A weighted CSP approach to cost-optimal planning

Martin C. Cooper *, Marie de Roquemaurel and Pierre Régnier
IRIT, University of Toulouse III,
118 route de Narbonne, 31062 Toulouse, France
E-mail: {cooper,deroquemaurel,regnier}@irit.fr

For planning to come of age, plans must be judged by a measure of quality, such as the total cost of actions. This paper describes an optimal-cost planner which guarantees global optimality whenever the planning problem has a solution.

We code the extraction of an optimal plan, from a planning graph with a fixed number k of levels, as a weighted constraint satisfaction problem (WCSP). The specific structure of the resulting WCSP means that a state-of-the-art exhaustive solver was able to find an optimal plan in planning graphs containing several thousand nodes.

Thorough experimental investigations demonstrated that using the planning graph in optimal planning is a practical possibility for problems of moderate size, although not competitive, in terms of computation time, with optimal state-space-search planners. Our general conclusion is, therefore, that planning-graph-based optimal planning is not the most efficient method for cost-optimal planning.

Nonetheless, the notions of indispensable (sets of) actions and too-costly actions introduced in this paper have various potential applications in optimal planning. These actions can be detected very rapidly by analysis of the relaxed planning graph.

Keywords: optimal planning, planning graph, soft constraints, soft arc consistency

1. Introduction

After the progress made by non-optimal planners, optimal planning still remains an important challenge from both a theoretical and practical point of view. In this paper, we show how soft constraint satisfaction [67,59] can be used to find an optimal solution to a planning problem in which each action has an associated strictly positive cost. A k-level planning graph is a disjunctive structure, which can be constructed in polynomial time and space, containing all potential parallel plans of length k. The extraction of an optimal plan from this graph can be coded as a weighted CSP which mixes crisp binary constraints (such as mutexes) and soft unary constraints (representing the costs of actions). Since our basic planner is a generalization of GP-CSP [22], we call it GP-WCSP. It guarantees cost optimality of parallel plans of length k.

Different notions of optimality are appropriate for different problems. In many problems, we would like to find the minimal-cost plan (possibly using parallel plan length as a secondary optimization criterion). However, in certain problems we may wish to give priority to minimizing parallel-plan length, using total cost as a secondary optimization criterion. For example, suppose that subcontractors are employed to perform tasks (actions) and that each task has an associated financial cost and requires at most one working day. The subcontractors only guarantee that their task will be finished by the end of the working day, so it is therefore not possible to schedule two mutually-exclusive actions on the same day. Minimizing the number of levels in a parallel plan is equivalent to minimizing the number of days in the plan, which may be more important than minimizing financial cost (which is nevertheless still used as a secondary optimization criterion).

However, in most applications we are interested in finding a plan which minimizes total cost, regardless of plan length. A non-trivial extension of GP-WCSP, called GP-WCSP*, calculates an upper bound $MaxLev$ on the number of levels that need to be constructed in the planning graph to

*Corresponding author: Martin C. Cooper, IRIT, University of Toulouse III, 118 route de Narbonne, 31062 Toulouse, France

AI Communications
ISSN 0921-7126, IOS Press.

guarantee finding such a globally optimal solution. GP-WCSP* is effectively an intelligent form of iterative-deepening search where depth corresponds to the number of levels in a parallel plan. Pruning depends critically on finding a good plan as soon as possible. If a good plan exists with a small makespan, then it will be found after searching only a few levels. At each level k, the construction of the planning graph, before embarking on an exhaustive search, allows us to detect and eliminate actions which do not lead to a goal and hence cannot possibly be part of a valid parallel plan of length k. Actions which are too costly to be part of an optimal plan can also be eliminated.

By coding the extraction of a plan from the planning graph as a WCSP, we can take advantage of different techniques for solving weighted CSPs. Notably, the search tree can be considerably pruned by establishing a form of soft arc consistency at each node. The construction of the planning graph tells us which variables are required in the WCSP. As a by-product of our experimental trials, the translation of a large number of planning benchmarks into WCSP instances provides challenging benchmarks on which new WCSP techniques could be tested.

A theoretical bound on the length of an optimal plan allows us to provide a guarantee of optimality when GP-WCSP* attains this number of levels. This bound can be dramatically reduced by detection of indispensable sets, sets of actions one of which must be part of every valid plan. Different types of indispensable sets can be rapidly detected by solving relaxed planning problems related to the original problem. On extensive trials on benchmark problems, the bound on the number of planning-graph levels was reduced by an average of 56% allowing us to solve many instances to optimality.

The rest of the paper is organized as follows. Section 2 describes previous work about optimal approaches of planning. Section 3 is devoted to preliminary definitions. Sections 4 and 5 describe how our planner extracts an optimal plan from a planning graph (of a given number of levels k) using soft constraint techniques. This algorithm can be used to find an optimal-cost plan among parallel plans of bounded length k, but to guarantee global optimality the planning graph must be extended until a proof of optimality is obtained (Section 6). To limit the size of the planning graph to be constructed, we detect indispensable sets (Sections 7 and 8) as well as too-costly actions which cannot belong to an optimal plan (Section 9). Results of our experimental trials are reported in Section 10.

Throughout this article, we remain in the classical planning framework (instantaneous, static and deterministic actions) except that each action now has a cost.

2. Previous work

The first approaches to planning which guaranteed a certain form of optimality were those which translated planning problems into SAT problems. While Kautz and Selman [45,46] thus used a SAT coding assuming that the minimal length of a solution-plan is known in advance, later approaches used the planning graph of GRAPHPLAN [2] or a coding which represents all plans of a fixed length k, where k is incremented until a solution is found. These approaches, which impose a limit on the size or the number of levels of solution-plans, are based on solving NP-complete problems, whereas the propositional planning problem is PSPACE-complete [8]. However, the plans produced are only optimal in the number of steps. In the $\forall$-step semantic, each step represents a set of independent actions, while the $\exists$-step semantic [21,66,9] allows a set of actions to be executed in parallel at one time step if there exists at least one possible execution order of the actions. Optimality in the number of $\forall$-steps or $\exists$-steps is not necessarily related to the quality of the solution in terms of the number of actions. In the framework of planning with preferences, Giunchiglia and Maratea [30,29] describe SATPLAN$\prec$, a SATPLAN-based planner using an optimization criterion based on makespan (number of steps) and number of actions in the plan: given the optimal makespan, it returns plans with the minimal number of actions and maximal number of the satisfied soft goals, with respect to both cardinality and subset inclusions.

