since CS-SA locates promising regions through framing them by clusters. This could likely indicate that the way the regions are pointed out by DCM could be appropriate. However, this detail cannot be clearly claimed since the CS is used jointly with a Simulated Annealing and a Local Search process. Therefore, a more in-depth analysis of the individual contribution of those components would be required.

In this work, a MBO approach for the DBAP is also implemented. The rationale behind including this algorithm is to compare the behaviour of DCM with MBO. Moreover, both algorithms are studied with and without a Local Search applied to each leader solution once their search process is over. The aim of applying a local search after the algorithms have been executed seeks to analyse if they are able to point out high promising regions in the search space. As can be seen in Table 3, the bold numbers indicate those solutions where DCM without local search is able to point out 21 regions, where the optimal solution obtained after applying the local search to each leader individual. In this regard, MBO is able to highlight 17 regions, where the optimal is included.

[2]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Table 3: Computational results for the instances provided by (Cordeau et al. 2005). Bold numbers indicate those solutions that after applying a local search it is possible to reach the optimal solution

	GSPP		CS-SA			PSO			MBO						DCM					
									w/LS			w/o LS			w/LS			w/o LS		
	Opt.	t (s.)	Best	Gap (%)	t (s.)	Best	Gap (%)	t (s.)	Best	Gap(%)	t (s.)	Best	Gap(%)	t (s.)	Best	Gap (%)	t (s.)	Best	Gap (%)	t (s.)
i01	1409	17.92	1409	0.00	12.47	1409	0.00	11.11	1411	0.14	3.42	1441	2.27	2.72	1409	0.00	5.95	**1420**	0.78	3.25
i02	1261	15.77	1261	0.00	12.59	1261	0.00	7.89	1261	0.00	3.52	**1265**	0.32	2.43	1261	0.00	4.15	1261	0.00	3.29
i03	1129	13.54	1129	0.00	12.64	1129	0.00	7.48	1129	0.00	3.63	**1144**	1.33	2.51	1129	0.00	4.18	**1130**	0.09	3.20
i04	1302	14.48	1302	0.00	12.59	1302	0.00	6.03	1302	0.00	3.81	**1304**	0.15	2.43	1302	0.00	4.25	1302	0.00	3.07
i05	1207	17.21	1207	0.00	12.68	1207	0.00	5.84	1207	0.00	3.13	**1212**	0.41	2.12	1207	0.00	3.21	1207	0.00	2.86
i06	1261	13.85	1261	0.00	12.56	1261	0.00	7.67	1261	0.00	3.46	**1272**	0.87	2.42	1261	0.00	4.04	**1262**	0.08	2.90
i07	1279	14.60	1279	0.00	12.63	1279	0.00	7.5	1279	0.00	3.05	**1291**	0.94	2.09	1279	0.00	3.36	**1280**	0.08	2.97
i08	1299	14.21	1299	0.00	12.57	1299	0.00	9.94	1299	0.00	3.30	**1313**	1.08	2.21	1299	0.00	4.96	**1304**	0.38	3.10
i09	1444	16.51	1444	0.00	12.58	1444	0.00	4.25	1444	0.00	3.48	**1457**	0.90	2.29	1444	0.00	5.25	**1446**	0.14	3.31
i10	1213	14.16	1213	0.00	12.61	1213	0.00	5.2	1213	0.00	3.40	**1219**	0.49	2.44	1213	0.00	3.46	1213	0.00	3.20
i11	1368	14.13	1368	0.00	12.58	1368	0.00	10.52	1370	0.15	3.41	1380	0.88	2.16	1368	0.00	5.21	**1374**	0.44	3.39
i12	1325	15.60	1325	0.00	12.61	1325	0.00	12.92	1330	0.38	3.54	1344	1.43	2.54	1325	0.00	4.62	**1330**	0.38	3.38
i13	1360	13.87	1360	0.00	12.58	1360	0.00	11.97	1360	0.00	3.59	**1372**	0.88	2.45	1360	0.00	3.76	**1362**	0.15	3.47
i14	1233	15.60	1233	0.00	12.56	1233	0.00	7.11	1233	0.00	3.27	**1242**	0.73	2.28	1233	0.00	4.14	1233	0.00	3.04
i15	1295	13.52	1295	0.00	12.61	1295	0.00	8.3	1295	0.00	3.43	**1306**	0.85	2.28	1295	0.00	4.31	1295	0.00	3.40
i16	1364	13.68	1364	0.00	12.67	1364	0.00	8.48	1367	0.22	4.14	1394	2.20	2.51	1364	0.00	4.89	**1368**	0.29	3.94
i17	1283	13.37	1283	0.00	13.80	1283	0.00	5.66	1283	0.00	2.63	1283	0.00	1.94	1283	0.00	3.09	1283	0.00	2.68
i18	1345	13.51	1345	0.00	14.46	1345	0.00	8.02	1345	0.00	3.38	**1350**	0.37	2.18	1345	0.00	4.14	**1347**	0.15	3.36
i19	1367	14.59	1367	0.00	13.73	1367	0.00	11.42	1372	0.37	3.81	1390	1.68	2.57	1367	0.00	5.93	**1374**	0.51	4.03
i20	1328	16.64	1328	0.00	12.82	1328	0.00	12.28	1329	0.08	3.55	1352	1.81	2.39	1328	0.00	5.60	**1334**	0.45	3.97
i21	1341	13.37	1341	0.00	12.68	1341	0.00	7.11	1343	0.15	3.93	1359	1.34	2.65	1341	0.00	5.54	**1346**	0.37	3.51
i22	1326	15.24	1326	0.00	12.62	1326	0.00	7.94	1326	0.00	3.38	**1348**	1.66	2.25	1326	0.00	4.97	**1333**	0.53	3.13
i23	1266	13.65	1266	0.00	12.62	1266	0.00	7.25	1266	0.00	3.47	**1283**	1.34	2.28	1266	0.00	4.01	1266	0.00	3.75
i24	1260	15.58	1260	0.00	12.64	1260	0.00	5.67	1260	0.00	3.51	**1264**	0.32	2.37	1260	0.00	4.90	**1261**	0.08	3.61
i25	1376	15.80	1376	0.00	12.62	1376	0.00	7.13	1377	0.07	3.30	1392	1.16	2.00	1376	0.00	5.54	**1381**	0.36	3.39
i26	1318	15.38	1318	0.00	12.62	1318	0.00	7.44	1319	0.08	3.45	1333	1.14	2.20	1318	0.00	4.92	**1325**	0.53	3.52
i27	1261	15.52	1261	0.00	12.64	1261	0.00	6.16	1261	0.00	3.16	**1273**	0.95	2.27	1261	0.00	4.00	1261	0.00	3.15
i28	1359	16.22	1359	0.00	12.71	1359	0.00	11.52	1361	0.15	3.42	1372	0.96	2.50	1359	0.00	5.56	**1363**	0.29	3.40
i29	1280	15.30	1280	0.00	12.62	1280	0.00	8.11	1281	0.08	3.77	1289	0.70	2.60	1280	0.00	5.82	**1282**	0.16	3.25
i30	1344	16.52	1344	0.00	12.58	1344	0.00	7.13	1349	0.37	3.78	1380	2.68	2.48	1344	0.00	5.76	**1350**	0.45	3.52
		14.98	1306.77		12.76	1306.77		8.17	1307.77		3.47	1320.80		2.35	1306.77		4.65	1309.77		3.33

Concerning the required computational effort, DCM is able to improve the computational time if compared with the best solution approaches presented in the literature. In this regard, it can be pointed out that MBO requires less computational time. However, since both algorithms are executed under the same stopping criterion of 20 iterations without any improvement in any solution or a maximum number of generated neighbours equal to $|N^3|$, that could likely indicate a premature convergence.

Moreover, as indicated by (Lalla-Ruiz et al. 2013) the GSPP mathematical formulation implemented in CPLEX is not able to provide even a feasible solution for some complex instances where other characteristics are considered. In this regard, we are interested in assessing the behaviour of DCM in such kind of instances. In doing so, a representative set of some of the largest instances proposed by (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012) has being tackled. The dimensions of the set of instances are 60 vessels and 5 berths. For evaluating the performance of DCM, a comparison among the best algorithmic methods used for those instances is provided:

- GSPP mathematical model (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012)
- Tabu Search (T^2S^*+PR) (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012)
- Decentralized Cooperative Metaheuristic (DCM)

Table 4 shows the computational results for the algorithms listed above. A column, MIN, with the best solution known for those instances is also included. As can be seen, GSPP is not able even to provide a feasible solution because it runs out of memory. Regarding the approximate solution approaches, DCM presents a better performance on average within an almost similar time requirement than the best approach (T^2S^*+PR) reported in the literature. Moreover, after analysing the use of the local search method, DCM points out 7/10 regions where the best solution known can be found after applying a local search. In this regard, DCM provides two new best objective function values that have not been reached before, namely, instances $i04$ and $i10$.

Conclusions and Further Research

The Dynamic Berth Allocation Problem (DBAP) has been addressed in this work. In order to efficiently solve it, we propose Decentralized Cooperative Metaheuristic (DCM). It is based on a decentralized grouping strategy for dividing a population of individuals into groups. The individuals within the same group cooperate by interchanging information. This grouping strategy improves the diversification of the search as well as the intensification in some regions of the search space through the sum of efforts among the individuals of the same group. Furthermore, the constrained relation for sharing information among individuals through the division of groups allows to reduce resources in comparison to ‘all to all’ communication.

It is concluded from the computational experimentation that the proposed algorithm is able to provide the optimal solutions within reasonable computational time for the instances proposed by (Cordeau et al. 2005). In this regard, the time advantage makes DCM suitable as a resolution method for being applied either individually or included into integrated schemes where the berth allocation is required. DCM is also appropriate for pointing out high promising regions in the search space.

Furthermore, the computational results show that DCM exhibits a better performance than other optimization algorithms presented in the literature for the DBAP. In this sense, the comparison with PSO and CS-SA remarks the benefits of applying a decentralized cooperative scheme for improving the processing times and detecting promising regions in the search space. Moreover, the experimentation over a representative set of instances, where the GSPP formulation implemented in CPLEX is not able to provide any feasible solution, shows that DCM is able to provide feasible solutions within small computational effort. In this regard, the comparison with the best approaches used for those instances indicates that DCM presents a better performance on average and provides two new best known solutions.

The analysis of different ways to exchange information among individuals and generate the groups will be a topic for future works. Moreover, we are also interested in assessing this approach in other berth allocation strategies and container terminal problems.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness (projects TIN2012-32608). Eduardo Lalla-Ruiz thanks the Canary Islands Government for the financial support he receives through his post-graduate grant.

References

Bajec, I. L.; Zimic, N.; and Mraz, M. 2005. Simulating flocks on the wing: the fuzzy approach. *Journal of Theoretical Biology* 233(2):199 – 220.

Bierwirth, C., and Meisel, F. 2010. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research* 202(3):615 – 627.

Buhrkal, K.; Zuglian, S.; Ropke, S.; Larsen, J.; and Lusby, R. 2011. Models for the discrete berth allocation problem: a computational comparison. *Transportation Research Part E* 47(4):461–473.

Christensen, C., and Holst, C. 2008. Berth allocation in container terminals. Master’s thesis, Department of Informatics and Mathematical Modelling, Technical university of Denmark. In Danish.

Christiansen, M.; Fagerholt, K.; Nygreen, B.; and Ronen, D. 2007. Chapter 4 maritime transportation. In Barnhart, C., and Laporte, G., eds., *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*. Elsevier. 189 – 284.

Cordeau, J.-F.; Laporte, G.; Legato, P.; and Moccia, L. 2005.

Table 4: Computational results for the instances provided by (Lalla-Ruiz, Melián-Batista, and Moreno-Vega 2012). Bold numbers indicate the best objective function value

	GSPP	MIN	T^2S^*+PR			MBO						DCM					
						w/LS			w/o LS			w/LS			w/o LS		
			Best	Gap (%)	t (s.)	Best	Gap (%)	t (s.)	Best	Gap (%)	t (s.)	Best	Gap (%)	t (s.)	Best	Gap (%)	t (s.)
i01	—	5753	**5753**	0,00	3,12	5761	0,14	2,81	5807	0,94	1,75	5759	0,10	3,71	5765	0,21	1,37
i02	—	6884	**6884**	0,00	3,20	**6884**	0,00	2,9	6932	0,70	1,75	**6884**	0,00	3,96	6886	0,03	1,45
i03	—	6780	6780	0,00	4,25	6792	0,18	2,63	6833	0,78	1,91	**6780**	0,00	4,17	6796	0,24	1,49
i04	—	5092	5105	0,26	2,30	5130	0,75	2,89	5223	2,57	1,54	**5092**	0,00	3,18	5102	0,20	1,48
i05	—	6715	**6715**	0,00	3,18	6723	0,12	2,45	6813	1,46	2,14	**6715**	0,00	3,68	6721	0,09	1,24
i06	—	6616	**6616**	0,00	3,53	6620	0,06	2,82	6686	1,06	1,75	6618	0,03	3,41	6632	0,24	1,29
i07	—	6011	**6011**	0,00	4,75	6017	0,10	3,7	6169	2,63	1,31	**6011**	0,00	3,58	6031	0,33	1,54
i08	—	4385	**4385**	0,00	3,77	4394	0,21	2,2	4472	1,98	1,63	**4385**	0,00	3,24	4396	0,25	1,52
i09	—	5235	**5235**	0,00	3,99	5251	0,31	2,82	5303	1,30	1,95	5237	0,04	3,15	5251	0,31	1,47
i10	—	7255	7281	0,36	3,62	7275	0,28	2,61	7340	1,17	1,83	**7255**	0,00	3,91	7266	0,15	1,56
		6072,6	6076,5	0,06	3,57	6084,7	0,21	2,78	6157,8	1,46	1,76	6073,6	0,02	3,60	6084,6	0,20	1,44

Models and tabu search heuristics for the berth-allocation problem. *Transportation Science* 39(4):526–538.

de Oliveira, R. M.; Mauri, G. R.; and Lorena, L. A. N. 2012. Clustering search for the berth allocation problem. *Expert Systems with Applications* 39(5):5499 – 5505.

Duman, E.; Uysal, M.; and Alkaya, A. F. 2012. Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences* 217:65–77.

Gutiérrez-Castro, J. P.; Melián Batista, B.; Moreno-Pérez, J. A.; Moreno-Vega, J. M.; and Ramos-Bonilla, J. 2008. Codea: An architecture for designing nature-inspired cooperative decentralized heuristics. In Krasnogor, N.; Nicosia, G.; Pavone, M.; and Pelta, D., eds., *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, volume 129 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg. 189–198.

Imai, A.; Nagaiwa, K.; and Chan, W. T. 1997. Efficient planning of berth allocation for container terminals in asia. *Journal of Advanced Transportation* 31(1):75–94.

Imai, A.; Nishimura, E.; and Papadimitriou, S. 2001. The dynamic berth allocation problem for a container port. *Transportation Research Part B: Methodological* 35(4):401–407.

Klotsman, M., and Tal, A. 2012. Animation of flocks flying in line formations.

Lalla-Ruiz, E.; Expósito-Izquierdo, C.; Melián-Batista, B.; and Moreno-Vega, J. M. 2013. A metaheuristic approach for the seaside operations in maritime container terminals. In *IWANN (Part II)*, volume 7903 of *Lecture Notes in Computer Science*. Springer.

Lalla-Ruiz, E.; Melián-Batista, B.; and Moreno-Vega, J. M. 2012. Artificial intelligence hybrid heuristic based on tabu search for the dynamic berth allocation problem. *Engineering Applications of Artificial Intelligence* 25(6):1132 – 1141.

Monaco, M. F., and Sammarra, M. 2007. The berth allocation problem: a strong formulation solved by a lagrangean approach. *Transportation Science* 41(2):265–280.

Ting, C.-J.; Wu, K.-C.; and Chou, H. 2013. Particle swarm optimization algorithm for the berth allocation problem. *Expert Systems with Applications* (0):–.

Umang, N.; Bierlaire, M.; and Vacca, I. 2013. Exact and heuristic methods to solve the berth allocation problem in bulk ports. *Transportation Research Part E: Logistics and Transportation Review* 54(0):14 – 31.

Yeo, H. J. 2010. Competitiveness of asian container terminals. *The Asian Journal of Shipping and Logistics* 26(2):225 – 246.

Combining heuristics to accelerate forward partial-order planning

Oscar Sapena, Eva Onaindía, Alejandro Torreño
Information Systems and Computation Dept.
Universitat Politècnica de València, Spain
E-mail: osapena,onaindia,atorreno@dsic.upv.es

Abstract

Most of the current top-performing planners are sequential planners that only handle total-order plans. Although this is a computationally efficient approach, the management of total-order plans restrict the choices of reasoning and thus the generation of flexible plans. In this paper we present FLAP2, a forward-chaining planner that follows the principles of the classical POCL (Partial-Order Causal-Link Planning) paradigm. Working with partial-order plans allows FLAP2 to easily manage the parallelism of the plans, which brings several advantages: more flexible executions, shorter plan durations (makespan) and an easy adaptation to support new features like temporal or multi-agent planning. However, one of the limitations of POCL planners is that they require far more computational effort to deal with the interactions that arise among actions. FLAP2 minimizes this overhead by applying several techniques that improve its performance: the combination of different state-based heuristics and the use of parallel processes to diversify the search in different directions when a plateau is found. To evaluate the performance of FLAP2, we have made a comparison with four state-of-the-art planners: SGPlan, YAHSP2, TFD and OPTIC. Experimental results show that FLAP2 presents a very acceptable trade-off between time and quality and a high coverage on the current planning benchmarks.