The success of the SAT approach initiated the use of other NP-complete formalisms such as the constraint satisfaction problem (CSP). The first CSP-based system CPLAN [69] achieves good performance by posing planning as a CSP but requires a new hand-generated encoding for each domain. The aspects of the CPLAN encodings which make

it so effective can, to a large extent, be automated by the GP-CSP planner [22] in which the planning-graph is coded in a conditional CSP [62]. More recently, a direct encoding of planning problems as CSPs has been proposed and implemented in the system CSPPLAN [58]. This encoding is not based on the planning graph although it captures all its properties and constraints. It allows additional binary constraints and sequence constraints to be detected and added. In the framework of planning with preferences, the planner PREFPLAN [5], which is built on the top of the GP-CSP planner, uses the preference specification formalism of TCP-nets [6]. The user provides constraints and preferences over possible goal states and PREFPLAN performs a TCP-net based optimization using a CSP encoding of the problem. The algorithm is sound, complete, and the solution-plan is Pareto-optimal with respect to the user's preferences. In the specific framework of flexible planning, the FGP planner [60,61] uses soft constraint satisfaction. In this context, preconditions are not always imperative, but can be relaxed with an associated damage to the resultant plan. The planning problem is encoded as a flexible CSP and a degree of satisfaction is associated with different goals and operators. The degree of satisfaction of a plan is the minimal satisfaction level of the operators used and the goal achieved. In this paper we study additive costs.

The search for a fixed-length plan can also be coded using integer programming (IP). In real-world planning problems, this approach is particularly interesting as mixed integer linear programming provides an appropriate environment for using numerical constraints and arbitrary linear objective functions. The planner ILP-PLAN [49] uses IP formulations for resource planning problems incorporating action costs and complex objectives. The propositional planner IPPLAN uses new IP formulations [70], one of which uses GRAPHPLAN's parallelism and allows at most one state change for each state variable per plan step to produce optimal makespan plans.

In optimal planning, another efficient method is heuristic planning. In the majority of these planners, the heuristic is computed by using a relaxed version of the original problem. This relaxation can essentially be performed by ignoring delete lists, approximating atom sets by smaller subsets, or ignoring certain atoms. HSP* is a family of optimal domain-independent planners built on ideas from their non-optimal predecessors, HSP and HSPr [3]. They can generate sequential plans, minimizing the sum of action costs. As optimal solutions must be guaranteed, they use admissible heuristics which are derived automatically from the problem representation [37]. Haslum et al. [36] introduce two refinements of these admissible heuristics for domain-independent planning. FDP [32,31] is another planner based on the paradigm of planning as constraint satisfaction, that searches for optimal sequential plans in a structure related to GRAPHPLAN's planning graph. In this planner, the use of a constraint representation helps focus search on the relevant choice points, to reduce the branching factor. Each time the graph is extended, a search for a sequential plan is made using specific consistency rules and filtering and decomposition mechanisms. The search algorithm is a Depth-First Iterative Deepening or IDA* with an admissible heuristic based on unit-cost actions. The authors claim that FDP could easily be extended to compute minimal-cost plans when actions have distinct costs. Recently, several improvements have been carried out in the field of heuristic planning. To deal with planning models where costs are associated with actions and states, Bonet and Geffner [4] formulate a more expressive planning model (and a corresponding admissible heuristic) where preferences in the form of penalties and rewards are associated with fluents as well as actions. All heuristic values can be computed in time linear in the size of the compilation and, using this heuristic, a best-first search algorithm can handle negative costs and produce optimal plans. Brafman and Chernyavsky [5] also attempt to generate a plan for the optimal feasible goal state for a problem in which costs and rewards are combined additively.

An abstraction is a mapping that reduces the size of the state space, by collapsing several states into one. When the abstract space is small enough, it is feasible to find the optimal solution by blind search. During a preprocessing phase, distances in the abstract space are computed and memorized. Then, during search, heuristic evaluation can be performed by a simple lookup. Edelkamp [23] uses data structures such as BDDs to compact the representation of sets of states, and uses external storage to explore large search spaces. Helmert et al. [40] describe a way to calculate a new admissi-

ble pattern database heuristic for forward search which can be generalized to the case of minimizing the sum of positive action costs. If a divide-and-conquer solution reconstruction is used to reduce the memory requirements of search, Zhou and Hansen [74,75] show that breadth-first search can become more efficient than best-first search. Their planner BFHSP finds optimal and approximate solutions when all actions have unit cost. They have also built an anytime algorithm based on A* which uses weighted heuristic search to find an initial, possibly suboptimal solution and continues to search for improved solutions until convergence to a provably optimal solution [34].

In this paper we show how weighted-CSP techniques can be used within a GRAPHPLAN-based planner to minimize an objective function based on the sum of action costs and/or the number of $\forall$-step levels. We compare experimentally our planner with planners based on heuristic search.

3. Preliminary definitions

Definition 3.1 (planning problem with costs) *A planning problem with costs (or optimal-planning problem) is a triple* $\Pi = \langle A, I, G\rangle$ *such that:*

1. *the initial state* I *is a finite set of literals (also known as fluents);*
2. A *is a set of actions, i.e. of triples* $\langle prec(a), eff(a), cost(a)\rangle$*, where* $prec(a)$ *is the set of propositional preconditions of the action* a*,* $eff(a)$ *is the set of propositional effects of the action* a *and* $cost(a)$ *is a strictly positive natural number representing the cost of the action* a*;*
3. *the goal* G *is the set of propositions to be satisfied. A proposition* p *is satisfied in a state* S *iff* $p \in S$*.*

Given an action a, we denote the set of positive fluents (non-negated literals) in $eff(a)$ by $add(a)$ and the set of negative fluents (negated literals) in $eff(a)$ by $del(a)$.

Definition 3.2 (application of an action) *The application of an action* a *in a state* S *(denoted* $S \uparrow a$*) is possible iff all the fluents of prec(a) are satisfied in* S*. The result is a state* S' *such that:* $S' = (S - del(a)) \cup add(a)$*.*

One of the most efficient and influential algorithms in the field of planning is that of GRAPHPLAN [2], which opened the way to compilation-based planning methods (as described in Section 1). These techniques begin by constructing a disjunctive structure (the polynomial-time and space constructible planning graph in the case of GRAPHPLAN), which can be considered as a compact summary of all plans up to a given maximal length, and then extract a solution from this structure.