Introduction

Until the late 1990s, Partial-Order Planning (POP) was the most popular approach to AI planning. In this approach, based on the least-commitment philosophy, decisions about action orderings and parameter bindings are postponed until a decision must be taken. This is an attractive idea as avoiding premature commitments requires less backtracking during the search process. Nevertheless, the most recent total-order forward-chaining planners, such as LAMA (Richter and Westphal 2010), Fast Downward Stone Soup-1 (Helmert, Röger, and Karpas 2011) or SGPlan (Chen, Wah, and Hsu 2006), have demonstrated to be more efficient than partial-order planners, mainly due to:

- Search states can be generated much faster as there is no need to check *threats* (conflicts) among actions.
- They can generate complete state information and take advantage of powerful state-based heuristics or domain-specific control.

However, the general move towards state space search ignores some important benefits of partial-order planning:

- A partial-order plan offers more flexibility in execution.
- The search can be easily guided to improve the action parallelism in the plan.
- It is a very suitable approach in multi-agent planning systems, either with loosely (Kvarnström 2011) or tightly coupled (Torreño, Onaindía, and Sapena 2012) agents.
- It can easily be adapted to deal with temporal planning (Benton, Coles, and Coles 2012).

These desirable properties have led many current researchers to adopt POP techniques and to dedicate their efforts to improve the performance of this planning approach.

In this paper we present FLAP2, a partial-order forward-chaining planner that follows the design principles of POP, except for the delayed parameter binding, thus keeping the benefits of this successful approach. In spite of the inevitable increase of the search cost, we will show that FLAP2 improves the performance of existing partial-order planners and that it is competitive against some total-order planners. Particularly, FLAP2 returns solutions that represent a good trade-off between time and quality and it also offers a high coverage on the current planning benchmarks.

In the remainder of the paper we present the related work, some background, the planning approach of FLAP2 and a brief description of the other four planners that we will use in the experiments. Finally, we present an empirical evaluation of the performance of FLAP2 and we conclude with some final remarks.

Related work

Looking at the winners of the last International Planning Competitions (IPC'2011[1] and IPC'2008[2]), we can observe that the majority of planners participated in the sequential tracks. Fast Downward Stone Soup-1 (Helmert, Röger, and Karpas 2011), Selective Max (Domshlak, Karpas, and

[1]http://www.plg.inf.uc3m.es/ipc2011-deterministic
[2]http://ipc.informatik.uni-freiburg.de/

Markovitch 2010) and Merge and Shrink (Helmert et al. 2013) are optimal sequential planners built upon the classical Fast Downward planning system (Helmert 2006) based on heuristic search. LAMA (Richter and Westphal 2010), FF(h^s_a) (Keyder and Geffner 2008) and C^3 (Lipovetzky and Geffner 2009) are also forward state-space search planners that use powerful heuristics and compute (often suboptimal) solution plans very rapidly.

Planners that generate partial-order plans are basically found in temporal planning like SGPlan (Chen, Wah, and Hsu 2006), Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2009), DAE$_{YAHSP}$ (Khouadjia et al. 2013), YAHSP2 (Vidal 2011) and POPF2 (Coles et al. 2010). Temporal planning requires the ability of dealing with action parallelism due to the existence of temporally overlapping durative actions. With the exception of POPF2, all of these planners are built upon the parading of sequential planning. SGPlan, for example, uses Metric-FF (Hoffmann 2002) as a search engine, while DAE$_{YAHSP}$ and YAHSP2 are developed on top of the YAHSP planner (Vidal 2003). These three planners need an additional module to parallelize the obtained sequential plans and to enforce the temporal constraints of the problem. This separation between action selection and scheduling is doomed to fail in temporally expressive domains and suffer from severe drawbacks in temporally simple problems, as choosing the wrong actions might render the final solutions to be purely sequential and therefore of very low quality.

The approach taken by Temporal Fast Downward (TFD) is to perform forward search in the space of time-stamped states, where at each search state either a new action can be started or time can be advanced to the end point of an already running action, thereby combining action selection and scheduling (Eyerich 2012). This approach is usually very good in terms of quality but their coverage on current benchmarks is typically relatively low.

From the aforementioned planners, POPF2 is the only one that follows a partial-order planning approach. It is a forward planner that works with time, numbers and continuous effects. POPF2 records state information at each step of the plan (frontier state), like the negative interactions among the variable assignments, and updates the state accordingly. The frontier state is used to determine the set of applicable actions at each step of the plan. The late-commitment approach of POPF2 is based on delaying commitment to ordering decisions on the frontier state, thus ignoring other alternative choices that would come earlier, i.e. *before* the frontier state. Completeness, however, is ensured as search performs backtracking to find an alternative plan when necessary.

OPTIC (Benton, Coles, and Coles 2012) is the latest version of POPF2 and also handles soft constraints and preferences. The key of its good performance is the fast generation of the successor states during the search and the use of effective domain-independent heuristics. OPTIC yields high quality plans, although, computationally speaking, it is not that efficient as most of the sequential planners.

In this paper we present FLAP2, a partial-order forward-chaining planner that follows the design principles of POP. This approach is similar to the one of OPTIC, but introduces two important differences:

- OPTIC adds additional temporal constraints over the action to ensure that preconditions of the new actions are met in the frontier state. The approach of FLAP2 is more flexible as it does not commit to an action ordering if this is not required, just like traditional POCL planners do.
- FLAP2 can add new actions at any point in the current plan. OPTIC only adds actions after the frontier state, so that the new actions do not threaten the preconditions of earlier actions.

These two differences lead to a more flexible partial-order planner, although this improvement entails a higher computational effort to deal with the interactions among actions. However, FLAP2 outperforms OPTIC in many domains because it uses more sophisticated search methods and more powerful heuristics. Moreover, delaying commitment on the orderings of the actions allows FLAP2 to reach a solution from a higher number of search nodes, which also improves the search performance.

Background

For the purposes of this paper, we restrict ourselves to propositional planning tasks. A planning task is a tuple $T = \langle O,V,A,I,G\rangle$. O is a finite set of objects that model the elements of the planning domain over which the planning actions are applied. V is a finite set of state variables that model the states of the world. A state variable $v \in V$ is mapped to a finite domain of mutually exclusive values D_v. A value of a state variable in D_v corresponds to an object of the planning domain, that is, $\forall v \in V, D_v \subseteq O$. When a value is assigned to a state variable, the pair $\langle$variable, value$\rangle$ acts as a ground atom in propositional planning. A is the set of deterministic actions. I is the set of initial values assigned to the state variables and represents the initial state of the task. G is the set of goals of the task, i.e., the values the state variables are expected to take in the final state.

Definition 1. (Fluent) A ground atom or fluent is a tuple of the form $\langle v,d\rangle$ where $v \in V$ and $d \in D_v$, which indicates that variable v takes the value d.

Definition 2. (Action) An action $a \in A$ is a tuple $\langle PRE(a), EFF(a)\rangle$ where $PRE(a) = \{p_1,\dots,p_n\}$ is a set of fluents that represents the preconditions of a and $EFF(a) = \{e_1,\dots,e_m\}$ is a set of fluents that represents the consequences of executing a.

We define a partial-order plan for a planning task $T = \langle O, V, A, I, G\rangle$ as follows:

Definition 3. (Partial-order plan) A partial-order plan is a tuple $\Pi = \langle \Delta, OR, CL\rangle$. $\Delta \subseteq A$ is the set of actions in Π. OR is a set of ordering constraints ($\prec$) on Δ. CL is a set of causal links over Δ. A causal link is of the form $a_i \xrightarrow{\langle v,d\rangle} a_j$, meaning that precondition $\langle v,d\rangle$ of $a_j \in \Delta$ is supported by an effect of $a_i \in \Delta$.

This definition of a partial-order plan represents the mapping of a plan into a directed acyclic graph, where Δ represents the nodes of the graph (actions) and OR and CL are the sets of directed edges that describe the precedences and causal links among these actions, respectively.

The introduction of new actions in a partial plan may trigger the appearance of flaws. There are two types of flaws in a partial plan: preconditions that are not yet solved (or supported) through a causal link and threats. A threat over a causal link $a_i \xrightarrow{\langle v,d \rangle} a_j$ is caused by an action a_k that is not ordered w.r.t. a_i or a_j and modifies the value of v, i.e. $\langle v,d' \rangle \in EFF(a_k) \wedge d' \neq d$, making the causal link unsafe. Threats are addressed by introducing either an ordering constraint $a_k \prec a_i$, which is called *demotion* because the causal link is posted after the threatening action, or an ordering $a_j \prec a_k$, which is called *promotion* as the causal link is placed before the threatening action (Chapman 1987).

We define a *flaw-free* plan as a threat-free partial plan in which the preconditions of all the actions are supported through causal links. Given a flaw-free partial-order plan Π, we compute the frontier state, S_Π, resulting from the execution of Π in the initial state I. More formally:

Definition 4. (Frontier state) The frontier state S_Π of a flaw-free partial-order plan $\Pi = \langle \Delta, OR, CL \rangle$ is the set of fluents $\langle v,d \rangle$ achieved in Π by an action $a \in \Delta / \langle v,d \rangle \in EFF(a)$, such that any action $a' \in \Delta$ that modifies the value of v ($\langle v,d' \rangle \in EFF(a')/d \neq d'$) is not reachable from a by following the orderings and causal links in Π.

The basic POP algorithm starts by building an initial minimal plan containing two fictitious actions: the initial action a_{init}, with no preconditions and $EFF(a_{init}) = I$, and the goal action a_{goal}, with no effects and $PRE(a_{goal}) = G$. The algorithm works by following the next three steps until a solution is found: 1) select the next subgoal to achieve, 2) choose an action to support the selected subgoal and 3) solve the *threats* that arise as a consequence of the variables value modification.

In the following section we describe the planning algorithm of FLAP2 as well as the necessary modifications to adapt a POP algorithm to support a forward search. In our effort to maintain all the benefits of this approach, we tried to keep the changes as minimal as possible.

Planning algorithm

FLAP2 is a modified version of FLAP planner (Sapena, Onainda, and Torreño 2013). In the following subsections we briefly describe the planning approach of FLAP and the changes made in FLAP2 to improve its performance, respectively.

FLAP's working scheme

FLAP implements an A* search, as the standard textbook algorithm in (Russell and Norvig 2009), guided by an evaluation function. A search node is a partial-order plan and the starting node is the initial initial plan $\Pi_0 = \langle \{a_{init}\}, \emptyset, \emptyset \rangle$. Although Π_0 does not contain the fictitious goal action a_{goal}, this action is available to be added to the plan as the rest of actions in A, i.e. $a_{goal} \in A$. In fact, a solution plan is found when a_{goal} is inserted in the plan.

FLAP follows two steps at each iteration of the search process until a solution plan is found: **a)** it selects the best node, Π_i, from the set of open nodes according to the evaluation function, and **b)** all possible successors of Π_i are generated, evaluated and added to the list of open nodes. FLAP considers that Π_j is a successor of a plan Π_i if the following conditions are met:

- Π_j adds a new action a_j to Π_i, i.e., $\Delta_j = \Delta_i \cup \{a_j\}$
- All preconditions of a_j are supported with actions in Π_i by inserting the corresponding causal links: $\exists a_i \xrightarrow{p} a_j \in CL_j, a_i \in \Delta_i, \forall p \in PRE(a_j)$.
- All threats in Π_j are solved through promotion or demotion by adding new ordering constraints; the result is that Π_j is a flaw-free plan.

The forward-search approach of FLAP allows to use state-based heuristics, which are much more informed than classical POP-based heuristics. In order to evaluate a partial-order plan Π, FLAP computes the frontier state S_Π. It uses three different heuristics:

- h_{DTG}. A Domain Transition Graph (DTG) of a state variable is a representation of the ways in which the variable can change its value (Helmert 2004). Each transition is labeled with the necessary conditions for this to happen, i.e. the common preconditions to all the actions that induce the transition. These graphs are used to estimate the cost of the value transition required to support an action precondition, and the *Dijkstra* algorithm is applied to calculate the length of the shortest path in the DTG that causes the transition. The h_{DTG} heuristic returns the minimum number of actions in a relaxed plan, where delete effects are ignored, that achieves the problem goals from S_Π. Actions in the relaxed plans are selected according to the sum of the estimated cost of their preconditions.
- h_{FF}. FLAP also makes use of the traditional FF heuristic function h_{FF} (Hoffman and Nebel 2001), which builds a relaxed plan by ignoring the delete effects of the actions and returns its number of actions. The actions of this plan are selected according to their levels in the relaxed planning graph.
- h_{LAND_DTG} and h_{LAND_FF}. Landmarks are fluents that must be achieved in every solution plan (Hoffmann, Porteous, and Sebastia 2004; Sebastia, Onaindía, and Marzal 2006). FLAP computes a landmark graph and uses this information to calculate heuristic estimates: since all landmarks must be achieved in order to reach a goal, the goal distance can be estimated through the set of landmarks that still need to be achieved from the state being evaluated onwards. Once we have the set of non-supported landmarks, the heuristic value is the result of estimating the cost of reaching these landmarks with either h_{DTG} or h_{FF}. This way, FLAP has two versions of the landmarks heuristic, called h_{LAND_DTG} and h_{LAND_FF}, respectively.

For evaluating a plan $\Pi = \langle \Delta, OR, CL \rangle$, FLAP defines two different evaluation functions:

- $f_{FF}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND\_FF}(\Pi) + w_3 * h_{FF}(S_\Pi)$
- $f_{DTG}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND\_DTG}(\Pi) + w_3 * h_{DTG}(S_\Pi)$

$g(\Pi)$ measures the cost of Π in number of actions, i.e. $g(\Pi) = |\Delta|$. The weights in the two functions are set to $w_1 = 1$, $w_2 = 4$ and $w_3 = 2$. FLAP uses both evaluation functions to simultaneously explore different parts of the search space, thus defining two main search processes.

Additionally, a new A* search is started in parallel when one of the two main search processes is stuck in a plateau, i.e. the evaluation function does not improve after several iterations. The goal of this new search is not to escape from the plateau, but to find a solution plan starting from the frontier state of the best node found so far, as this node is more likely to be closer to a solution than the initial state. The parallel search is cancelled if the main search manages to leave the plateau.

FLAP planner is sound and complete since all possible successors are considered at each point and, when a_{goal} is added to the plan, the support of all problem goals as well as the plan consistency is guaranteed.

Performance improvements in FLAP2

In order to improve the performance of FLAP we performed an analysis of the search process, specifically of the behaviour of the heuristics in domains with different characteristics. This analysis is shown in the following subsection. Finally, in a second subsection, we describe the modifications introduced in FLAP2 according to the conclusions of the analysis.

Analysis of heuristics and the plateau escaping method. Regarding h_{DTG}, we found that this heuristic is more informative than h_{FF} in planning domains that satisfy some specific characteristics:

- the state variables have rather large domains containing multiple different values, and
- the DTGs of these variables are sparse graphs.

In Figure 1 we can observe an example of the DTGs of two variables: (*empty t1*) and (*at d1*). There are only two values, *true* and *false*, in the domain of (*empty t1*), meaning that the cabin of the truck *t1* can be empty or not. On the contrary, the position of driver *d1* can take several different values: location 1 (*l1*), cities 1, 2 and 3 (*c1*, *c2* and *c3*) and truck 1 (*t1*). The values of h_{DTG} obtained from the DTG of variable (*empty t1*) are not very accurate because there is only one transition that makes the variable change from *true* to *false*, and this transition is derived by many different actions, particularly all actions in which *d1* boards *t1* at any possible city. Hence, selecting the action to be included in the relaxed plan to support this transition is not an easy task and a wrong decision would worsen the quality of the heuristic.

On the contrary, the DTG of variable (*at d1*) is more informative. For example, the path to change its value from *l1* to *c1* contains three transitions: $l1 \rightarrow c2 \rightarrow t1 \rightarrow c1$ or $l1 \rightarrow c3 \rightarrow t1 \rightarrow c1$, depending on the position of the truck. Moreover, each transition in the path is produced by a single action and thus the correct action is always selected by h_{DTG} when computing the relaxed graph. Our conclusion is that h_{DTG} performs slightly better than h_{FF} in transportation-like domains, such as *DriverLog* or *ZenoTravel*, where the DTGs of several variables are rather large sparse graphs. For the rest of domains, h_{FF} clearly outperforms h_{DTG}.

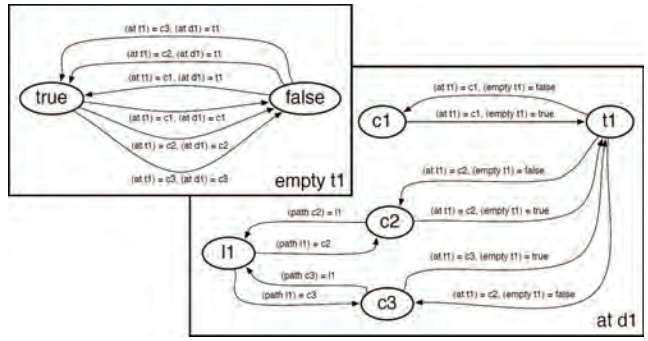


Figure 1: DTGs of variables (*empty t1*), the state of the cabin of truck *t1*, and (*at d1*), the location of driver *d1*, in a *DriverLog* problem example.

h_{DTG} also presents some limitations in non-reversible domains, where the effects of some actions cannot be undone. The search space of these domains may contain dead-ends, i.e., nodes with frontier states from which the problem goals are unreachable. h_{FF} is able to detect many of these dead-ends as it builds a relaxed planning graph at each node of the search tree: if any of the problem goals is not reachable in the relaxed graph, the node is a dead-end. On the contrary, h_{DTG} only detects a dead-end state if no transition path can be found in the DTGs that transforms the value of a variable into its final value. Then, h_{DTG} does not take into account the interactions between variables to detect dead-ends. This limitation can be alleviated by computing mutex fluents in a preprocessing stage, i.e. fluents that cannot be true in a state at the same time. Improvements in the h_{DTG} heuristic is an issue we want to address in future works.