Definition 3.3 (planning graph) *A planning graph is a directed graph composed of several levels each with action nodes and fluent nodes, together with precondition, add and delete arcs. Level 0 contains only the fluents of the initial state. Each level* $i > 0$ *is composed of all those actions which are applicable from the fluents of level* $i-1$*, together with the fluents which are produced by these level* i *actions. The arcs represent the relations between the actions and the fluents: the level* i *actions are linked to the precondition fluents of level* $i-1$ *by precondition arcs, and to their level* i *adds and deletes by add/delete arcs.*

Fluents f *are maintained between level* $i-1$ *and level* i *by noop actions which have the single fluent* f *as both precondition and effect and zero cost. At each level of the graph, binary mutual exclusions (mutexes) between pairs of actions and between pairs of fluents are computed and memorized.*

Definition 3.4 (mutual exclusion) *Two actions* a_1 *and* a_2 *are mutex at level* i *of the graph (denoted* $mutex(a_1, a_2, i)$*) iff:*

1. *an action deletes a fluent required or added by the other action:* $\Big((del(a_1) \cap (prec(a_2) \cup add(a_2))) \cup (del(a_2) \cap (prec(a_1) \cup add(a_1)))\Big) \neq \emptyset$ *or*
2. *the actions have precondition fluents which are mutex at level* $i-1$ *(they therefore cannot be triggered at the same time):* $\exists (p, q) \in prec(a_1) \times prec(a_2)$*,* $mutex(p, q, i-1)$*.*

Two fluents p *and* q *at a level* i *of the graph are mutex iff all the pairs of actions which produce them at level* i *are mutex.*

Definition 3.5 (independent set of actions) *Two actions* a *and* b *are independent (denoted* $a\#b$*) iff*

they do not verify (1) in Definition 3.4. A set of actions $Q_i = \{a_1, \ldots, a_n\}$ is an independent set iff all its actions are pairwise independent: $\forall\, a, b \in Q$, $a\#b$. Applying a set of independent actions $Q = \{a_1, \ldots, a_n\}$ in a state S (denoted $S \uparrow Q$) is possible iff $\forall\, a \in Q$, $prec(a)$ is satisfied in S. The result is the state $S' = \big(S - \bigcup_{a\in Q} del(a)\big) \cup \bigcup_{a\in Q} add(a)$.

When a set of actions Q is independent, the application of the actions in Q leads to an identical result-state whatever the order in which they are executed.

Definition 3.6 (parallel plan) *An n-level parallel plan is a sequence of independent sets Q_i of actions, denoted $P =< Q_1, \ldots, Q_n >$. If all the Q_i are singletons then the plan P is a sequential plan. The application of a parallel plan P to a state S is possible iff Q_1 is applicable in S, producing a state S_1, Q_2 is applicable in S_1, producing a state S_2, ..., Q_n is applicable in S_{n-1}, producing the state $S_n = ((\ldots((S \uparrow Q_1) \uparrow Q_2)\ldots) \uparrow Q_n)$. We say that S_n is reachable from S, and P is a correct parallel plan. A correct parallel plan P is a parallel solution-plan (or valid parallel plan) iff G is satisfied in the state S_n.*

GRAPHPLAN constructs parallel plans and hence in this article we are only concerned with this type of plan.

Definition 3.7 (level-off) *In the planning graph, level-off is attained when two consecutive levels contain the same actions and fluents as well as the same mutexes between actions and fluents.*

Definition 3.8 (reduced graph) *The reduced graph is a version of the planning graph in which, starting with the goal fluents at level k, we delete all actions at level k (together with their preconditions arcs) which do not produce these fluents. This process is then repeated at level $k-1$ by the deletion of all actions which do not produce any of the preconditions of the remaining level-k actions, and so on down to level 1.*

Definition 3.9 (relaxation) *The relaxation a^+ of an action a consists of ignoring its deletes: $a^+ = \langle prec(a), add(a), cost(a)\rangle$. The relaxed version of the problem $\Pi = \langle A, I, G\rangle$ is a triple $\Pi^+ = \langle A^+, I, G\rangle$ where $A^+ = \{a^+ | a \in A\}$. The planning graph corresponding to the relaxed problem, constructed until level-off, is known as the relaxed graph. The reduced relaxed graph, denoted G_r^+, is the graph resulting from the reduction of the relaxed graph; we denote by $A_r^+ \subseteq A$ the set of actions occurring in this graph.*

Definition 3.10 (plan metric) *The quality of a plan P is estimated by a function known as a plan metric. In this article we assume an additive metric of the form $metric(P) = \sum_{a\in P} cost(a)$. We denote by $\mathcal{P}_i$ the set of solution-plans of length less than or equal to i levels. $\mathcal{P}_i^*$ is the set of solution-plans in $\mathcal{P}_i$ which minimize the metric: $\forall\, P \in \mathcal{P}_i^* : metric(P) = \min_{P'\in\mathcal{P}_i} metric(P')$. P_i^* denotes a plan in $\mathcal{P}_i^*$ and C_i^* its cost.*

For an optimal-planning problem $\Pi = \langle A, I, G\rangle$, our aim is to find a parallel plan which minimizes the metric and, among such minimal-cost plans, has the least number of levels.

We performed our experimental trials on classic benchmark planning problems. For many of these benchmark domains, it has been shown that determining the existence of a plan of bounded length is NP-complete [38]. The addition of an optimization criterion renders these problems even more difficult to solve in practice.

We give a transportation problem which will serve as a running example throughout the paper. This example is part of a class of shortest-path problems which can be solved in polynomial time, as a function of the size of the graph, by dynamic programming. We use it simply to illustrate the notions introduced in this paper. The problem concerns five cities A, B, C, D and E, a vehicle v and a crate c. The fluent v_i represents the fact that the vehicle v is in city i. The vehicle v is driven from city i to city j by means of the action $move_{ij}$. The cost of this action depends on the distance between the cities. The fluent c_i represents the fact that the crate is in city i, and c_v the fact that the crate is on board the vehicle v. The crate c can be loaded into the vehicle in city i by means of the action $load_i$, with a cost of 5, and unloaded in city i by means of the action $unload_i$ which has a cost of 3. The problem $\Pi = \langle A, I, G\rangle$ is defined by the initial state $I = \{v_A, c_A\}$, the goal $G = \{c_B\}$ and the following set of actions A: $\forall\, i \in$ {A,B,C,D,E},

- $\forall\, j \in$ {A,B,C,D,E}, $move_{ij} : \langle\{v_i\}, \{v_j, \neg v_i\}, cost_{ij}\rangle$. Fig. 1 indicates the possible moves and their costs.

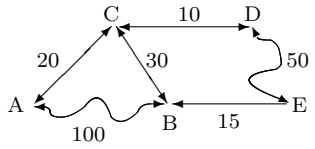

Fig. 1. Possible moves and their associated costs in a simple transportation planning problem

- $load_i$: $\langle\{v_i, c_i\}, \{c_v, \neg c_i\}, 5\rangle$.
- $unload_i$: $\langle\{v_i, c_v\}, \{c_i, \neg c_v\}, 3\rangle$.

To start with, the planning graph is constructed until level 3 is reached, since this is the first level in which the goal fluent appears. Extracting an optimal parallel plan from this graph, we obtain the solution-plan with cost $C_3^* = 108$: $\langle\{load_A\}, \{move_{AB}, noop_{c_v}\}, \{unload_B\}\rangle$ which corresponds to the unique sequential plan $\langle load_A, move_{AB}, unload_B\rangle$. Although this is the minimal-cost plan among length-3 parallel plans, it is not optimal, since there is an optimal length-4 plan: $\langle load_A, move_{AC}, move_{CB}, unload_B\rangle$ of cost 58. An important question which then arises is: how many levels of the planning graph do we need to construct to prove that this is a globally optimal plan?

The Weighted Constraint Satisfaction Problem (WCSP) is a generalization of the standard CSP which allows us to model optimization problems [59]. Infinite costs can be used to model strict constraints, such as mutual exclusion, whereas finite costs can be used to model preferences, probabilities or financial cost.

Definition 3.11 (binary WCSP) *A binary WCSP is composed of:*

1. *a set of n variables $X_1, \ldots, X_n$ with respective domains $D_1, \ldots, D_n$,*
2. *a set of cost functions of arity no greater than 2. The unary cost functions $c_i : D_i \rightarrow \mathbb{R}^{\geq 0} \cup \{\infty\}$ (for $i \in N = \{1, \ldots, n\}$) associate a cost with each possible assignment to X_i. The binary cost functions $c_{ij} : D_i \times D_j \rightarrow \mathbb{R}^{\geq 0} \cup \{\infty\}$ (for $(i, j) \in E$, where $\langle N, E\rangle$ is a directed graph) associate a cost with each pair of values simultaneously assigned to the pair of variables (X_i, X_j). The nullary cost function is simply a constant $c_\emptyset \in \mathbb{R} \cup \{\infty\}$ to be added to the cost of any assignment.*

The cost of an assignment $x = (x_1, \ldots, x_n)$ to variables $X_1, \ldots, X_n$ is given by:

$$Cost(x) = c_\emptyset \; + \sum_{i \in N} c_i(x_i) + \sum_{(i,j) \in E} c_{ij}(x_i, x_j)$$

A solution to the WCSP is an assignment $x \in D_1 \times \ldots \times D_n$ which minimizes $Cost(x)$.

4. Plan-extraction coded as a WCSP

As demonstrated by [22], the extraction stage of GRAPHPLAN can be seen as a Conditional Constraint Satisfaction Problem [62]. Initially, only a subset of the variables are active, and the objective is to find assignments for all active variables that are consistent with the constraints. Indeed, during this extraction stage, there is a CSP variable for every fluent f at a level i of the planning graph; the set of actions that achieve f constitutes its domain and the mutexes produced during the construction stage are the constraints of the CSP. Starting at the goal fluents, GRAPHPLAN tries to extract a plan from the planning graph by assigning a value (an action) to every variable (fluent) so as to satisfy the set of constraints (mutexes). The assignment of values to variables is a dynamic process because every assignment of a variable at a level i activates other variables (the preconditions of the selected action) in the previous level. In the extraction of an optimal solution-plan from the planning graph, we retain the essential idea that fluents are variables, actions are values and mutexes are strict constraints. Once the planning graph has been constructed, we reduce it by eliminating nodes which do not lead to any goal fluent. The reduced planning graph is then coded as a WCSP as follows, with two new steps (5 and 6) being required compared to the coding in [22]:

1. *Creation of variables and domains:* for each fluent (not present in the initial state), we create a variable whose domain is the set of actions which produce this fluent. For each fluent (not present in the goal state) we add the value -1 to represent its non-activation.
2. *Translation of mutexes between fluents:* for all mutex fluents f_i and f_j, this mutual exclusion is coded by a constraint which states that f_i and f_j cannot both be activated: $(f_i = -1) \vee (f_j = -1)$.

3. *Translation of mutexes between actions:* for all mutex actions a and b with (respective) effects f_i and f_j ($f_i \neq f_j$), this mutual exclusion is coded by a constraint which states that f_i and f_j cannot simultaneously be triggered by the actions a and b: $\neg((f_i = a) \wedge (f_j = b))$.
4. *Translation of activity arcs:* the activation of a fluent f_i produced by an action a implies the activation of the preconditions of this action. This is coded by activity constraints: $\forall f_j \in prec(a)$, $(f_i = a) \Rightarrow (f_j \neq -1)$.
5. *Translation of the cost of actions:* For each fluent f_i, there is a corresponding unary cost function which associates with each assignment $f_i = a$ the cost of the action a. Noops have cost 0.
6. *Actions with multiple effects:* when an action a produces several fluents $f_i \in \mathit{eff}(a)$, the cost of this action could be counted several times [20]. To avoid this problem, we create an intermediary fluent f^{int} with domain $\{a, -1\}$. Furthermore, the action a is replaced by an imaginary action a^{int} (of zero cost) in the domains of each of the fluents f_i. We add the following activity constraint between the fluent f^{int} and each of the fluents $f_i \in \mathit{eff}(a)$: $(f_i = a^{int}) \Rightarrow (f^{int} = a)$.

Consider our running example. We initially construct a planning graph up to level 3, since this is the first level at which all the goals appear without mutex. Figure 2 shows the reduced planning graph after renaming of variables. The coding of this graph as a WCSP is shown in Figure 3 where all binary constraints are mutexes (and hence all arcs between values in different domains represent infinite costs). The optimal solution to this WCSP is the plan $P_3^* = \langle\{load_A\}, \{move_{AB}, noop_{c_v}\}, \{unload_B\}\rangle = \langle load_A, move_{AB}, unload_B\rangle$ of cost 108.

We explain in the following section how to efficiently solve the WCSP resulting from the encoding described above.

5. Arc consistency in the resulting WCSP

In this section we review the different notions of soft arc consistency which we used in our experimental trials to speed up the search for a solution to the WCSP derived from the planning graph. In our trials soft arc consistency was established at each node of a branch-and-bound search tree. We performed experimental trials to compare the different forms of soft arc consistency described in this section. Recall that a solution to a binary WCSP is an n-tuple x which minimizes $\sum_i c_i(x_i) + \sum_{ij} c_{ij}(x_i, x_j) + c_\emptyset$, where c_i denotes the unary cost function on variable X_i, c_{ij} denotes the binary cost function on variables X_i, X_j and $c_\emptyset$ represents the nullary cost function (which is independent of the values assigned to the variables). Cost functions take values in $\mathbb{R}_{\geq 0} \cup \{\infty\}$.