On the other hand, the landmark-based heuristic, h_{LAND}, is very informative in domains which contain a large number of atomic landmarks. An atomic landmark, which is a single fluent that every solution plan must achieve at some point, is usually much more accurate than a disjunctive landmark since a disjunctive landmarks is less restrictive. In FLAP, h_{LAND} (both h_{LAND_FF} and h_{LAND_DTG}), is always used in combination with h_{FF} or h_{DTG}. However, we observed that, when the number of atomic landmarks is similar or greater than the number of disjunctive landmarks, h_{LAND} is informative enough to be used as a stand-alone heuristic.

These three heuristics (h_{DTG}, h_{FF} and h_{LAND}) assess the quality of a plan by estimating the number of actions required to reach the problem goals. However, this does not seem to be the most appropriate approach for a planner that works with concurrent actions. When dealing with partial-order plans, optimizing the plan duration (makespan) is always preferable if we aim to improve the plan parallelism. Even so, as we will see in the Experimental Results section, the quality of the plans generated by FLAP2 w.r.t. the makespan is quite good because it exploits the advantages of working directly with concurrent actions. However, adapting the heuristics to evaluate the plans according to their makespan could significantly improve the quality of the solutions, a research line we intend to explore in the future.

Finally, we analyzed the plateau escaping mechanism of FLAP. The parallel search process started when one of the main search processes gets stuck in a plateau is not enough to solve some difficult problems as this new search may also get stuck in another plateau.

Modifications in the search process of FLAP. Taking all the above considerations into account, we designed FLAP2 as follows. First of all, we check if sufficient information can be extracted from the landmarks graph. We define $\lambda = |disjunctive\_landmarks|/|atomic\_landmaks|$, i.e. the ratio between the number of disjunctive landmarks and the number of atomic landmarks; when no atomic landmarks are found, $\lambda = \infty$. We consider that there is enough information when $\lambda \leq 1.2$.

When h_{LAND} is not informative enough, $\lambda > 1.2$, FLAP2 starts a single main A* search with the f_{FF} evaluation function with $w_1 = 1$, $w_2 = 4$ and $w_3 = 2$. The weight for h_{LAND_FF}, w_2, is higher to make up for the poor heuristic values returned by h_{LAND}. Unlike FLAP, in FLAP2 we do not start a second main search with h_{DTG} because, as we said in the previous section, h_{DTG} is only worth using in transportation-like domains and thereby a general use of h_{DTG} does not compensate for the overhead in computation time and memory consumption. Consequently, h_{DTG} is only used in FLAP2 when search needs to be diversified due to the existence of a plateau.

The search process of FLAP2 uses a variable, Π_{best}, that stores the node with the best heuristic value found so far. Initially Π_{best} is set to the initial plan, i.e. $\Pi_{best} = \Pi_0$. When a search node with a better heuristic value than the one of Π_{best} is found, Π_{best} is updated to this node. We consider that the search is stuck in a plateau when Π_{best} has not been updated in several iterations. In this case, two new search processes are started from the frontier state of Π_{best} to increase the chances of escaping from the plateau. The first one uses f_{FF} and the second one the f_{DTG} evaluation function, both with the same weight values than the ones used for the main search. By using two new searches with different heuristic functions, we allow to diversify the search directions and find a plateau exit more effectively.

A *child* search works equally as the main search. In fact, when a child search finds a plateau, it also starts two new search processes. This behaviour can be observed in Figure 2. When a search manages to escape from a plateau, i.e. when a node with a heuristic value better than the value of Π_{best} is found, then its two child processes are terminated.

In the case that h_{LAND} is informative enough, $\lambda \leq 1.2$, FLAP2 starts a search process with f_{FF} and a second main A* search with the following evaluation function: $f_{LAND\_FF}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND\_FF}(\Pi)$, with $w_1 = 1$ and $w_2 = 1$. In this case, h_{LAND_FF} is used as a stand-alone heuristic function and it is given a small weight because this is already a very informative heuristic when many atomic landmarks are extracted from the problem. In this case, if a plateau is found, two child searches are started in the same way as for the case of $\lambda > 1.2$, but now we use f_{FF} with

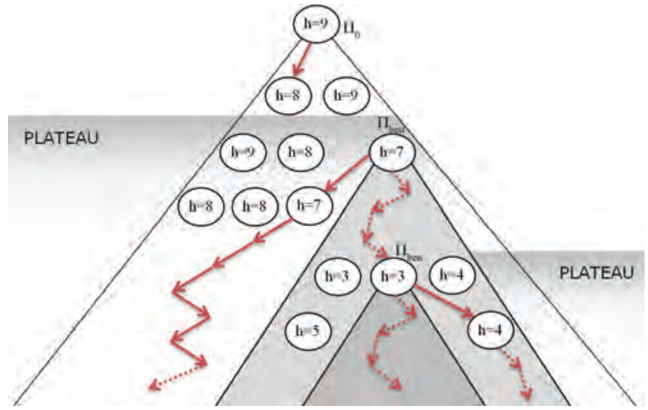


Figure 2: Parallel A* search processes for plateau escaping.

$w_1 = 1$, $w_2 = 1$ and $w_3 = 1$, and $f_{LAND\_DTG}(\Pi) = w_1 * g(\Pi) + w_2 * h_{LAND\_DTG}(\Pi)$ with $w_1 = 1$ and $w_2 = 1$. Table 1 summarizes the configuration of the search processes of FLAP2 according to the value of λ.

	$\lambda > 1.2$	$\lambda \leq 1.2$
Main search	$f_{FF}, w_1 = 1, w_2 = 4, w_3 = 2$	$f_{FF}, w_1 = 1, w_2 = 4, w_3 = 2$ $f_{LAND\_FF}, w_1 = w_2 = 1$
Child search	$f_{FF}, w_1 = 1, w_2 = 4, w_3 = 2$ $f_{DTG}, w_1 = 1, w_2 = 4, w_3 = 2$	$f_{FF}, w_1 = w_2 = w_3 = 1$ $f_{LAND\_DTG}, w_1 = w_2 = 1$

Table 1: Configuration of the search processes in FLAP2.

This configuration has been fixed as the result of an extensive experimental analysis and it offers a good trade-off between search time and plan quality in most of the problems. Other settings significantly improve the performance in some domains, but they are less robust since they worsen the results in the other ones.

The mechanism of parallel searches implemented in FLAP2 yields very good results but it can lead to an exponential growth in the number of simultaneous processes. However, this problem does not usually occur in practice since the number of simultaneous search processes that exceeded the number of processing cores (8 in our test computer) only occurred in a few problems. Specifically, we tested FLAP2 in 244 problems from 10 different domains and only 7 of them required more than 8 search processes at the same time. And yet, this did not prevent FLAP2 from finding a solution plan for these problems.

Temporal planning systems

In order to evaluate the performance of FLAP2, we selected four current top-performing planners that return parallel plans: SGPlan, YAHSP2, OPTIC and TFD. All of them are temporal planners as only this type of planners are currently able to synthesize plans with concurrent actions. These planners are briefly described in the following subsections.

SGPlan

SGPlan is designed to solve both temporal and non-temporal planning problems specified in PDDL3 with soft goals, derived predicates or ADL features. SGPlan was the winner of the temporal satisficing track in the sixth planning competition (IPC 2008).

For each subgoal, SGPlan uses search-space reduction to eliminate irrelevant actions and solves it using a modified version of Metric-FF. If it fails to find a feasible plan within a time limit, SGPlan aborts the run of Metric-FF and tries to decompose the problem further. It first applies landmark analysis to decompose and solve the subproblem and, if it is unsuccessful in solving the subproblem, it tries path optimization for numerical and time initial literals problems to further partition the subproblem. When a separate subplan has been computed for each subgoal, SGPlan merges them into a consistent plan.

This partition-and-resolve process has proved to be very successful in a wide range of domains, although its performance worsens in domains in which there are strong interactions between the subgoals. This is because the actions that achieve the subgoals are highly related, making it more difficult to obtain a significant fraction of global constraints.

YAHSP2

YAHSP is a heuristic planner for suboptimal STRIPS domains. The heuristic is similarly computed to FF heuristic, but used in a different way. When a state is being evaluated, the heuristic computes a relaxed plan where delete effects are ignored. The beginning actions of the relaxed plan that form a valid plan are applied to the state being evaluated, resulting in another state that will often bring the search closer to a solution state. The states computed this way are called lookahead states. YAHSP uses this lookahead strategy in a complete best-first search algorithm in which the helpful actions (Hoffman and Nebel 2001) computed by the heuristic are prioritized.

YAHSP2 is designed as a simplified version of YAHSP. The main modifications are the following:

- The relaxed plans used to build the lookahead plans are computed directly from a critical path heuristic like h^{add}, avoiding the need of complex data structures to build planning graphs.
- The heuristic value of states is no longer the length of the relaxed plans, but the h^{add} value of the goal set.
- Some refinements, such as the use of helpful actions, are abandoned due to the lack of robustness.

This minimalist approach makes YAHSP2 to be an extremely fast planner with a wide coverage on the current benchmarks. In fact, a multi-core version of this planner was the runner-up ex-aequo in the temporal satisficing track of the IPC 2011. The lack of optimizations on the plan quality, however, leads to the generation of overlength plans in many problem instances.

OPTIC

Unlike SGPlan and YASHP2, OPTIC does not handle two independent processes for action selection and temporal scheduling of the actions, thus obtaining high quality plans with respect to the makespan. It is an extended version of POPF2, which was the runner-up ex-aequo in the temporal satisficing track of the IPC 2011.

OPTIC is a forward-chaining temporal planner that incorporates some ideas from partial-order planning: during search, when applying an action to a state, it seeks to introduce only the ordering constraints necessary to resolve threats, rather than insisting the new action occurs after all of those already in the plan. OPTIC supports a substantial portion of PDDL 2.1 level 5, including actions with (linear) continuous numeric effects and effects dependent on the durations of the actions. It also handles soft constraints and preferences.

Temporal Fast Downward (TFD)

TFD, the runner-up in the temporal satisficing track of the IPC 2008, is a variant of the propositional Fast Downward planning system. It introduces several adjustments to cope with temporal and numeric domains and no longer uses the causal graph heuristic. Instead, it makes use of the context-enhanced additive heuristic (CEA) proposed by Geffner (Geffner 2007), which is a generalization of both the causal graph heuristic and the additive heuristic.

TFD uses a greedy best-first search approach enhanced with deferred heuristic evaluation. Besides the values of the state variables, the time-stamped states in the search space contain a real-valued time stamp as well as information about scheduled effects and conditions of currently executed actions. A transition from one time-stamped state to another is accomplished by either **a)** adding an applicable action starting at the current time point, applying its start effects and scheduling its end effects as well as its over-all and end conditions, or **b)** letting time pass until the next scheduled happening and applying effects scheduled for the new time point and deleting expired conditions. This integrated process of action selection and time scheduling yields very good results in terms of plan quality.

Experimental results

In this section we compare the performance of FLAP2 against the four aforementioned planners. Due to the different characteristics of these planners, we have divided this section in two subsections:

- Comparison of FLAP2 with SGPlan and YAHSP2, two sequential planners that apply a scheduler to parallelize the plans at a later stage. This approach is extremely fast but finds more difficulties in producing plans of good quality regarding the makespan.
- Comparison of FLAP2 with OPTIC and TFD, two planners that merge the action selection and the scheduling process. Working with partial-order planners allows to compute more flexible plans, with a better makespan, but slows down the search process.

In both cases, we selected six temporal domains from the International Planning Competitions (IPC), setting the duration of all actions to 1 as FLAP2 is still unable to work with durative actions. The IPCs provide an extensive set of benchmarking problems to assess the state of the art in the field of planning (Linares, Jiménez, and Helmert 2013).

We observed that the behaviour of these planners varies greatly depending on the level of interaction between the problem goals. For this reason we selected three domains with strong dependencies between the goals, *BlocksWorld*, *Depots* and *DriverLog*, and three domains with rather independent goals, *Satellite*, *Rovers* and *ZenoTravel*. These domains are described below:

- *Blocksworld*: this domain, presented in the IPC 2000, consists of a set of blocks that must be arranged to form one or more towers. We used a variation of this domain where several robot arms are used to handle the blocks, thus allowing parallel actions in the plans.
- *Depots*: this domain, introduced in the IPC 2002, combines a transportation-like problem with the *Blocksworld* domain.
- *Driverlog*: this domain, used in the IPC 2002, involves transportation, but vehicles need a driver before they can move.
- *Satellite*: this domain, used in the IPC 2004, involves satellites collecting and storing data using different instruments to observe a selection of targets.
- *Rovers*: used in the IPC 2006, the objective is to use a collection of mobile rovers to traverse between waypoints on the surface of Mars, carrying out a variety of data-collection missions and transmitting data back to a lander.
- *Zenotravel*: in this domain, presented in the IPC 2002, people must embark onto planes, fly between locations and then debark, with planes consuming fuel at different rates according to their speed of travel.

Testing was performed on a 2.3 GHz i7 computer with 12 GB of memory running Ubuntu 64-bits. In the presented results we only consider the first plan returned by the planners, as most of them do not continue searching for better plans. Each experiment was limited to 30 minutes of wall-clock time.

FLAP2 vs. SGPlan and YAHSP2

Table 2 shows the number of solved problems and the average time employed by these planners to find the first solution. Average times are calculated considering only those problems that were solved by the three planners.

As it can be observed, FLAP2 solves more problems and shows a more stable behaviour. Both, SGPlan and YAHSP2 present some difficulties in domains with strong interactions between the goals (*BlocksWorld*, *Depots* and *DriverLog*), but they are significantly faster in the other three domains. The landmarks heuristic and the plateau escaping mechanism of FLAP2 are very helpful to deal with strong dependencies among the goals. FLAP2 also easily solves the problems from the *Rovers*, *Satellite* and *ZenoTravel* domains, but the overhead to cope with threats among actions together with a higher branching factor prevents FLAP2 from being as faster as SGPlan or YASHP2 in these domains.

		FLAP2		SGPlan		YAHSP2	
Domain	Prob	Solved	Average time	Solved	Average time	Solved	Average time
BlocksWorld	34	34	0.40	22	5.80	34	57.78
Depots	20	20	1.99	19	0.15	16	121.24
DriverLog	20	20	3.38	17	1.02	20	0.11
Satellite	20	20	4.19	20	0.07	20	0.05
Rovers	20	20	4.21	20	0.04	20	0.04
ZenoTravel	20	20	6.91	20	0.23	20	0.16
Total	134	134	3.52	118	1.22	130	29.90

Table 2: Number of problems solved and average time (in seconds) of FLAP2, SGPlan and YAHSP2.

Regarding the plan quality, Figures 3 and 4 show the makespan of the plans computed by the three planners. The results are normalized by the makespan of the plans obtained by FLAP2 for a better viewing. This way, a value of 2 indicates a plan with a makespan twice as much as the makespan of FLAP2, and a value of 0.5 a plan two times shorter.

In general, FLAP2 generates plans with better quality than SGPlan and YAHSP2. SGPlan produces slightly worse plans, 1.36 times longer in the six domains. The plan quality of YAHSP2 is much worse as the generated plans are 2.4 times longer than FLAP2 on average.

FLAP2 vs. OPTIC and TFD

Table 3 shows the number of solved problems and the average makespan of FLAP2, OPTIC and TFD. As it can be observed, FLAP2 also solves more problems than OPTIC and TFD. The average makespan is computed taking into account only those problems that were solved by the three planners. Regarding the makespan, FLAP2 is in a intermediate position between TFD, that produces plans of very good quality, and OPTIC.

		FLAP2		OPTIC		TFD	
Domain	Prob	Solved	Average makespan	Solved	Average makespan	Solved	Average makespan
BlocksWorld	34	34	10.92	24	15.88	34	7.25
Depots	20	20	11.93	11	14.86	10	9.10
DriverLog	20	20	14.47	15	12.93	16	13.40
Satellite	20	20	17.00	16	11.50	20	14.25
Rovers	20	20	12.65	20	13.35	17	14.29
ZenoTravel	20	20	8.56	16	8.31	20	8.31
Total	134	134	12.59	102	12.81	117	11.10

Table 3: Number of problems solved and average makespan of FLAP2, OPTIC and TFD.

In Figures 5 and 6 we show the computation time of FLAP2, OPTIC and TFD to find the first solution plan. For the average times shown in these figures, we considered only the problems that the three planners have managed to solve. FLAP2 is much faster than OPTIC in the *BlocksWorld*, *Depots*, *Satellite* and *ZenoTravel* domains. On the contrary, OPTIC is slightly faster than FLAP2 in the *Rovers* domain. On average, OPTIC is 113.94 times slower than FLAP2 in all the six domains. TFD is also slower than FLAP2, especially

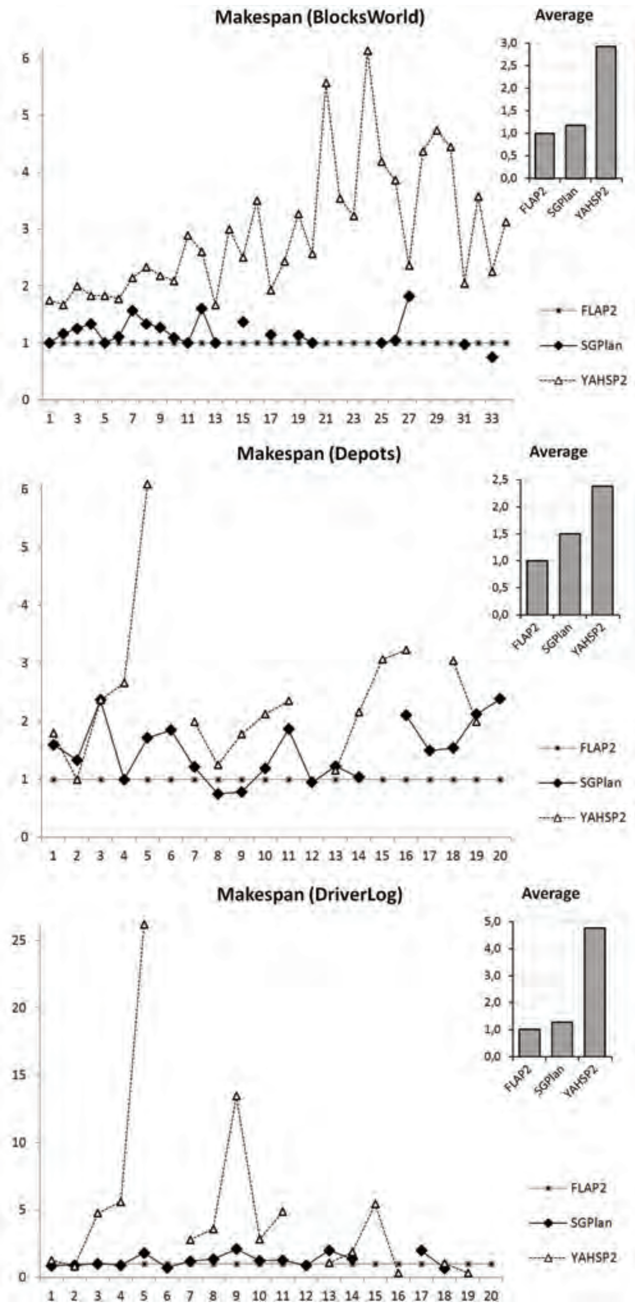


Figure 3: Makespan of the plans of SGPlan and YAHSP2, normalized by the makespan of the plans of FLAP2, for the *BlocksWorld*, *Depots* and *DriverLog* domains.

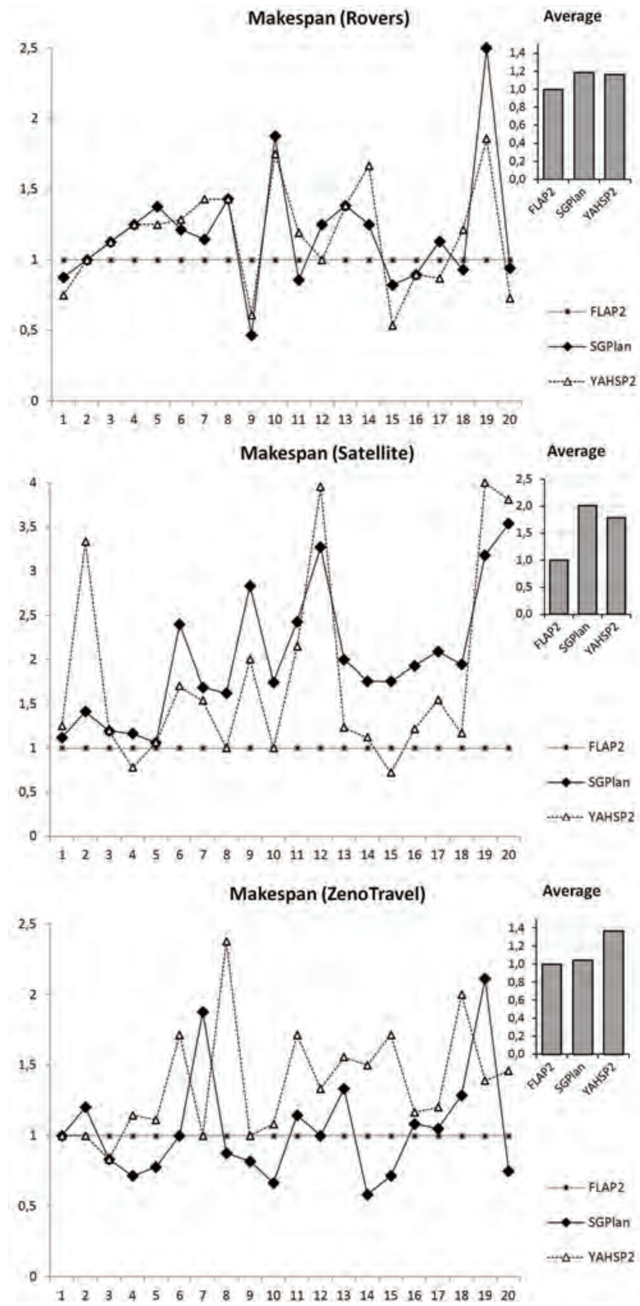


Figure 4: Makespan of the plans computed by SGPlan and YAHSP2, normalized by the makespan of the plans of FLAP2, for the *Rovers*, *Satellite* and *ZenoTravel* domains.

in the *Depots* and *DriverLog* domains. On average, TFD is 45.3 times slower than FLAP2 in all the six domains.

In summary, we can conclude that FLAP2 is very competitive in comparison with these four top-performing planners. It solves more problems than SGPlan, YAHSP2, OPTIC and TFD in the tested domains. FLAP2 also produces plans of better quality than the sequential planners SGPlan and YAHSP2, and is far more faster than OPTIC and TFD, planners that, like FLAP2, handle partial-order plans.

Conclusions

The flexibility of the Partial-Order Planning (POP) paradigm allows for the generation of high-quality parallel plans. However, current sequential planners outperform partial-order planners because they require less computational effort as they not need to cope with interactions among actions and can use very effective state-based heuristics.

In this paper we present FLAP2, an improved version a

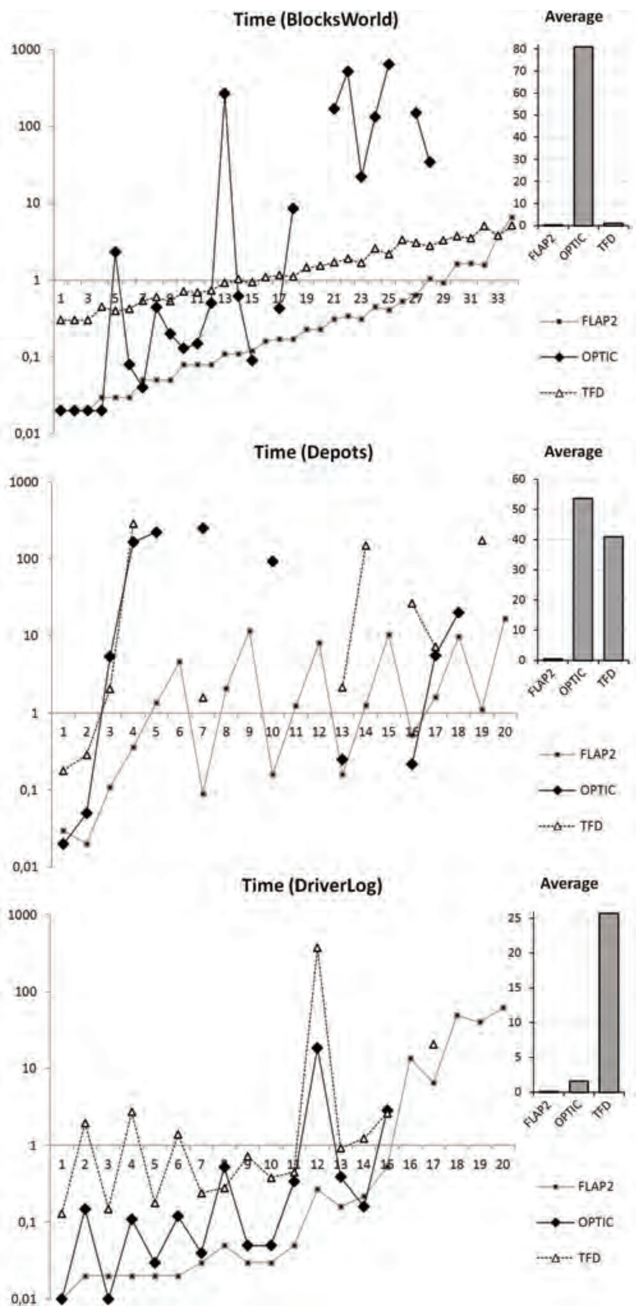


Figure 5: Planning time (in seconds) of FLAP2, OPTIC and TFD in the *BlocksWorld*, *Depots* and *DriverLog* domains.

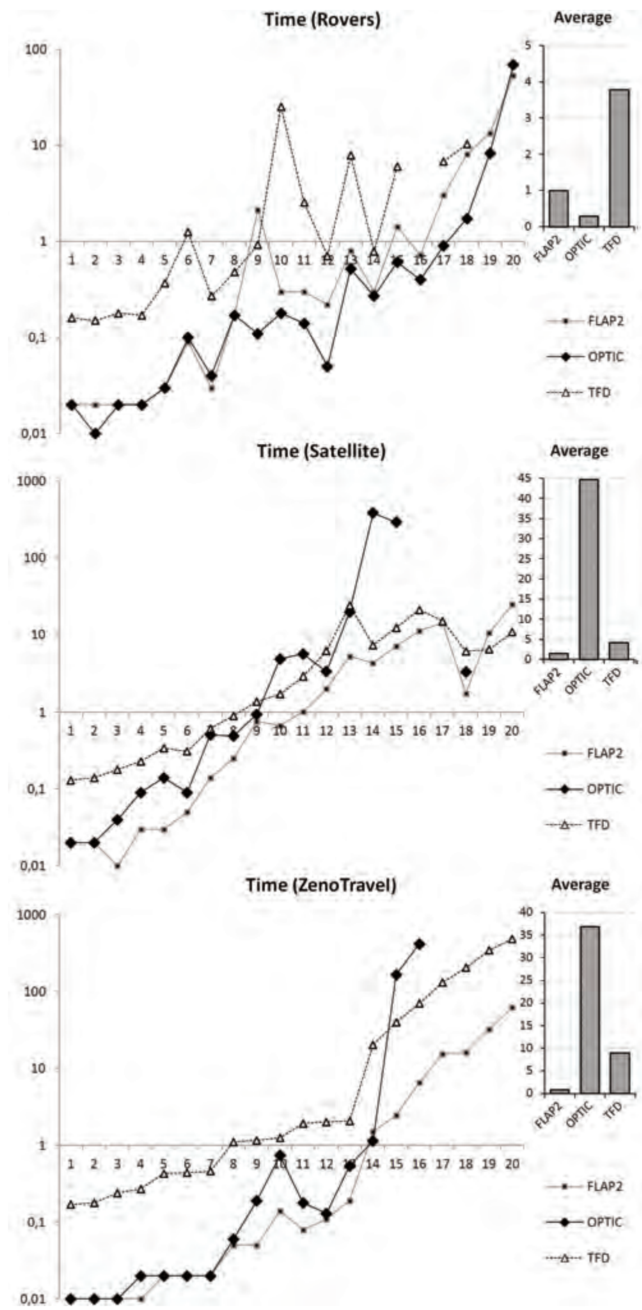


Figure 6: Planning time (in seconds) of FLAP2, OPTIC and TFD in the *Rovers*, *Satellite* and *ZenoTravel* domains.

FLAP. FLAP is a forward partial-order planner that combines three different heuristics to guide the search and implements a novel plateau-escaping method that diversifies the search in different directions. FLAP2 changes the way the heuristics are combined and applies a recursive method to deal with plateaus, thus significantly improving the planning performance.

We compared FLAP2 with SGPlan, YAHSP2, OPTIC and Temporal Fast Downward (TFD), four top-performing planners that can generate plans with concurrent actions. Like FLAP2, OPTIC and TFD handle partial-order plans, combining the action selection and the scheduling processes. On the contrary, SGPlan and YAHSP2 are total-order planners that parallelize the computed plans at a later stage.

FLAP2 is the only one that was able to solve all the problems in the selected benchmark set. Regarding the makespan (plan duration), partial-order planners generate plans of much better quality than the total-order planners. Particularly, FLAP2 has shown to obtain plans of very good quality, only surpassed by TFD, which is able to produce plans with a slightly better makespan. As for the planning time, FLAP2 has shown to be competitive with the sequential planners, SGPlan and YAHSP2, especially in domains with strong interactions between the problem goals, and far more faster than the other partial-order planners, OPTIC and TFD.

As a future extension, we intend to investigate the adaptation of the heuristic functions of FLAP2 to optimise the makespan and to mitigate the problem of h_{DTG} with dead-end states in non-reversible domains. Then, we want to exploit the good performance of FLAP2 and its flexibility as a partial-order planner to develop a new version for dealing with temporal planning problems.

Acknowledgements

This work has been partly supported by the Spanish MICINN under projects Consolider Ingenio 2010 CSD2007-00022 and TIN2011-27652-C03-01.

References

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. *International Conference on Automated Planning and Scheduling (ICAPS)* 2–10.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333–377.

Chen, Y.; Wah, B.; and Hsu, C. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research* 26:323–369.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. *International Conference on Automated Planning and Scheduling (ICAPS)* 42–49.

Domshlak, C.; Karpas, E.; and Markovitch, S. 2010. To max or not to max: Online learning for speeding up optimal planning. *AAAI Conference on Artificial Intelligence*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Eyerich, P. 2012. Preferring properly: Increasing coverage while maintaining quality in anytime temporal planning. *Proceedings of the European Conference on Artificial Intelligence (ECAI)* 312–317.

Geffner, H. 2007. The causal graph heuristic is the additive heuristic plus context. *Proceedings of ICAPS Workshop on Heuristics for Domain-Independent Planning*.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2013. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. *Proceedings of the ICAPS-2011 Workshop on Planning and Learning (PAL)* 28–35.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. *International Conference on Automated Planning and Scheduling (ICAPS)* 161–170.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Articial Intelligence Research* 26:191–246.

Hoffman, J., and Nebel, B. 2001. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Hoffmann, J. 2002. Extending FF to numerical state variables. *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI)* 571–575.

Keyder, E., and Geffner, H. 2008. Heuristics for planning with action costs revisited. *European Conference on Artificial Intelligence (ECAI)* 588–592.

Khouadjia, M.; Schoenauer, M.; Vidal, V.; Dréo, J.; and Savéant, P. 2013. Multi-objective AI planning: Comparing aggregation and pareto approaches. *Proceedings of the 13th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP)* 7832:202–213.

Kvarnström, J. 2011. Planning for loosely coupled agents using partial order forward-chaining. *International Conference on Automated Planning and Scheduling (ICAPS)* 138–145.

Linares, C.; Jiménez, S.; and Helmert, M. 2013. Automating the evaluation of planning systems. *AI Communications* 26(4):331–354.

Lipovetzky, N., and Geffner, H. 2009. Inference and decomposition in planning using causal consistent chains. *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS)* 217–224.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 29(1):127–177.

Russell, S. J., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Sapena, O.; Onainda, E.; and Torreño, A. 2013. Forward-chaining planning with a flexible least-commitment strategy. *Congreso Internacional de la Asociacin Catalana de Inteligencia Artificial (CCIA)* 256:41–50.

Sebastia, L.; Onaindía, E.; and Marzal, E. 2006. Decomposition of planning problems. *AI Communications* 19(1):49–81.

Torreño, A.; Onaindía, E.; and Sapena, O. 2012. An approach to multi-agent planning with incomplete information. *European Conference on Artificial Intelligence (ECAI)* 242:762–767.

Vidal, V. 2003. A lookahead strategy for solving large planning problems. *Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI)* 1524–1525.

Vidal, V. 2011. YAHSP2: Keep it simple, stupid. *Proceedings of the 7th International Planning Competition (IPC-2011)* 83–90.

A Constraint-based Approach for Planning UAV Activities

Christophe Guettier, François Lucas
christophe.guettier@sagem.com
francois.lucas@artelys.com

Abstract

Unmanned Aerial Vehicles (UAV) represent a major advantage in defense, disaster relief and first responder applications. UAV may provide valuable information on the environment if their Command and Control (C2) is shared by different operators. In a C2 networking system, any operator may request and use the UAV to perform a remote sensing operation. These requests have to be scheduled in time and a consistent navigation plan must be defined for the UAV. Moreover, maximizing UAV utilization is a key challenge for user acceptance and operational efficiency. The global planning problem is constrained by the environment, targets to observe, user availability, mission duration and on-board resources. This problem follows previous research works on automatic mission Planning & Scheduling for defense applications. The paper presents a full constraint-based approach to simultaneously satisfy observation requests, and resolve navigation plans.