Maintaining arc consistency at every node of a backtracking search tree is a standard technique in solving classical constraint satisfaction problems. Recent research on soft constraints has produced various different generalizations of arc consistency to WCSPs, all of which propagate inconsistencies and also produce a lower bound on the cost of a solution. Unlike classical CSPs, weighted CSPs do not, in general, have a unique arc consistency closure [17]. There is a trade-off between the time spent in finding a lower bound and the quality of this bound. For example, finding an optimal arc consistency closure can be achieved by solving a linear program but this is too time-consuming to be applied at every node of a branch-and-bound search tree [16].

Reducing the size of the search space by propagating inconsistencies can be directly generalized from CSPs to WCSPs, by propagating infinite costs [59]. Finite costs can also be propagated but such propagations must be compensated: simultaneously subtracting α from $c_{ij}(a, u)$ (for each value u in the domain of variable X_j) while adding α to $c_i(a)$ produces an equivalent WCSP [17]. When $\alpha > 0$ this operation is known as a projection; when $\alpha < 0$ it is known as an extension. A third operation is unary projection, which simultaneously subtracts α from $c_i(u)$ (for each value u in the domain of X_i) while adding α to $c_\emptyset$. In all three cases, the operation can only be applied if the resulting costs are all non-negative. The aim of the propagation of finite costs is to increase the lower bound $c_\emptyset$ in order to reduce the search space explored by branch-and-bound.

Applying all possible unary projection operations establishes node consistency (NC) [56]. Propagating all infinite costs (between unary and binary constraints) and applying projection opera-

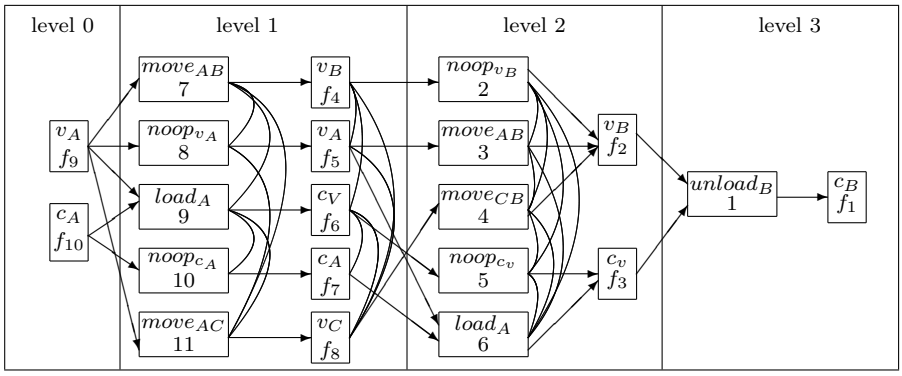

Fig. 2. planning graph after reduction and renaming.

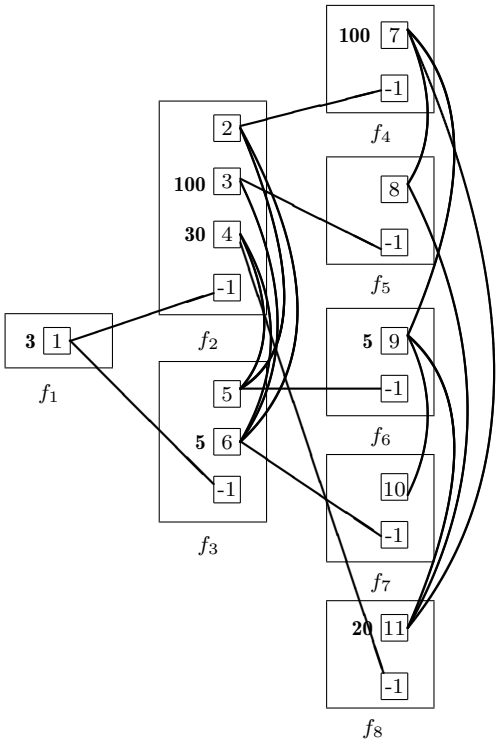

Fig. 3. The WCSP derived from the planning graph of Figure 2; unary costs are shown next to domain values and arcs link incompatible values in distinct domains.

tions until convergence establishes (soft) arc consistency (SAC) [17]. In order to have non-zero costs available as soon as possible during search, directional arc consistency (DAC) always chooses to send costs (via projection and extension operations) towards variables which occur earlier in the instantiation order. This also has the effect of tending to group together costs at the same variables, which naturally leads to a larger value of $c_\emptyset$ after establishing node consistency. Full directional arc consistency (FDAC) [13] is the combination of directional arc consistency and arc consistency.

As an example of FDAC, consider the WCSP given in box (1) of Figure 4 composed of three variables X_i with domains $\{a_i, b_i\}$ $(i = 1, 2, 3)$. An arc between two values represents a cost of 1. For example, the arc (a_1, a_3) means that $c_{13}(a_1, a_3) = 1$. FDAC begins by projecting a cost of 1 from $c_{12}(b_1, a_2)$, $c_{12}(b_1, b_2)$ down to $c_1(b_1)$ (which establishes DAC) and then from $c_{23}(a_2, b_3)$, $c_{23}(b_2, b_3)$

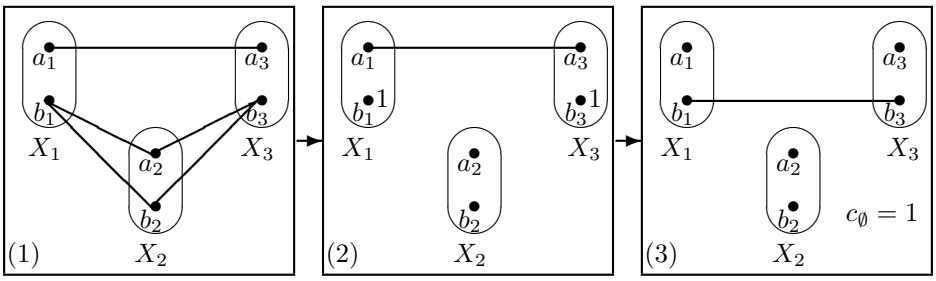

Fig. 4. Example of the simplification of a WCSP by FDAC.

down to $c_3(b_3)$ (a SAC operation which sends costs towards variables which occur later in the instantiation order). The resulting WCSP, shown in box (2) of Figure 4, is no longer directional arc consistent: extending a cost of 1 from $c_3(b_3)$ up to c_{13} allows us to project a cost of 1 from c_{13} down to $c_1(a_1)$. Box (3) of Figure 4 shows the WCSP obtained by then applying unary project to c_1. This WCSP is full directional arc consistent according to the variable order X_1, X_2, X_3.