Introduction

Using Unmanned Aerial Vehicles (UAV) has become a major trend in first responder, security and defense areas. UAV navigation plans are generally defined during mission preparation. However, during mission preparation or execution, different users can request for additional observations to be performed by the UAV. It is then necessary to insert these actions in UAV navigation plans. The user must deal with constraints that will impact the overall plan feasibility, such as observation preconditions, duration of the UAV mission or resource consumption. For example, a rotorcraft can easily perform an observation using stationary flight, but has poor endurance. In turn, a fixed wing can perform longer missions but needs to orbit around a waypoint to acquire and observe a target. This paper addresses vehicle planning issues, managing constraints composed of mission objectives, execution time and resource requirements. In this problem, UAVs can communicate with the network to transmit remote videos to ground manned vehicles on ground.

The optimization problem consists in finding the path that maximizes the overall mission efficiency while ensuring mission duration and resource consumption. The structure of consumption and observation constraints make the problem difficult to model and hard to solve. Determining the shortest path may not lead to the most efficient one, since observation requests may occur for various different places. The paper proposes a constraint model for UAV activity optimization, before and during mission execution. It is formulated as a Constraint Satisfaction Problem (CSP), and implemented using the Constraint Logic Programming (CLP) framework over finite domains. The constraint-based model combines flow constraints over $\{0,1\}$ variables, with resource constraints and conditional task activation models. A solving method is also proposed, which tends to be a very generic approach for solving these complex problems. It is based on branch&bound, constraint propagation and a *probing* technique. Probing is a search strategy that manages the state space solver exploration using the solution of a low-computational relaxed problem evaluation. Results are reported using a SICStus Prolog CLP(FD) implementation, with performances that suit operational needs.

The first section introduces the problem and the second one describes our constraint based approach, compared to the state of the art. Next section presents problem formulation as a CSP. Search algorithms are then described. We give a few results on realistic benchmarks and a general conclusion.

UAV Mission Planning Problem

Intrinsic UAV characteristics (i.e. maximal speed, manoeuvrability, practical altitudes) have a direct impact on operation efficiency. Figure 1 presents the *Patroller*, a UAV that has large wingspan to allow medium altitude flight, which enables performing long-range missions by minimizing energy consumption. UAV operations are not only constrained by energetic resources, but also mission time and terrain structure. Figure 2 shows a set of potential waypoints to flyby. They are defined during mission preparation, by terrain analysis, mission objectives and situation assessment. Navigation constraints are also defined by available corridors, that are provided either by navigation authorities, in civilian space, or by the Air Command Order (ACO), in military context.

Figure 1: *The Patroller UAV can detect targets at long range. With such UAV, the operator requests observation at preparation time or during mission execution.*

Informal Description

A navigation plan consists in a subset of waypoints, totally ordered, estimated flyby dates and some observations to perform. Choosing the final mission plan depends on multiple criteria (duration, available energy, exposure, objectives, initial and recovery points). Maximizing mission objectives, for instance the number of observations performed during the mission, is the primary cost objective of planning automation. The overall mission duration, exposure and onboard energy may also be maintained as low as possible. To decide a mission plan, the user must deal with the following elements:

- Initial UAV conditions: initial positions and remaining energy.
- Terrain structure: defined as a set of navigation waypoints, connected by available paths. Each waypoint has a geographical reference, and a distance metric is defined to compute the value between any couple of waypoints.
- Mission Objectives: the final recovery point, and any waypoint to which a sensing observation has been associated (requested by some user).
- On-board resource consumption: resources can be consumed due to UAV mobility (from a waypoint to another one) and/or observation action.
- Exposure: in some defense missions, the UAV exposure to threats shall be mastered.

In general, the plan is defined at mission preparation time, but it can be redefined on-line due to the situation evolution:

- Situation changes: new threats might appear.
- Mission objective: sensing and observations actions can be updated. The recovery points can be updated during mission execution.
- UAV state: energy consumption is not what was expected (for instance due to wind conditions).

The remote operator receives in real time all the critical data that may require a replanning event. To be able to keep operational efficiency, it is fundamental to have fast solving algorithms that can address realistic missions plans and be able to deal with all the mission constraints.

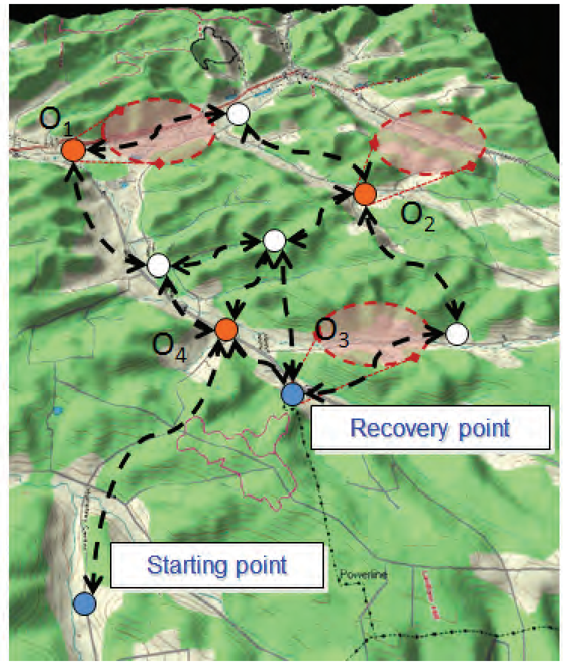


Figure 2: *Navigation plan and observation requests from users. The UAV must maximize the number of requested observations (specified by field of views, represented in red), under time and energetic constraints.*

Example

In figure 2, the UAV takes off from the initial position (blue circle) and must perform a maximal set of observations among $\{O_1, O_2, O_3, O_4\}$. Each observation consumes energy and time, as for navigation between two points. In case of a defense mission, it also exposes the UAV to opponent visibility. The UAV is recovered after a last potential observation in O_3 (blue circle). To satisfy energy and UAV exposure, the user decides to only plan for observation actions $\{O_1, O_2, O_3\}$ and discards observation O_4. White circles are potential flyby navigation points.

Complexity

Some simplified versions of the problem are equivalent to known hard problems. If the set of observations is fixed, then the problem can be specialized as a *Travelling Salesman Problem* (TSP) with multiple distance constraints. Maximizing the set of observation actions can also be relaxed as a *knapsack* problem by formulating a path weight for each action. In both cases, these problems are known to be NP-hard, and solution verification can be performed in polynomial time. Solution can be evaluated by a simple check of the navigation plan, verifying that each action is correctly scheduled and metrics are correctly instantiated.

Therefore some problem instances are NP-hard on worst cases, although polynomial families may certainly be exhibited.

A Constraint Programming Approach

State of the Art

Several approaches can deal with such problems, ranging from classical planning to very specific algorithms.

- Domain-independent planning(Fox and Long 2000), using Planning Domain Description Language (PDDL) formalisms (Fox and Long 2003). This language can model several complex actions.
- Dedicated planners have been developed for UAV, UGV and vehicle planning: Ix-TeT (Laborie and Ghallab 1995), Heuristic Scheduling Testbed System (HSTS) (Muscettola 1993), Reactive Model-based Programming Language (RMPL) (Abramson, Kim, and Williams 2001)...
- Planning frameworks like Hierarchical Task Network (HTN)(Goldman et al. 2002; Meuleau et al. 2009) have been developed to tackle specific operational domains.

All these framework need to be complemented with CSP formulation in order to tackle resource and temporal constraints.

Linear Programming (LP) techniques can also be envisaged. However, if dealing with non-linearity or discrete variables, constraints cannot be easily reformulated into linear ones without a massive increase of the variable set.

Many heuristic search methods are based on the well-known A* (Hart, Nilsson, and Raphael 1968) and also commonly used in vehicle planning. Several families have been derived, such as Anytime A* (Hansen and Zhou 2007), or other variants, adapted to dynamic environments. They can be divided into two categories: incremental heuristic searches(Koenig, Sun, and Yeoh 2009) and real-time heuristic searches (Botea, Mller, and Schaeffer 2004). For example, an experiment has been performed for emergency landing (Meuleau et al. 2011), that uses A* algorithm, integrated into aircraft avionics. These algorithms can be efficient but are limited to simple cost objectives or basic constraint formulations.

Advanced search techniques can also solve vehicle routing problems, using Operation Research(Gondran and Minoux 1995) (OR) or local search(Aarts and Lenstra 1997) techniques. Simulated Annealing (Cerny 1985), Genetic Algorithms (Goldberg 1989), Ant Colony Optimization (ACO) (Dorigo and Gambardella 1997), and more generally metaheuristics are also good candidates. These techniques do not necessarily provide optimality nor completeness, but scale very well to large problems. However, it may require strong effort to implement complex mission constraints.

This work follows previous research in vehicle routing using constraint logic programming (CP) in Prolog and hybrid techniques (Lucas et al. 2010; Lucas and Guettier 2010). In the field of logic programming, new paradigms have emerged such as *Answer Set Programming* leading to *A-Prolog* or, more recently, *CR-Prolog* languages (with their dedicated solvers). However, their declarative extensions are not significant in the context of this work.

Using Constraint Logic Programming

Operational users are not only interested in performance, feasibility or scalability, but at first in mission efficiency. In this paper, we consider maximizing mission observations while taking into account time, energetic or exposure constraints.

To satisfy user needs, the problem must be addressed globally, which requires composition of different mathematical constraints. This can be done using a declarative logical approach, constraint predicates and classical operators (Hentenryck, Saraswat, and Deville 1998). Due to the introduction of complex navigation constraints related to actions description, other approaches cannot be efficiently used. Search techniques can be complex to design (in the case of A*) or models difficult to express (in the case of LP). Furthermore, as shown in previous works, the problem can be extended in several ways by combining different formulations. Search algorithms and heuristics must be developed or adapted without reconsidering the whole model.

This can be achieved using CLP expressiveness, under a model-based development approach. CLP is a competitive approach to solve either constrained path or scheduling problems. In CLP, CSP follows a declarative formulation and is decoupled from search algorithms, so that both of them can be worked out independently. Designers can perform a late binding between CSP formulation and search algorithm. This way, different search techniques can be evaluated over multiple problem formulations. The development method also enables an easier management of tool evolutions by the designers.

CSP formulation and search algorithms are implemented with the CLP(FD) domain of SICStus Prolog library. It uses the state-of-the-art in discrete constrained optimization techniques: Arc Consistency-5 (AC-5) for constraint propagation, using CLP(FD) predicates. With AC-5, variable domains get reduced until a fixed point is reached by constraint propagation.

Most of constraint programming frameworks have different tools to design hybrid search techniques, by integrating Metaheuristics, OR and LP algorithms (Ajili and Wallace 2004). An hybrid approach is proposed to solve the mission planning problem by exploiting Dijkstra algorithm and to elaborate a meta-metric over search exploration structure. This approach, known as probing, relies on problem relaxation to deduce the search tree structure. This can be done either statically or dynamically. The CLP framework also enables concurrent solving over problem variables.

The global search technique under consideration guarantees completeness, solution optimality and proof of optimality. It relies on three main algorithmic components:

- Variable filtering with correct values, using specific labelling predicates to instantiate problem domain variables. AC being incomplete, value filtering guarantees the search completeness.

- Tree search with standard backtracking when variable instantiation fails.
- Branch and Bound (B&B) for cost optimization, using minimize predicate.

Designing a good search technique consists in finding the right variables ordering and value filtering, accelerated by domain or generic heuristics. In general, these search techniques are implemented with a conjunction of multiple specific labelling predicates.

Problem Formalization

A navigation plan is represented using a directed graph $G(X, U)$ where:

- the set U of edges represents possible paths;
- the set V of vertices are navigation points. In the remaining of the paper, a vertex is denoted x, while an edge can be denoted either u or (x, x').

Navigation Plan

A navigation plan is defined by the set of positive flows over edges. The set of variables $\varphi_u \in \{0, 1\}$ models a possible path from $start \in X$ to $end \in X$, where an edge u belongs to the navigation plan if and only if a decision variable $\varphi_u = 1$. The resulting navigation plan, can be represented as $\Phi = \{u|\ u \in U,\ \varphi_u = 1\}$.

Consistency Constraints

From an initial position to a final one, path consistency is enforced by flow conservation equations, where $\omega^+(x) \subset U$ and $\omega^-(x) \subset U$ are outgoing and incoming edges from vertex x, respectively.

$$\sum_{u\ \in\ \omega^+(start)} \varphi_u = 1, \quad \sum_{u\ \in\ \omega^-(end)} \varphi_u = 1, \tag{1}$$

$$\sum_{u\ \in\ \omega^+(x)} \varphi_u = \sum_{u\ \in\ \omega^-(x)} \varphi_u \leq 1 \tag{2}$$

Since flow variables are $\{0, 1\}$, equation (2) ensures path connectivity and uniqueness while equation (1) imposes limit conditions for starting and ending the path. This constraint provides a linear chain alternating flyby waypoint and navigation along the graph edges.

Plan and metric formulations

Assuming a given date D_x associated with a position (e.g. vertex) x we use a path length formulation (3). Variable D_x is expressing the time at which the UAV reaches a position x (see example in figure 3). Assuming that variable $d_{(x',x)}$ represents the time taken to perform the manoeuvre from position x' to x (at an average edge speed) and perform potential observations on x'. This time cumulates action duration and navigation between waypoints.

We have:

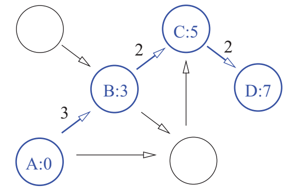


Figure 3: *Illustrating manoeuvres over a graph of navigation waypoints . This graph is a spatial representation of navigation plan. A solution, representing the UAV manoeuvres, corresponds to the set of positive values (here $\Phi = \{(A, B), (B, C), (C, D)\}$). Assuming a cumulative time metric (edge values are transit times), flyby instant is $\Delta = \{(A, 0), (B, 3), (C, 5), (D, 7)\}$.*

$$D_x = \sum_{(x',x)\ \in\ \omega^-(x)} \varphi_{(x',x)}(d_{(x',x)} + D_{x'}) \tag{3}$$

$$\forall (x, x') \in U,\ d_{(x,x')} \in \mathbb{N},\ l_{(x,x')} \leq d_{(x,x')} \leq u_{(x,x')} \tag{4}$$

Note that upper and lower speed limits (resp. $u_{(x,x')}$ and $l_{(x,x')}$) in (4) are an edge. Similar constraints are used for propagating resource consumption, as variables $< R_x, r_{(x,x')} >$, or UAV exposures, as variables $< E_x, e_{(x,x')} >$. These variables are also associated to vertices and edges. In practice E_x and R_x are normalised as a percentage of consumption.

Navigation and action realization

The set of navigation points belonging to the plan P can also be expressed as follows (5):

$$\forall x, n_x = min(1, D_x),\ P = \{x \in X,\ n_x = 1\} \tag{5}$$

where n_x states whether a position x is part of the navigation plan. If $D_x = 0$, the UAV does not flyby x. For simplicity, n_x is assimilated to a boolean variable.

A set of potential observation actions O is represented by $\|V\|$ variables $O_x \in \{0, 1\}$ and

- an observation duration constant δ_x.
- a resource consumption constant ρ_x.
- a visibility exposure constant η_x

If there is no action on vertex O_x to be performed, its default value is 0. Action activation model is defined using the following preconditions (6) and postconditions (7,8,9):

$$O_x \implies n_x \wedge E_x \geq v_x \tag{6}$$

$$\forall x, \forall x' \in \omega^+(x),$$

$$d_{(x,x')} = \delta_{(x,x')} + O_x.\delta_x \tag{7}$$

$$r_{(x,x')} = \rho_{(x,x')} + O_x.\rho_x \tag{8}$$

$$e_{(x,x')} = \eta_{(x,x')} + O_x.\eta_x \tag{9}$$

and where constant $\delta(x, x')$ is the time to navigate from point x to x'.

In equation (6), the constant v_x is an exposure threshold that is tolerated and compared to the total exposure up to waypoint x. Indeed, to satisfy the action, the UAV must be incoming to the observation location, which is the role of the term n_x. This way, each observation precondition is constrained by the level of exposure. Note that there is no precondition for energy and time. Arrival date at the recovery point is enough to constraint the whole CSP.

$D_{end} \leq D_{max}$, where D_{max} is the maximal mission duration.

Similarly, there must be remaining energy when arriving at the recovery point.

$E_{end} \geq 0$.

Other preconditions can be defined, depending on the type of action to perform (including time windows, communication, target mobility). Using our model, it is easy to overload the conjunction. However the problem can become very complex and there is not necessarily a need as long as we consider a unique UAV. Moreover, we notice that the set of preconditions is predominant compared to postconditions.

Optimization Problem

The final cost function is the total amount of observations to perform (10).

$$\Omega(V) = \sum_{x \in V} O_x \tag{10}$$

The sets of decision variables are Φ and O such that the CSP can then be formulated in Prolog as follows (1):

Algorithm 1 *Optimizing observations*

Instantiate variable sets Φ, O

- **Satisfying** navigation constraints (1), (2), (5),
- **Satisfying** metric constraints (3), (4) and
- **for all** actions $\{O_1, \dots O_x, \dots O_n\}$
  - **satisfying** preconditions (6)
  - **satisfying** postconditions (7), (8) and (9)

Maximizing $\Omega(V)$

Search Algorithms

Overview

The goal of hybridizing global solving with stochastic approaches is to save the number of backtracks by quickly focusing the search towards good solutions. It consists in designing the tree search according to the problem structure, revealed by the probe. The idea is to use the prober to order problem variables, as a pre-processing. Instead of dynamic probing with tentative values such as in (Sakkout and Wallace 2000), this search strategy uses a static prober which orders problem variables to explore according to the relaxed solution properties. Then, the solving follows a standard CP search strategy, combining variable filtering, AC-5 and B&B. As shown in figure 4, the probing technique proceeds in three steps (the three blocks on the left). The first one is to

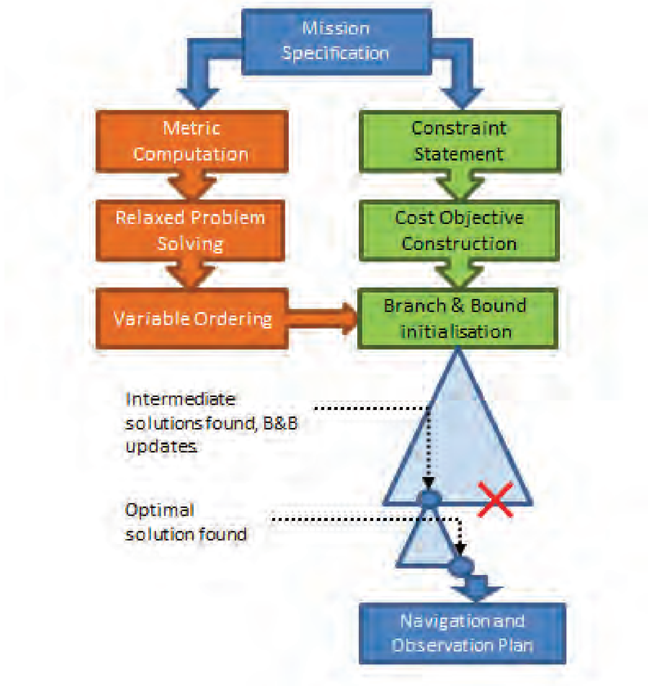


Figure 4: *Diagram of the complete solver using probing techniques.*

establish the solution to the relaxed problem. As a reference, we can for example compute the shortest path between starting and ending vertices, abstracting away mandatory waypoints. The next step is to establish a minimal distance between any problem variable and the solution to the relaxed problem. This step can be formally described as follows. Let $X_s \subset X$ be the set of vertices that belong to the relaxed solution. The distance is given by the following evaluation:

$$\forall x \in X, \delta(x) = \min_{x' \in X_s} ||(x, x')|| \tag{11}$$

where $||.||$ is a specific distance metric (in our case, the number of vertices between x and x'). The last step uses the resulting partial order to sort problem variables in ascending order. At global solving level the relaxed solution is useless, but problem variables are explored following this order.

Properties

Two interesting probe properties can be highlighted:

- *probe complexity*: since computation of minimum distance between a vertex and any node is polynomial thanks to Dijkstra or Bellman-Ford algorithms, the resulting probe construction complexity is still polynomial in worst cases. The complexity of quicksort can in practice be neglected (see below for further details).
- *probe completeness*: since the probe does not remove any value from variable domains and the set of problem vari-

ables remains unchanged, the probe still guarantees global solving completeness.

Complexity analysis: let γ be the cardinality of V_s and n the one of V. The complexity of probe construction is:

- in worst case performance: $O(n^2)$;
- in average case performance: $O(\gamma.n.\log(n))$.

Sketch of the proof: the probing method first determines the minimal distance between all vertices $X' \in X'$ where $X' = X \setminus X_s$ and any vertex $x_s \in X_s$. A Dijkstra algorithm runs over a vertex x_s allows to compute the distance to any point of X' with $O(n.\log(n))$ worst case complexity where n is the number of nodes in X. This has to be run over each vertex of X_s and a comparison with previous computed values must be done for every vertex x', to keep the lowest one. Thus, the resulting complexity is $O(\gamma.n.\log(n))$. Variables must finally be sorted with a quicksort-like algorithm. The worst case complexity of this sort is $O(n^2)$ but is generally computed in $O(n.\log(n))$ (average case performance). Hence, the worst case complexity of the probing method is $O(n^2)$, but in practice behaves in $\max\{O(\gamma.n.\log(n)), O(n.\log(n))\} = O(\gamma.n.\log(n))$.

Pseudocode

Algorithm 2 synthesizes probe construction mechanisms. Firstly, a vector L_d of size n (n being the number of nodes in X) is created and initialized with infinite values. At the end of the execution, it will contain a value associated to each vertex, corresponding to the minimal distance between this vertex and the solution to the relaxed problem. To do so, a Dijkstra algorithm is run over each node of the solution. During a run, distances are evaluated and replaced in L_d if lower than the existing value (in the pseudo code, comparison are made at the end of a run for easier explanation). Once minimal distances are all computed, they are used to rank the set of vertices X in ascending order (to be used by the complete solver).

Algorithm 2 *Probe construction*

Initialize a vector L_d of distances (with infinite values)
Get P the best solution of the relaxed problem
for each node x_i of P **do**
    $L'_d \Leftarrow$ Run Dijkstra algorithm from x_i
    $L_d = \min(L_d, L'_d)$ (value by value)
end for
Sort X using L_d order
return the newly-ordered X list

Preliminary Results

Experiments on four benchmarks are presented. They are representative of modern peace keeping missions or disaster relief. Missions must be executed in less than 30 minutes. Areas range from 5x5 kms to 20x20 kms.

1. Recon villages: Observing different villages after a water flooding event.
2. Reinforce UN: Bring support to a United Nations mission by observing an unsecure town.
3. Sites inspections: Observing different parts of a town during inspection of suspect sites.
4. Secure humanitarian area: Observing different threats before securing refugees, over a large area.

For each benchmark, four experimentations are run. Two sets of runs are performed, one with the simple branch and bound, the other one with the probing method. For each set, two different constraints are preconditions to observation actions (constraint 6):

- Energy constraint, precondition is simplified to

$$O_x \implies n_x$$

- Exposure constraint, precondition is as (6):

In practice, the exposure threshold is set between 10 and 20 percent for each observation action. This overconstrains the problem, allowing us to observe performance differences.

Experiments			**Results**		
Problem	**Algorithm**	**Actions**	**Time (ms) for opt.**	**proof**	**Best Value (#actions)**
1. Recon villages (22 nodes, 74 edges, 702 vars, 2251 constraints)					
Energy	Probing	3	250	560	1
Exposure	Probing	3	234	609	1
Energy	Simple	3	274	1092	1
Exposure	Simple	3	358	982	1
2. Reinforce UN (23 nodes, 76 edges, 723 vars, 2312 constraints)					
Energy	Probing	3	93	93	3
Exposure	Probing	3	296	702	2
Energy	Simple	3	1045	1061	3
Exposure	Simple	3	5460	11139	2
3. Site inspection (22 nodes, 68 edges, 654 vars, 2081 constraints)					
Energy	Probing	4	109	249	3
Exposure	Probing	4	187	312	3
Energy	Simple	4	717	1575	3
Exposure	Simple	4	1451	2261	3
4. Secure area (33 nodes 113 edges, 1069 vars, 3447 constraints)					
Energy	Probing	3	2371	4977	2
Exposure	Probing	3	7566	10234	2
Energy	Simple	3	8237	15944	2
Exposure	Simple	3	22074	29375	2

Figure 5: *Results overview on benchmark scenarios, maximizing the number of action to perform*

Table 5 reports the time to find the optimal solution, as well as for proving optimality. It also shows the maximal number of observations that can be executed. Simple problems can be solved fairly quickly, but the last benchmark is more computation demanding, which is certainly due to a large area to cover. On the second benchmark, exposure constraints prevent from performing all observations. Again for all the problem instances, the probing method improves drastically the solver performances, which confirm former researches (Lucas et al. 2010) and (F. and C. 2012). By comparing with energetic constraints, exposure preconditions makes the problem really harder to solve.

Conclusion

This paper shows the development of the mission planning framework, that can be used either for C2 systems or for unmanned systems. Introducing actions with complex preconditions and postconditions increases the practical complexity of problem instances. In particular, with the existing design, the solving approach does not scale huge numbers of observation or large graph structures. Nevertheless, as expected by previous results, the probing approach improves drastically solving performances. Using the modeling approach, the formulation of action preconditions and postconditions can be extended in several ways. Further works will focus on scalability as well as different forms of probing, relying on action definition in the relaxation process.

References

Aarts, E., and Lenstra, J. 1997. *Local Search in Combinatorial Optimization*. Princeton University Press.

Abramson, M.; Kim, P.; and Williams, B. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Ajili, F., and Wallace, M. 2004. Constraint and integer programming : toward a unified methodology. *Operations research/computer science interfaces series*.

Botea, A.; Mller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1(1).

Cerny, V. 1985. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45:41–51.

Dorigo, M., and Gambardella, L. 1997. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1(1):53–66.

F., L., and C., G. 2012. Hybrid solving technique for vehicle planning. In *Proceedings of Military Communication Conference (MILCOM)*.

Fox, M., and Long, D. 2000. Automatic synthesis and use of generic types in planning. In *Proceedings of the Artificial Intelligence Planning System*, 196–205. AAAI Press.

Fox, M., and Long, D. 2003. Pddl 2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Goldman, R.; Haigh, K.; Musliner, D.; and Pelican, M. 2002. Macbeth: A multi-agent constraint-based planner. In *Proceedings of the 21st Digital Avionics Systems Conference*, volume 2, 7E3:1–8.

Gondran, M., and Minoux, M. 1995. *Graphes et Algorithmes*. Editions Eyrolles.

Hansen, E., and Zhou, R. 2007. Anytime heuristic search. *Journal of Artificial Intelligence Research* 28:267–297.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics* 4(2):100–107.

Hentenryck, P. V.; Saraswat, V. A.; and Deville, Y. 1998. Design, implementation, and evaluation of the constraint language cc(fd). *J. Log. Program.* 37(1-3):139–164.

Koenig, S.; Sun, X.; and Yeoh, W. 2009. Dynamic fringe-saving a*. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2, 891–898.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Lucas, F., and Guettier, C. 2010. Automatic vehicle navigation with bandwidth constraints. In *In Proceedings of MILCOM 2010*.

Lucas, F.; Guettier, C.; Siarry, P.; de La Fortelle, A.; and Milcent, A.-M. 2010. Constrained navigation with mandatory waypoints in uncertain environment. *International Journal of Information Sciences and Computer Engineering (IJISCE)* 1:75–85.

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2009. Emergency landing planning for damaged aircraft. In *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference*.

Meuleau, N.; Neukom, C.; Plaunt, C.; Smith, D.; and Smithy, T. 2011. The emergency landing planner experiment. In *21st International Conference on Automated Planning and Scheduling*.

Muscettola, N. 1993. Hsts: Integrating planning and scheduling. In *Technical Report CMU-RI-TR-93-05*.

Sakkout, H. E., and Wallace, M. 2000. Probe backtrack search for minimal perturbations in dynamic scheduling. *Constraints Journal* 5(4):359–388.

A Metaheuristic Technique for Energy-Efficiency in Job-Shop Scheduling

Joan Escamilla[1], **Miguel A. Salido**[1], **Adriana Giret**[2] and **Federico Barber**[1]
[1]Instituto de Automática e Informática Industrial
[2]Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

Many real-world scheduling problems are solved to obtain optimal solutions in term of processing time, cost and quality as optimization objectives. Currently, energy-efficiency is also taking into consideration in these problems. However, this problem is NP-Hard, so many search techniques are not able to obtain a solution in a reasonable time. In this paper, a genetic algorithm is developed to solve an extended version of the Job-shop Scheduling Problem in which machines can consume different amounts of energy to process tasks at different rates. This problem represents an extension of the classical job-shop scheduling problem, where each operation has to be executed by one machine and this machine can work at different speeds. The evaluation section shows that the powerful commercial tools for solving scheduling problems was not able to solve large instances in a reasonable time, meanwhile our genetic algorithm was able to solve all instances with a good solution quality.

Introduction

Nowadays, the main objective of many companies and organizations is to improve profitability and competitiveness. These improvements can be obtained with a good optimization of resources allocation. But in the last years many companies are not only facing complex and diverse economic trends of shorter product life cycles, quick changing science and technology, increasing customer demand diversity, and production activities globalization, but also enormous and heavy environmental challenges of global climate change (e.g. greenhouse effect), rapid exhaustion of various nonrenewable resources (e.g. gas, oil, coal), and decreasing biodiversity.

Scheduling problems are widely discussed in the literature and two main approaches can be distinguished (Billaut, Moukrim, and Sanlaville 2008):

- Classical deterministic methods, which consider that the data are deterministic and that the machine environment is relatively simple. Some traditional constraints are taken into account (precedence constraints, release dates, due dates, preemption, etc.). The criterion to optimize is often standard (makespan). A number of methods have been proposed (exact methods, greedy algorithms, approximate methods, etc.), depending on the difficulty of a particular problem. These kinds of studies are the most common in the literature devoted to scheduling problems.
- On-line methods. Sometimes, the algorithm does not have access to all the data from the outset, the data become available step by step, or ”on-line”. Different models may be considered here. In some studies, the tasks that we have to schedule are listed, and appear one by one. The aim is to assign them to a resource and to specify a start time for them. In other studies, the duration of the tasks is not known in advance.

In both cases, the job-shop scheduling problem (JSP) has been studied. It represents a particular case of scheduling problems where there are some specific resources or machines which have to be used to carry out some tasks. Many real life problems can be modeled as a job-shop scheduling problem and can be applied in some variety of areas, such as production scheduling in the industry, departure and arrival times of logistic problems, the delivery times of orders in a company, etc. Most of the solving techniques try to find the optimality of the problem for minimizing the makespan, tardiness, flow-time, etc.

Nowadays, the main objective of many companies and organizations is to improve profitability and competitiveness. These improvements can be obtained with a good optimization of resources allocation. But in the last years many companies are not only facing complex and diverse economic trends of shorter product life cycles, quick changing science and technology, increasing customer demand diversity, and production activities globalization, but also enormous and heavy environmental challenges of global climate change (e.g. greenhouse effect) (Mestl et al. 2005), rapid exhaustion of various non-renewable resources (e.g. gas, oil, coal) (Yusoff 2006), and decreasing biodiversity.

Recently some works have focused on minimizing the energy consumption in scheduling problems (Mouzon and Yildirim 2008)(Dai et al. 2013), mainly from the Operations Research Community (Bruzzone et al. 2012), (Mouzon, Yildirim, and Twomey 2007) and (Li, Yan, and Xing 2013).

In job-shop scheduling problem with voltage scaling, machines can consume different amount of energy to process tasks at different speeds (Malakooti et al. 2013). By changing the voltage level, the frequency at which a processor executes a task is adjusted, and processing speed changes as a result. We focus our attention in a job-shop scheduling problem with different speed machine (JSMS). It represents an extension of the classical job-shop scheduling problem ($J||C_{max}$ according to classification scheme proposed in (Blazewicz et al. 1986)), where each operation must be executed in a machine at a determined speed with a determined energy consumption (by a classical deterministic method).

Problem Description

Formally the job-shop scheduling problem with different speed machine (JSMS) can be defined as follows. There exist a set of n jobs $\{J_1, \ldots, J_n\}$ and a set of m resources or machines $\{R_1, \ldots, R_m\}$.

Each job J_i consists of a sequence of T tasks $(\theta_{i1}, \ldots, \theta_{iT_i})$. Each task θ_{il} has a single machine requirement $R_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. The main difference with the traditional job shop scheduling problem is related to each machine can work at different speeds. Thus, each task θ_{il} is linked up to different durations ($p_{\theta_{il1}}$, $p_{\theta_{il2}}$,...,$p_{\theta_{ilp}}$) and their corresponding energy consumption $e_{\theta_{il1}}$, $e_{\theta_{il2}}$,...,$e_{\theta_{ilp}}$ used by the corresponding machine. Figure 1 shows the relationship between energy consumption and processing time of each task in a machine. This curve can be approximated by the equation 1:

$$T\theta_{il} = 1 + \frac{1}{ln(1 + E\theta_{il}^3)} - \frac{1}{ln(1 + E\theta_{il})} \qquad (1)$$

where T is the processing time and E in the energy consumption of a task in a machine. It can be observed that if the speed of a machine is high, the energy consumption increases, but the processing time of the task decreases, meanwhile if the speed is low, the energy consumption decreases and the processing time increases. For simplicity and without loss of generality, we consider three different energy consumptions and processing times for each task. Thus, we can apply the former formula to the benchmarks presented in the literature in order to obtain the optimal and energy aware schedule. The original processing time of each task (value 1) is assigned to an energy consumption of 1 (regular speed). If the processing time of a task is increased a 70%, the energy consumption is reduced a 20% (low speed). However, if the processing time of a task is reduced 30%, the energy consumption is increased 20% (high speed). Depending on the specific problem, this curve can vary, and therefore the proportion between processing time and energy consumption can significantly change. In this paper, the processing times are randomly selected by applying the expressions (6) and (7).

A feasible schedule is a complete assignment of starting times to tasks that satisfies the following constraints: (i) the tasks of each job are sequentially scheduled, (ii) each machine can process at most one task at any time, (iii) no preemption is allowed. The objective is finding a feasible schedule that minimizes the completion time of all the tasks and the energy used.