Suppose that during branch-and-bound the cost of the best solution found so far is m. Then, assuming that the aim is to find a single optimal solution, we can prune a branch of the search tree as soon as $c_\emptyset \geq m$. Furthermore, we can set to ∞ any unary cost $c_i(a) \geq m - c_\emptyset$ and any binary cost $c_{ij}(a, b) \geq m - (c_i(a) + c_j(b) + c_\emptyset)$. Applying these latter operations provides a stronger form of arc consistency, but increases the worst-case time complexity of FDAC from $O(ed^2)$ [13] to $O(end^3)$ [57], where e is the number of binary constraints, n the number of variables and d the maximum domain size.

FDAC has been extended to existential directional arc consistency (EDAC) [18] which also performs the following operation: if for *each* value a in the domain of variable X_i, there exists a neighbouring variable X_j such that it is possible to increase $c_i(a)$ by sending costs from c_j and c_{ij}, then perform all these operations and then apply a unary projection operation at X_i to increase $c_\emptyset$.

Consider the WCSP in Figure 3. The optimal solution P_3^* to this WCSP is a minimum-cost plan among the parallel-plans stored in the planning graph of length 3 and has cost 108. Applying FDAC allows us to establish a lower bound of $c_\emptyset = 58$ before even embarking on a branch-and-bound search.

The resulting equivalent WCSP is shown in Figure 5. In this example, there is a dramatic difference between FDAC and NC, since NC only produces a lower bound of $c_\emptyset = 3$. Although, in the original problem, finite non-zero costs only occurred in the unary cost functions, propagating these costs through binary constraints allows us to obtain a much better lower bound. In the WCSP produced by establishing FDAC, shown in Figure 5, finite non-zero costs now occur in both the unary and binary cost functions. In this example, EDAC produces the same lower bound as FDAC.

6. Finding a globally optimal plan

We have seen how GP-WCSP [14] finds an optimal solution among parallel plans involving a minimum number of levels. In many problems this will not be a globally optimal solution. For example, in route-planning problems, if the shortest plan involves an expensive direct flight, a longer but less expensive plan often exists consisting of a complex itinerary involving several trains or buses.

The anytime algorithm GP-WCSP* (Algorithm 1) we have developed is based on the GRAPHPLAN algorithm. At the end of the planning-graph expansion phase, this graph is coded as a WCSP (as described in Section 4) which is then solved (as described in Section 5). The cost of the optimal plan found at a given level of the planning graph is used as an upper bound during the subsequent search for a better plan-solution at the next level.

Our algorithm is guaranteed to return an optimal plan P^* *if it exists*, under the assumption that all actions have strictly positive costs. Indeed, in order to calculate $MaxLev$, the maximum number of planning-graph levels to construct, we need first to find at least one valid plan. However, as for all previous GRAPHPLAN-based planners (e.g. SATPLAN'06 [47,48], GP-CSP [22], MAXPLAN [11]) termination is not guaranteed when no solution ex-

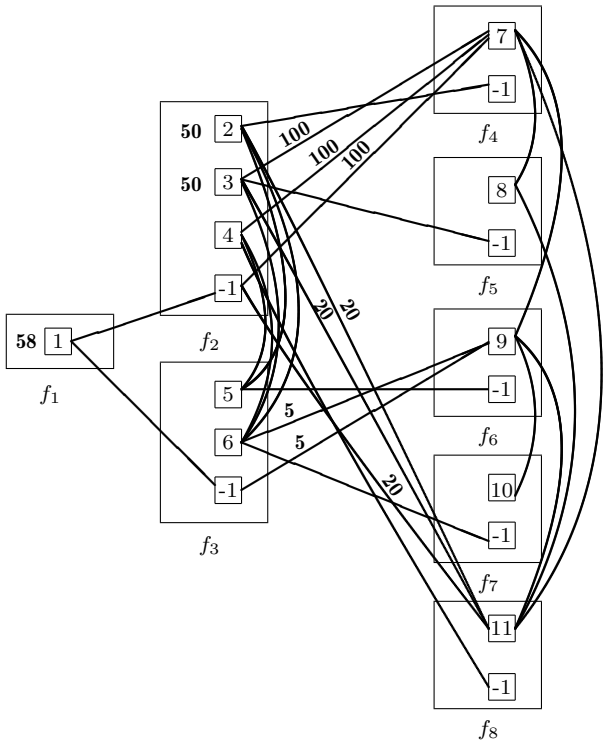

Fig. 5. Result of applying FDAC to the WCSP of figure 3.

ists. Termination was only guaranteed in the original GRAPHPLAN [2] by a process called memoization which involves storing a possibly exponential number of unreachable goal sets.

We now describe how to calculate $MaxLev$, the number of levels to construct in order to guarantee finding a cost-optimal plan. This upper bound can be significantly decreased by detecting certain properties of actions or sets of actions, as described in the following section.

Only actions in A_r^+ (the set of actions which lead to a goal fluent in the planning graph of the relaxed problem) can be part of a valid plan. Let C_{min} be the minimum cost of these actions:

$$C_{min} = \min_{a \in A_r^+} cost(a) \tag{1}$$

The set A_r^+ is a subset of the original set of actions A which are all assumed to have strictly positive costs. Note that A contains neither the zero-cost noop actions introduced during the construction of the planning graph nor the zero-cost imaginary actions a^{int} introduced during the coding of the planning graph as a WCSP (and described in Step (6) of Section 4).

Let C_k^* be the cost of an optimal plan P_k^* among parallel plans of length k and let C^* be the cost of a globally optimal plan. We can use C_{min} and C_k^* to bound the length of an optimal plan. In the worst case, all optimal plans P^* are sequential plans composed only of actions of cost C_{min}.

Thus, the number of levels we need to construct in order to be sure of finding an optimal plan is at most equal to:

$$MaxLev \ = \ \lceil C_k^*/C_{min} \rceil - 1 \tag{2}$$

since any plan containing $\lceil C_k^*/C_{min} \rceil$ actions has a cost of at least C_k^*. If this value of $MaxLev$ is less than or equal to the current number of levels in the planning graph, then this implies that the current best plan P_k^* is globally optimal.

In the running example, the minimum-cost action which could possibly be useful for attaining the goal is $unload_B$ of cost 3. The first value calculated for $MaxLev$ using P_3^* is $MaxLev = 35$.

Equation (2) considers the worst case in which all actions of the optimal plan have the same cost C_{min}. The value of $MaxLev$ can be improved by estimating in a more precise manner the costs of actions which are actually used in the optimal plan. This is the subject of the following section.

Algorithm 1. Resolution of an optimal-planning problem $\Pi\langle A, I, G\rangle$

Functions:

- *constructGraph*: constructs the graph until all goals appear in the current level without mutex. If level-off is attained before this occurs, then return failure.
- $constructGraphNextLevel(G_k, A)$: returns the $(k+1)$-level graph containing only actions from A.
- $codeWCSP(G)$: returns the WCSP corresponding to the reduced graph G.
- $solveWCSP(wcsp, C)$: returns an optimal solution to the WCSP and its cost. If this is not strictly less than C, then return failure.
- $calculateMaxLev(C, A)$: updates the set of actions A and calculates $MaxLev$.

Algorithm GP-WCSP*:

// initialization, search for a first solution
$G_k \leftarrow constructGraph(A, I, G)$
if the construction of the graph G_k is a failure **then**
  $failure$ // the problem has no solution
end if
$wcsp \leftarrow codeWCSP(G_k)$
$(P_k^*, C_k^*) \leftarrow solveWCSP(wcsp, \infty)$
while $P_k^* = failure$ **do**
  $k \leftarrow k + 1$
  $G_k \leftarrow constructGraphNextLevel(G_{k-1}, A)$
  $wcsp \leftarrow codeWCSP(G_k)$
  $(P_k^*, C_k^*) \leftarrow solveWCSP(wcsp, \infty)$
end while
$MaxLev, A \leftarrow calculateMaxLev(C_k^*, A)$

// anytime loop
while $k < MaxLev$ **do**
  $k \leftarrow k + 1$
  $G_k \leftarrow constructGraphNextLevel(G_{k-1}, A)$
  $wcsp \leftarrow codeWCSP(G_k)$
  $(P_k^*, C_k^*) \leftarrow solveWCSP(wcsp, C_{k-1}^*)$
  if $P_k^* = failure$ **then**
    $P_k^*, C_k^* \leftarrow P_{k-1}^*, C_{k-1}^*$
  else
    $MaxLev, A \leftarrow calculateMaxLev(C_k^*, A)$
  end if
end whileReturn (P_k^*, C_k^*).

7. Indispensable (sets of) actions

It is well known that planning can be accelerated by initial analysis of problem domains. Various different approaches have been used, most notably: detection of invariants [64,65,50,27,28,24, 25,55], elimination of fluents resulting from static predicates [72,54], taking advantage of symmetries [24,25,33], hierarchical decomposition of domains [68,73,51,1,7], and serialization of subgoals [52,53,63,42]. This section presents methods for determining actions, and then sets of actions, which are indispensable for a solution to exist.

7.1. Indispensable actions

Definition 7.1 (indispensable action) *An action a is indispensable in a problem $\Pi = \langle A, I, G\rangle$ iff it occurs in every solution to Π.*

Indispensable actions are also known as action landmarks [44]. The notion of indispensable action can be seen as a generalization of the notion of frozen variable; a CSP variable is *frozen* if it takes the same value in all possible solutions. The problem of determining whether a variable in a CSP instance is frozen is DP-complete [43]. Clearly, determining whether an action is indispensable in $\Pi = \langle A, I, G\rangle$ is equivalent to solving the planning problem $\Pi = \langle A - \{a\}, I, G\rangle$ and hence PSPACE-hard [8]. We avoid intractability by detecting only those indispensable actions that can be detected via the relaxed planning graph.

Lemma 7.2 *If the problem $\Pi^+_{-a} = \langle A^+ - \{a\}, I, G\rangle$ has no solutions, then $\Pi_{-a} = \langle A - \{a\}, I, G\rangle$ has no solutions either and a is an indispensable action for the problem Π.*

In the example, $unload_B$ and $load_A$ are indispensable actions. The detection of indispensable actions immediately provides a lower bound on the cost of a solution plan, namely the sum of the costs of indispensable actions. In our approach, taking into account indispensable actions leads to a better estimation of $MaxLev$ (which is necessarily less than or equal to the value given by Equation (2)). The cost of each indispensable action a_i, for which we used the pessimistic estimate C_{min} in Equation (2), can now be replaced by $cost(a_i)$. (We omit the formula for $MaxLev$ since it is a special case of a more general formula given below in Equation (3)).

7.2. Indispensable sets of actions

Some problems have no indispensable actions. Nevertheless, we can often still detect that any plan must contain actions of cost greater than C_{min}. This can be achieved by generalizing Definition 7.1 to a set of actions.

Definition 7.3 (indispensable set) *A set S of actions is an indispensable set in a problem $\Pi = \langle A, I, G\rangle$ if each solution to Π contains at least one of the actions in S.*

In independent work, Helmert & Domshlak [39] show how indispensable sets (which they call disjunctive action landmarks) can be used to calculate a new admissible heuristic for optimal-cost planning.

We denote by $A^+_{\geq} = \{a_1, a_2, \ldots, a_n\}$ the set of actions occurring in the reduced relaxed graph of a problem $\Pi = \langle A, I, G\rangle$, in decreasing order of cost: $\forall\ j \in \{1, .., n\}, a_j \in A^+_r$ and $\forall\ j \in \{1, .., n-1\}, cost(a_j) \geq cost(a_{j+1})$. We denote by mincost$(X)$ the minimum cost of a set X of actions: mincost$(X) = \min\{cost(a) : a \in X\}$.

Lemma 7.4 (indispensable set of costliest actions) *Let $A^i = \{a_1, a_2, \ldots, a_i\}$ be the subset of $A^+_{\geq}$ built incrementally starting with $\{a_1\}$ until it contains an action a_i such that:*

- *$\Pi^+_{i-1} = \langle A^+_{\geq} - \{a_1, \ldots, a_{i-1}\}, I, G\rangle$ has a solution,*
- *$\Pi^+_i = \langle A^+_{\geq} - \{a_1, \ldots, a_{i-1}, a_i\}, I, G\rangle$ has no solution.*

Then A^i is an indispensable set.

For every problem, there is always an indispensable set of costliest actions: in the worst case, this set is $A^+_{\geq} = \{a_1, a_2, \ldots, a_n\}$ where a_n has the minimum cost C_{min}. The set of actions which establish a goal fluent is also an indispensable set.

Let add_f denote the set of actions $a \in A$ such that $f \in add(a)$. Let $prec_f$ denote the set of actions $a \in A$ such that $f \in prec(a)$.

Lemma 7.5 (goal-based indispensable set) *If g is a fluent of the goal G, then add_g is an indispensable set.*

For an indispensable action a to be applicable, some other actions must establish those preconditions of a which were not present in the initial state. Furthermore, for an action a satisfying $add(a) \cap G = \emptyset$ to be indispensable, at least one fluent of $add(a)$ must be used by some other action. These observations allow us to detect new indispensable sets based on indispensable actions which have already been discovered.