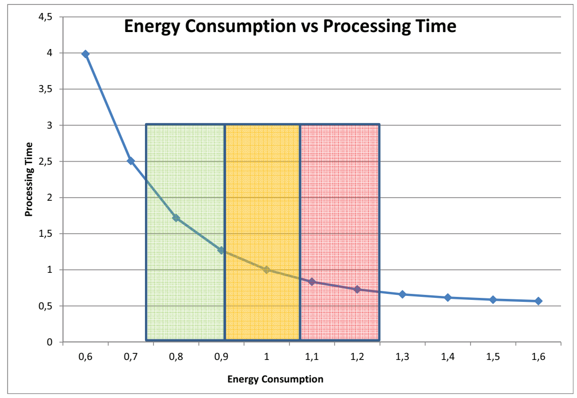


Figure 1: Relationship between energy consumption and processing time

Energy Efficiency

Nowadays manufacturing enterprisers are not only facing complex and diverse economic trends of shorter product life cycles, quick changing science and technology, increasing customer demand diversity, and production activities globalization, but also enormous and heavy environmental challenges of global climate change (e.g. greenhouse effect), rapid exhaustion of various non-renewable resources (e.g. gas, oil, coal), and decreasing biodiversity. Statistical data in 2009 shows the Germany industrial sector was responsible for approximately 47% of the total national electricity consumption, and the corresponding amount of CO2 emissions generated by this electricity summed up to 18%-20% (BMWi 2009). Thus, manufacturing companies are responsible for the environmental outcome and are forced to have manufacturing systems that demonstrate major potential to reduce environmental impacts (Duflou et al. 2012). Recently, there has been growing interest in the development of energy savings due to a sequence of serious environmental impacts and rising energy costs. Research on minimizing the energy consumption of manufacturing systems has focused on three perspectives: the machine level, the product level, and the manufacturing system level. From the machine-level perspective, developing and designing more energy-efficient machines and equipment to reduce the power and energy demands of machine components is an important strategic target for manufacturing companies (Li et al. 2011)(Neugebauer et al. 2011). Unfortunately, previous studies show that the share of energy demand for removal of metal material compared to the share of energy needed to support various functions of manufacturing systems is quite small (less than 30%) of total energy consumption (Dahmus and Gutowski 2004). From the product-level perspective, modeling embodied product energy framework based on a product design viewpoint for energy reduction approach is beneficial to support the improvements of product design and operational decisions (Seow and Rahimifard 2011)(Weinert, Chiotellis, and Seliger 2011). It requires strong commercial simulation software to facilitate the analysis and evaluation of the embodied product energy. The results cannot be applied easily in most manufacturing companies, especially in small- and medium-size enterprises due to the enormous financial investments required. From the manufacturing system-level perspective, thanks to decision models that support energy savings, it is feasible to achieve a significant reduction in energy consumption in manufacturing applications. In the specialized literature about production scheduling, the key production objectives for production decision models, such as cost, time and quality have been widely discussed. However, decreasing energy consumption in manufacturing systems through production scheduling has been rather limited. One of the most well-known research works is the work of Mouzon et al.(Mouzon, Yildirim, and Twomey 2007), who developed several algorithms and a multiple-objective mathematical programming model to investigate the problem of scheduling jobs on a single CNC machine in order to reduce energy consumption and total completion time. They pointed out that there was a significant amount of energy savings when non-bottleneck machines were turned off until needed; the relevant share of savings in total energy consumption could add up to 80%. They also reported that the inter-arrivals would be forecasted and therefore more energy-efficient dispatching rules could be adopted for scheduling. In further research, Mouzon and Yildirim (Mouzon and Yildirim 2008) proposed a greedy randomized adaptive search algorithm to solve a multi-objective optimization schedule that minimized the total energy consumption and the total tardiness on a machine. Fang et al. (Fang et al. 2011) provided a new mixed-integer linear programming model for scheduling a classical flow shop that combined the peak total power consumption and the associated carbon footprint with the makespan. Yan et al. (Yan and Li 2013) presented a multi-objective optimization approach based on weighted grey relational analysis and response surface methodology. Bruzzone et al. (Bruzzone et al. 2012) presented an energy-aware scheduling algorithm based on a mixed-integer programming formulation to realize energy savings for a given flexible flow shop that was required to keep fixed original job assignment and sequencing. Although the majority of the research on production scheduling has not considered energy-saving strategies completely, the efforts mentioned above provide a starting point for exploring an energy-aware schedule optimization from the viewpoint of energy consumption.

Modeling and Solving a JSMS as a Genetic Algorithm

The more natural way to solve a traditional Jop-Shop Scheduling Problem is to represent all variables and constraints related to jobs, tasks and machines (Garrido et al. 2000) (Huang and Liao 2008) in order to be solved by a sound and completed search technique. The traditional objectives are to obtain solutions that minimize the typical objective functions presented in the literature (makespan, tardiness, completion time, etc). It is well-known that this problem is NP-hard, so that optimal solutions can only be achieved for small instances. However, few techniques have been developed to minimize energy consumptions in these problems. Only in the last few years, some researchers have focused their attention in the machine level to solve the scheduling problem by minimizing the energy consumption (Dai et al. 2013). This requirement increases the complexity of the problem so it is not possible to obtain optimal solutions. This problem called job-shop scheduling problem with different speed machine (JSMS) must be solved by heuristic and metaheuristic techniques in order to obtain optimized solutions, mainly in large instances. To this end, in this paper we develop a genetic algorithm to solve the JSMS. In the evaluations section, it can be observed that powerful commercial techniques were not able to solve large instances in a reasonable time; meanwhile small instances are solved by both techniques with similar solution quality.

In this section we propose a Genetic Algorithm (GA) to solve the job-shop scheduling problem with machines at different speeds (JSMS). Genetic Algorithms (GA) are adaptive methods which may be used to solve optimization problems (Beasley, Martin, and Bull 1993). They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principle of natural selection, i.e. survival of the fittest. At each generation, every new individual (chromosome) corresponds to a solution, that is, a schedule for the given JSMS instance. Before a GA can be run, a suitable encoding (or representation) of the problem must be devised. The essence of a GA is to encode a set of parameters (known as genes) and to join them together in order to form a string of values (chromosome). A fitness function is also required, which assigns a figure of merit to each encoded solution. The fitness of an individual depends on its chromosome and is evaluated by the fitness function. During the run, parents must be selected for reproduction and recombined to generate offspring. Parents are randomly selected from the population, using a scheme which favors fitter individuals. Having selected two parents (Procedure Select-Parents in Algorithm 1, their chromosomes are combined, typically by using crossover and mutation mechanisms to generate better offspring that means better solutions. The process is iterated until a stopping criterion is satisfied.

Algorithm 1 shows the general steps of our GA. All functions will be explained in detail to understand the behavior of the algorithm.

Chromosome encoding and decoding

In genetic algorithms, a chromosome represents a solution in the search space. The first step in constructing the GA is to define an appropriate genetic representation (coding). A good representation is crucial because it significantly affects all the subsequent steps of the GA. Many representations for the JSP have been developed.

A chromosome is a permutation of the set of operations that represents a tentative ordering to schedule them, each one being represented by its job number. Figure 2 shows an example of a job shop schedule with 3 jobs, where job 1 has 2 tasks, and both jobs 2 and 3 have 3 tasks. Each number in the chromosome cell (3,2,1,3,1,2,2,3) represents the job of the task. The first number ”3” represents the first task of the

Algorithm 1: GeneticAlgorithm (JSMS, λ)

```
Begin
if (λ ≠ 0 and λ ≠ 1) then
   Initial-Population(Population, Size,
   Speed=Random(1,3));
else
   if (λ = 0) then
      Initial-Population(Population, Size, Speed=1);
   else
      Initial-Population(Population, Size, Speed=3);
   end if
end if
Evaluate-Fitness(Population);
while (Stopping criterion is not fulfilled) do
   Select-Parents(Population, Parent1, Parent2);
   Crossover(Parent1,Parent2,Offspring);
   mutation(Offspring,Offspring');
   Evaluate-Fitness(Offspring');
   Update-Population(Population,Offspring');
end while
Report Best Schedule;
End
```

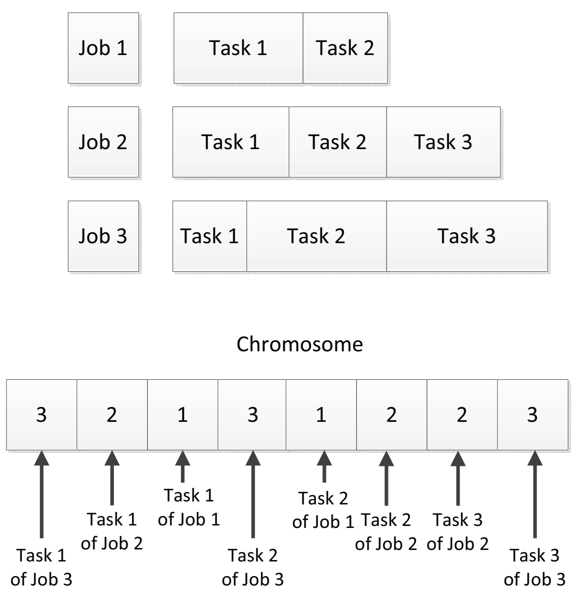


Figure 2: Codification of a chromosome in a JSP

job 3. As it was shown in (Varela, Serrano, and Sierra 2005), this encoding has a number of interesting properties for the classic job-shop scheduling problem. For instance a random assignment of values 1,2 and 3 generates a valid solution.

However, in the problem JSMS, the machine speed of each task has to be represented, therefore a new value must be added to each task in order to represent the machine speed. So a valid chromosome is $2n$ length, where n is the total number of tasks. Figure 3 shows the coding of a chro-

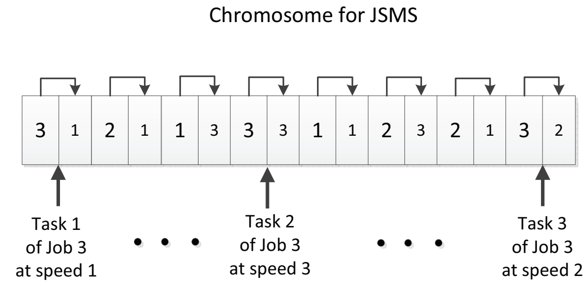


Figure 3: Codification of a chromosome in a JSMS

mosome in JSMS. Such coding increases the former coding for adding the speed at which each task is processed. Thus, the first two digits 3 and 1 represent the first task of the job 3 is processed at speed 1. Again, a random assignment of values to tasks and speed generates a valid solution. When the chromosome representation is decoded each task starts as soon as possible following the precedence and machine constraints. With the machine speed representation, the processing time of each task and energy consumption of the machine can be calculated, and therefore the makespan and total energy consumption.

Initial Population and Fitness

As we have pointed out before, each gene represents one task of the problem and the next gene represents the speed at which this task is processed. The position of each task determines its dispatch order, in this genome/solution. The initial chromosomes are obtained by a random permutation of tasks. The machine speed for each gene is also randomly generated among one of the possible speeds. Thus, the initial population is randomly generated, and it always generates feasible schedules (Procedure Initial-Population in Algorithm 1). The population size was 200 individual for Agnetis instances and 400 for Watson instances. These values were selected by testing different alternatives and selecting the best results.

JSMS can be considered a multiobjective problem due to the fact that the goal is to minimize the makespan and also to minimize the energy consumption. However both objectives are contrary so that minimize the makespan supposes to increase the speed of machines, and viceversa. Thus, in these problems no single optimal solution exists. Instead, a set of efficient solutions are identified to compose the Pareto front. Diverse techniques have been developed to solve multiple objective optimization problems. One of the most well-known methods for solving multiple objective optimization problems is the Normalized Weighted Additive Utility Function (NWAUF), where multiple objectives are normalized and added to form a utility function. NWAUF has been implemented in wide range of multiple objective optimization problems due to its simplicity and natural ability to identify efficient solutions. Let f_{ij} be the ith objective function value of alternative j. Then, the NWAUF for alternative j with k objectives is defined as:

$$U_j = w_1 f'_{1j} + w_2 f'_{2j} + ... + w_k f'_{kj} \quad (2)$$

where $w_1, w_2, ..., w_k$ are weights of importance and $f'_{1j}, f'_{2j}, ..., f'_{kj}$ are normalized values of $f_{1j}, f_{2j}, ..., f_{kj}$. By normalizing different objectives, all objectives are evaluated in the same scale. Weights show decision maker's preference for each objective where, $\sum_{i=1}^{k} w_i = 1$ and $0 \leq w_i \leq 1$ for $i = 1, ..., k$. Using this utility function, the multiple objective optimizations can now be solved as a single objective optimization problem.

The definition of fitness function is just the reciprocal of the objective function value. The objective is to find a solution that minimizes the multi-objective makespan and energy consumption. Following NWAUF rules, our fitness function $F(i)$ (3) is a convex combination between the normalized values of makespan and energy consumption of solution i.

$$F(i) = \lambda * NormMakespan(i) + (1-\lambda) * NormEnergy(i) \quad (3)$$

$$NormMakespan(i) = \frac{Makespan(i)}{MaxMakespan} \quad (4)$$

$$NormEnergy(i) = \frac{SumEnergy(i)}{MaxEnergy} \quad (5)$$

where $\lambda \in [0, 1]$. $NormMakespan$ (4) is the makespan divided by the maximum makespan value in a genetic algorithm execution when the λ value is equal to 0 ($MaxMakespan$). $MaxMakespan$ values can be found in the benchmark section of our webpage[1]. $NormEnergy$ (5) is calculated by summing the energy used in the execution of all the tasks, divided the maximum energy ($MaxEnergy$). $MaxEnergy$ is the sum of the energy needed to execute all tasks at top speed.

Once the λ parameter is set for the fitness function ((3), the initial population can be generated in a specific way. Thus, for $\lambda = 0$, the objective function is only focused to reduce the energy consumption ($F = NormEnergy$), so the initial population can be randomly generated to order the tasks, but the corresponding speeds are fixing to the lowest value (see Figure 4a). In the same way, if $\lambda = 1$, the objective function is only focused to reduce makespan ($F = NormMakespan$), so the initial population can also be randomly generated to order the tasks, but the corresponding speeds are fixing to the highest value (see Figure 4b). for $\lambda \in]0, 1[$, the speed of each task can be appropriately generated.

Crossover operator

For chromosome mating, our GA uses a (Job, Energy)-based Order Crossover. Thus, given two parents, a set of pairs (job, energy) of a random job is selected from the first parent and copied in the same position to the offspring. Afterwards, the set of pairs (job, energy) of the remaining jobs are translated from the second parent to the offspring in the same order

[1]http://gps.webs.upv.es/jobshop/

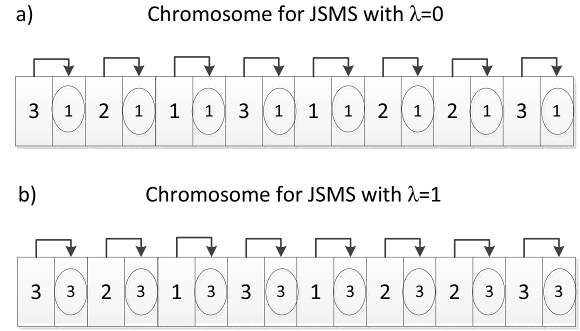


Figure 4: Initial population for $\lambda = 0$ and $\lambda = 1$

(Procedure CrossOver in Algorithm 1). We clarify how this technique works in the next example. Let us consider the following two parents (see Figure 5:

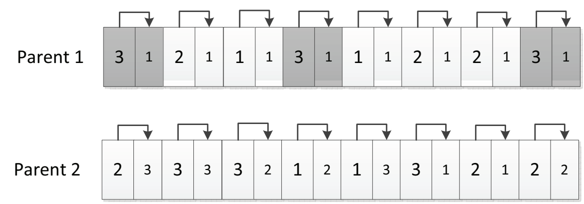


Figure 5: Two parents for the crossover operator

If the selected subset of jobs from the first parent just includes the job 3 (dark genes in Figure 5), the generated offspring is showed in Figure 6.

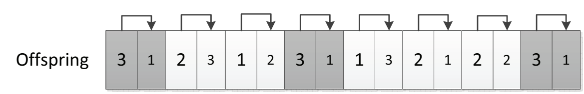


Figure 6: Offspring for the crossover operator

Hence, this technique maintains for a machine a subsequence of operations in the same order as they are in Parent 1 and the remaining ones in the same order as they appear in Parent 2. The crossover is applied in a dual way, so two parents generate two offspring (parent 1-parent 2) and (parent 2- parent 1). Parent couples are selected shuffling population and choosing each couple two by two, so all individual will be selected but only some couples will be crossed in accordance to crossover probability.

Mutation operator

The two offsprings generated with crossover operation can be also mutated in accordance to the mutation probability (Procedure Mutation in Algorithm 1). Two pair (task, energy) position of chromosome child are randomly chosen (position "a" and position "b"), where "a" must be lower than "b". Pairs between "a" and "b" are shuffled randomly, also in each gene machine speed values are randomly changed. In this step, the speeds of the machines in tasks are also randomly modified.

Finally, tournament replacement among every couple of parents and their offspring is done to obtain the next generation (Procedure Update-Population in Algorithm 1).

Evaluation

In this section we evaluate the behavior of our GA against a successful and well-known commercial solver IBM ILOG CPLEX CP Optimizer tool 12.5 (CP optimizer) (IBM 2012). It is a commercial solver embedding powerful constraint propagation techniques and a self-adapting large neighborhood search method dedicated to scheduling (Laborie 2009). This solver is expected to be very efficient for a variety of scheduling problems as it is pointed in (IBM 2007), in particular when the cumulative demand for resources exceeds their availability as it happens.

These algorithms have been evaluated with extended benchmarks of the typical job-shop scheduling problem. The extension has been focused on assigning different speeds and durations to each task (as we pointed out in section 3). The extension with machines working at different speeds have been implemented considering that each task is executed by a machine and it has different optional modes where each represents the duration of the task and an associated energy consumption (Salido et al. 2013).

To this end, we extend the benchmarks proposed in (Agnetis et al. 2011) and (Watson et al. 1999) because to the best of our knowledge there not exist benchmarks for job-shop scheduling problems that incorporate different speeds and energy consumptions. All instances are characterized by the number of jobs (j), the number of machines (m), the maximum number of tasks by job (v_{max}) and the range of processing times (p). In Agnetis instances, j is set to 3 and these can be represented as $m\_v_{max}\_p$, and the number of operators was not considered in this study, so we fixed it to the number of machines.

The authors consider two types of instances: small and large Agnetis instances:

- j = 3; m = 3, 5, 7; v_{max} = 5, 7, 10; p = [1, 10], [1, 50], [1, 100]
- j = 3; m = 3; v_{max} = 20, 25, 30; p = [1, 50], [1, 100], [1, 200]

In Watson instances follow the same characterized, but in this case the variable that changes is the number of jobs (j):

- j = 50, 100, 200; m = 20; v_{max} = 20; p = [1, 100]

For each type of instance we work with 10 instances so the results presented in this section are always the averages value. We have modeled the instances to be solved by the CP Optimizer. We have also extended the original instances to add three different energy consumptions (e_1, e_2, e_3) to each task according to three processing times (pt_1, pt_2, pt_3), where pt_1 is equal to the value of processing time in the original instances. pt_2 and pt_3 were calculated following the expressions (6) and (7), respectively (Salido et al. 2013). These instances can be found in the web page[2].

$$pt_2 = Max(maxdur*0.1+pt_1, Rand(1.25*pt_1, 2.25*pt_1)) \quad (6)$$

$$pt_3 = Max(maxdur*0.1+pt_2, Rand(1.25*pt_2, 2.25*pt_2)) \quad (7)$$

The value $maxdur$ represents the maximum duration of a task for the corresponding instance and the expression $rand$ represents a random value between both expressions. Similar expressions were developed to calculate the energy consumption represented in expressions (8, 9, 10).

$$e_1 = Rand(pt_1, 3*pt_1)) \quad (8)$$

$$e_2 = Max(1, Min(e_1 - maxdur*0.1, Rand(0.25*e_1, 0.75*e_1)) \quad (9)$$

$$e_3 = Max(1, Min(e_2 - maxdur*0.1, Rand(0.25*e_2, 0.75*e_2)) \quad (10)$$

Following these expressions the processing times of pt_1, pt_2, pt_3 increase as the energy consumption e_1, e_2, e_3 decrease (see section 3). For example, give an instance with 5 tasks per job, three triplets are represented for each task: the id of the task, the energy used and the processing time ($< id, e, pt >$):

$< id, e_3, pt_3 >, < id, e_2, pt_2 >, < id, e_1, pt_1 >$
$< 1, 14, 14 >, < 1, 16, 10 >, < 1, 19, 7 >,$
...
$< 15, 3, 6 >, < 15, 5, 4 >, < 15, 6, 3 >,$

Comparative study between CP Optimizer and GA

CP and GA techniques try to minimize the multiobjective makespan and energy consumption. The weight of each objective can be changed by λ parameter, following the expression (3). To compare both techniques, they have been executed in a Intel Core2 Quad CPU Q9550, 2.83GHz and 4Gb Ram computer with Ubuntu 12.04 Operating system. The small Agnetis instances were executed during 5 seconds and the large Agnetis and Watson instances had a 100 seconds time-out. The next tables present the most important parameter to be analyzed: $\lambda \in [0, 1]$ that represents the weight given to makespan and energy consumption, MK is the makespan, En is the energy consumption, and F is the fitness function. The objective is to obtain the lowest value of F.

Table 1 shows the results for two small Agnetis instances, the smallest (3_5_10) and the largest (7_10_100) of this group. The results for the instances 3_5_10 show that the F value was equal or almost equal in both CP Optimizer and GA. Furthermore, there were small differences in all λ values for instance 7_10_100. These results show that both algorithms maintained the same behavior for small instances (the difference is in the fourth decimal).

In large Agnetis instances, the results were also similar for all the instances. Table 2 shows the results for the instances 3_25_100 as an example. The difference of F value between CP Optimizer and GA was almost in the third decimal is most cases. It must be taken into account that for $\lambda = 0.6$ or $\lambda = 0.6$, the F value of our GA was lower than in CP

	3_5_10						7_10_100					
	CP Optimizer			Genetic			CP Optimizer			Genetic		
λ	Mk	En	F	Mk	En	F	Mk	En	F	Mk	En	F
0	71.4	84.4	0.553762	65.8	84.4	0.553762	1088.4	1571.4	0.533616	1006.3	1571.4	0.533616
0.1	65.2	84.5	0.556581	65.2	84.5	0.556581	999.3	1572.6	0.540932	999.3	1572.6	0.540931
0.2	64.4	84.7	0.558703	64.4	84.7	0.558703	987.2	1576.5	0.547508	987.2	1576.5	0.547509
0.3	63.2	85.2	0.559422	63.2	85.2	0.559422	922.2	1613.3	0.550868	926.9	1610	0.551018
0.4	59.7	88.1	0.557816	59.7	88.1	0.557816	885.9	1649.2	0.550650	891	1642.9	0.550638
0.5	53.9	94.3	0.547187	54.2	93.9	0.547270	838.8	1716	0.545249	847.5	1704.8	0.545938
0.6	48.4	104.2	0.529935	48.9	103.2	0.529687	779.1	1859.3	0.535095	782.4	1845.7	0.535361
0.7	45.3	111.9	0.500419	45	112.7	0.500130	708.5	2068.4	0.511331	703.9	2099.5	0.512783
0.8	42.2	123.4	0.461368	42.2	123.5	0.461509	651.8	2346	0.475184	642.4	2418.9	0.475810
0.9	41	133.2	0.414361	41	133.7	0.414712	626	2560.7	0.428228	626	2573.3	0.428603
1	41	143.1	0.363050	41	145.3	0.363050	625.9	2664.1	0.378956	625.9	2773.4	0.378956

Table 1: Results of Small Agnetis Instances

	3_25_100					
	CP Optimizer			Genetic		
λ	Mk	En	F	Mk	En	F
0	3160	3827.1	0.533532	3096	3829	0.533805
0.1	2768.1	3827.6	0.537461	2781.7	3827.9	0.537786
0.2	2719.3	3842.5	0.540966	2764.8	3845.5	0.543204
0.3	2597.9	3904.6	0.542188	2657.3	3920.2	0.547324
0.4	2480.7	4005.8	0.540172	2495.3	4068.7	0.546693
0.5	2342	4181.6	0.533724	2317.1	4257.3	0.536410
0.6	2147	4548.6	0.520427	2118.3	4617.4	0.520423
0.7	1935.5	5075.6	0.492575	1943.7	5097.4	0.494838
0.8	1806.2	5666	0.456913	1791.4	5726.2	0.456512
0.9	1725.9	6251.3	0.408634	1732.2	6311	0.410335
1	1673.4	6732.2	0.346046	1711.8	6797.2	0.353841

Table 2: Results of Large Agnetis Instances

	CP Optimizer			Genetic		
λ	F_50	F_100	F_200	F_50	F_100	F_200
0	0.53408	0.53061	0.77288	0.610346	0.63776	0.68297
0.1	0.55899	0.56114	**No Sol.**	0.63994	0.66249	0.69391
0.2	0.58329	0.59138	**No Sol.**	0.65530	0.68276	0.71923
0.3	0.60836	0.62144	**No Sol.**	0.66789	0.69951	0.73313
0.4	0.63038	0.65117	**No Sol.**	0.67253	0.70370	0.73440
0.5	0.65334	0.68317	**No Sol.**	0.66715	0.69871	0.73288
0.6	0.66936	0.69793	**No Sol.**	0.65396	0.69041	0.72827
0.7	0.65937	0.69234	**No Sol.**	0.63458	0.67832	0.72394
0.8	0.64346	0.68051	**No Sol.**	0.61773	0.66570	0.71953
0.9	0.63197	0.66429	**No Sol.**	0.59558	0.65161	0.7136
1	0.52675	0.63500	0.69452	0.57309	0.64023	0.70732

Table 3: Results of Watson Instances

Optimizer. This is due to the fact that the initial population, for $\lambda \approx 1$ is composed of high value of speed (3).

Table 3 shows the F values for Watson instances. In Agnetis instances, the maximum number of operations is 90 in instances 3_30_p. However, in Watson instances, the number of operations is ranged between 1000 (j=50 and v_{max}=20) and 4000 operations (j=200 and v_{max}=20). Therefore Watson instances are much larger than Agnetis instances. It can be observed that both algorithms were able to solve all instances with 50 and 100 jobs. The results for these instances were better for CP Optimizer for λ values lower than 0.6, meanwhile or GA had better results for $\lambda \in [0.6, 0.9[$. Figure 7 shows the average F value of 50 and 100 jobs for Watson instances. It can be observed that although both algorithms have similar behavior, GA is most focused on minimizing makespan (highest value for $\lambda = 0.4$) meanwhile CP Optimizer is most focused on minimizing energy consumption (highest value for $\lambda = 0.6$).

However for instances of 200 jobs, CP Optimizer was unable to solve almost all instances ranged for $\lambda \in]0, 1[$. This means CP Optimizer is not able to solve large-scale instances in a reasonable time so metaheuristic techniques are needed to obtain optimized solutions in a given time.

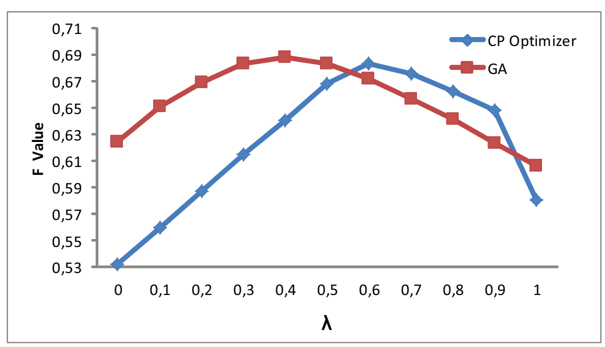


Figure 7: Average F value (F_50 and F_100) for Watson instances

[2]http://gps.webs.upv.es/jobshop/

Conclusions and Further Works

Many real life problems can be modeled as a job-shop scheduling problem in which machines can consume different amounts of energy to process tasks at different rates. It represents an extension of the classical job-shop scheduling problem, where each operation has to be executed by one machine and this machine has the possibility to work at different speeds. In this paper, we present a genetic algorithm to model and solve this problem. The inclusion of energy consumption in the chromosome gives us the opportunity to guide the search toward an optimized solution in an efficient way. A comparative study was carried out to analyze the behavior of our genetic algorithm against a well-known solver: IBM ILOG CPLEX CP Optimizer. The evaluation shows that our Genetic Algorithm had a similar behavior than CP Optimizer for small instances. However, for large instances, CP Optimizer was unable to solve them in the given time meanwhile our GA could solve all instances with the same optimality degree. Thus, our technique can be useful to be applied in large scale scheduling problems.

As conclusion, different solutions can be achieved to this problem, so given a makespan threshold, a solution that minimize energy consumption can be obtained and viceversa, given a energy consumption threshold, a solution that minimize makespan can be obtained. This represents an interesting trade-off for researchers in the area.

In further works, we will add a local search technique to improve the obtained solutions. This technique can be added inside the GA and also as a postprocess by increasing the speed of the latest tasks responsible of the makespan value. Furthermore, we will analyze the robustness of the obtained schedules due to the fact that energy-aware solutions are considered more robust that makespan-optimized solutions (Salido et al. 2013).

Acknowledgments

This research has been supported by the Spanish Government under research project MINECO TIN2013-46511-C2-1 and the CASES project supported by a Marie Curie International Research Staff Exchange Scheme Fellowship within the 7th European Community Framework Programme under the grant agreement No 294931.

References

Agnetis, A.; Flamini, M.; Nicosia, G.; and Pacifici, A. 2011. A job-shop problem with one additional resource type. *Journal of Scheduling* 14(3):225–237.

Beasley, D.; Martin, R.; and Bull, D. 1993. An overview of genetic algorithms: Part 1. fundamentals. *University computing* 15:58–58.

Billaut, J.; Moukrim, A.; and Sanlaville, E. 2008. Flexibility and robustness in scheduling. Wiley.

Blazewicz, J.; Cellary, W.; Slowinski, R.; and Weglarz, J. 1986. Scheduling under resource constraints-deterministic models. *Annals of Operations Research* 7:1–356.

BMWi. 2009. German federal ministry of economics and technology: Energy statistics.

Bruzzone, A.; Anghinolfi, D.; Paolucci, M.; and Tonelli, F. 2012. Energy-aware scheduling for improving manufacturing process sustainability: A mathematical model for flexible flow shops. *CIRP Annals-Manufacturing Technology*.

Dahmus, J., and Gutowski, T. 2004. An environmental analysis of machining. In *ASME International Mechanical Engineering Congress and RD&D Exposition, Anaheim, California, USA*.

Dai, M.; Tang, D.; Giret, A.; Salido, M.; and Li, W. 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.

Duflou, J.; Sutherland, J.; Dornfeld, D.; Herrmann, C.; Jeswiet, J.; Kara, S.; Hauschild, M.; and Kellens, K. 2012. Towards energy and resource efficient manufacturing: A processes and systems approach. *CIRP Annals-Manufacturing Technology*.

Escamilla, J. 2014. https://www.dsic.upv.es/~jescamilla.

Fang, K.; Uhan, N.; Zhao, F.; and Sutherland, J. 2011. A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction. *Journal of Manufacturing Systems* 30(4):234–240.

Garrido, A.; Salido, M.; Barber, F.; and López, M. 2000. Heuristic methods for solving job-shop scheduling problems. In *ECAI-2000 Workshop on New Results in Planning, Scheduling and Design, Berlin*, 36–43.

Huang, K., and Liao, C. 2008. Ant colony optimization combined with taboo search for the job shop scheduling problem. *Computers & Operations Research* 35(4):1030–1046.

IBM. 2007. Modeling with IBM ILOG CP Optimizer - practical scheduling examples. *IBM.*

IBM. 2012. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

Laborie, P. 2009. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR09)*, 148–162.

Li, W.; Zein, A.; Kara, S.; and Herrmann, C. 2011. An investigation into fixed energy consumption of machine tools. *Glocalized Solutions for Sustainability in Manufacturing* 268–273.

Li, L.; Yan, J.; and Xing, Z. 2013. Energy requirements evaluation of milling machines based on thermal equilibrium and empirical modelling. *Journal of Cleaner Production*.

Malakooti, B.; Sheikh, S.; Al-Najjar, C.; and Kim, H. 2013. Multi-objective energy aware multiprocessor scheduling using bat intelligence. *Journal of Intelligent Manufacturing* 24(4):805–819.

Mestl, H. E.; Aunan, K.; Fang, J.; Seip, H. M.; Skjelvik, J. M.; and Vennemo, H. 2005. Cleaner production as climate investmentintegrated assessment in taiyuan city, china. *Journal of Cleaner Production* 13(1):57–70.

Mouzon, G., and Yildirim, M. 2008. A framework to minimise total energy consumption and total tardiness on a single machine. *International Journal of Sustainable Engineering* 1(2):105–116.

Mouzon, G.; Yildirim, M.; and Twomey, J. 2007. Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research* 45(18-19):4247–4271.

Neugebauer, R.; Wabner, M.; Rentzsch, H.; and Ihlenfeldt, S. 2011. Structure principles of energy efficient machine tools. *CIRP Journal of Manufacturing Science and Technology* 4(2):136–147.

Salido, M. A.; Escamilla, J.; Barber, F.; Giret, A.; Tang, D.; and Dai, M. 2013. Energy-aware parameters in job-shop scheduling problems. *GREEN-COPLAS 2013: IJCAI 2013 Workshop on Constraint Reasoning, Planning and Scheduling Problems for a Sustainable Future* 44–53.

Seow, Y., and Rahimifard, S. 2011. A framework for modelling energy consumption within manufacturing systems. *CIRP Journal of Manufacturing Science and Technology* 4(3):258–264.

Varela, R.; Serrano, D.; and Sierra, M. 2005. New codification schemas for scheduling with genetic algorithms. In *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*. Springer. 11–20.

Watson, J.-P.; Barbulescu, L.; Howe, A. E.; and Whitley, L. D. 1999. Algorithm performance and problem structure for flow-shop scheduling. In *AAAI/IAAI*, 688–695.

Weinert, N.; Chiotellis, S.; and Seliger, G. 2011. Methodology for planning and operating energy-efficient production systems. *CIRP Annals-Manufacturing Technology* 60(1):41–44.

Yan, J., and Li, L. 2013. Multi-objective optimization of milling parametersthe trade-offs between energy, production rate and cutting quality. *Journal of Cleaner Production*.

Yusoff, S. 2006. Renewable energy from palm oil–innovation on effective utilization of waste. *Journal of cleaner production* 14(1):87–93.