Lemma 7.6 (precondition-based indispensable sets) *If f is a precondition of an indispensable action such that $f \notin I$, then add_f is an indispensable set.*

Lemma 7.7 (add-based indispensable set) *Let a be an indispensable action such that $add(a) \cap G = \emptyset$. If $X_a \subseteq A$ is the set of actions $a' \in A$ such that $add(a) \cap prec(a') \neq \emptyset$, then X_a is an indispensable set.*

A fluent f is known as a landmark if f is true at some point in all solution plans [63]. Landmarks have been used in non-optimal planning in order to partition the problem into easier subproblems. Some landmarks can be identified in polynomial time using the relaxed planning graph. There are strong links between landmarks and certain indispensable sets: a fluent f is a landmark if the set of actions having f as a precondition is an indispensable set; a fluent $f \notin G$ is a landmark if the set $\{a \in A : add(a) = \{f\}\}$ is an indispensable set.

We can detect certain landmarks from analysis of the relaxed planning graph [63] and then deduce indispensable sets from the following lemma (which can be seen as a generalization of Lemma 7.5).

Lemma 7.8 (landmark-adding indispensable set) *If f is a landmark fluent such that $f \notin I$, then add_f is an indispensable set.*

Let $A^+_r[i]$ denote the set of actions appearing in the first i levels of the reduced relaxed planning graph G^+_r. Let A_i be the set of actions, except noops, appearing in level i of the planning graph G. If the planning graph has actually been constructed up to level k, then let:

$$A'_i = \begin{cases} A_i \cap A^+_r[i] & \text{for i=1,...,k} \\ A^+_r[i] & \text{for } i > k \end{cases}$$

Let $\overline{A}_i$ denote the set of all actions which appear in level i of some valid parallel plan. Such actions necessarily appear in A_i and also at a level $j \leq i$ of the reduced relaxed graph. Therefore, $\forall i \geq 1$, $\overline{A}_i \subseteq A'_i$. An optimal plan must contain at least one action (which is not a noop) at each level, otherwise there would be an optimal plan with fewer levels. The following lemma follows immediately.

Lemma 7.9 (level-based indispensable set) *For $i \geq 1$, the set of actions A'_i, as defined above, is an indispensable set.*

We can make the level-based indispensable sets A'_i non-intersecting by making multiple copies of actions (one per level).

Indispensable sets have recently been used to calculate an admissible heuristic in cost-optimal planning [44,39]. Indispensable sets can also be used in IP-based planning [71]. For example, knowing that X is an indispensable set allows us to add the extra inequality constraint $\sum_{a\in X} n_a \geq 1$, where n_a represents the total number of occurrences of action a in a solution plan.

7.3. Using indispensable sets to improve the bound

Let $\mathcal{X}$ be a set of non-intersecting indispensable sets, i.e. $\forall\ X, Y \in \mathcal{X}, X \cap Y = \emptyset$. For simplicity of notation, we consider indispensable actions as singleton indispensable sets. We can now give an improved estimate of $MaxLev$:

$$MaxLev = |\mathcal{X}| - 1 + \left\lceil \frac{C^*_k - \sum_{X\in\mathcal{X}} \text{mincost}(X)}{C_{min}} \right\rceil \quad (3)$$

In our example, $\{move_{AB}, move_{BA}, move_{DE}, move_{ED}, move_{BC}, move_{CB}\}$ is the indispensable set of costliest actions; $\{unload_B\}$ is a goal-based indispensable set; $\{move_{AB}, move_{CB}, move_{EB}\}$ is an indispensable set derived from the preconditions of the indispensable action $unload_B$; $\{unload_A, unload_B, unload_C\}$ is an indispensable set derived from the adds of $load_A$. $\{load_A, load_B, load_C\}$ and $\{unload_A, unload_B, unload_C\}$ are indispensable sets derived from the landmark c_v. The cheapest action $unload_B$ is only present from level 3 onwards in the reduced relaxed graph, which allows us to deduce two level-based indispensable sets, at levels 1 and 2, in which the minimum-cost action is (a copy of) $load_A$ of cost 5. For example, $A'_1 = \{move_{AB}, load_A, move_{AC}\}$. Several sets of non-intersecting indispensable sets can be found: the one which provides the best upper bound $MaxLev$ is $\mathcal{X} = \{\{move_{AB}, move_{BA}, move_{DE}, move_{ED}, move_{BC}, move_{CB}\}\}$.

Only indispensable sets which have no action in common can be used to improve the upper bound $MaxLev$ of Equation (3). In our trials we constructed $\mathcal{X}$ using a greedy algorithm: an indispensable set X (deduced from Lemmas 7.2, 7.4, 7.5, 7.6, 7.7, 7.8 and 7.9) was added to $\mathcal{X}$ iff X had an empty intersection with the sets already added to $\mathcal{X}$. We used a greedy algorithm, since, as we will now show, choosing a set of non-intersecting indispensable sets so as to minimize the value of $MaxLev$ is NP-hard. We first give a formal definition of this problem.

Problem: OCIS (Optimal Choice of Indispensable Sets)
Instance: An optimal-planning problem $\Pi = \langle A, I, G\rangle$, with C_{min} the minimum cost of actions in A, a set J of indispensable sets ($\forall X \in J$, $X \subseteq A$ and X is an indispensable set of actions) and a constant L.
Question: Does there exist a subset $\mathcal{X} \subseteq J$ such that $\forall X, Y \in \mathcal{X}$, $X \cap Y = \emptyset$ and

$$\Sigma_{X\in\mathcal{X}}(\text{mincost}(X) - C_{min}) \ \geq\ L$$

An optimal choice $\mathcal{X}$ of indispensable sets maximizes $\Sigma_{X\in\mathcal{X}}(\text{mincost}(X) - C_{min})$, which is equivalent to minimizing the upper bound $MaxLev$ given by Equation (3). The proof of the following theorem is given in the Appendix.

Theorem 7.10 *OCIS is NP-hard.*

Let OCIS(max) denote the problem of finding the maximum value of L for which OCIS has a solution. A solution to OCIS(max) is a lower bound on the cost of a plan for the problem Π^- which is identical to Π except that the cost of each action has been reduced by C_{min}. OCIS(max) is equivalent to finding an assignment of actions to indispensable sets which maximizes $\sum_{X\in J} cost'(X)$ where $cost'(X) = \text{mincost}(X) - C_{min}$ if all actions $a \in X$ are assigned to X, and $cost'(X) = 0$ otherwise. We can express this as an integer program with variables $z_{aX} \in \{0, 1\}$ ($a \in X, X \in J$), with $z_{aX} = 1$ signifying that action a is assigned to the indispensable set X, together with the following inequalities:

$$\forall a \in A, \sum_{X\in J\ s.t.\ a\in X} z_{aX} = 1$$

$$\forall X \in J,\ \forall a \in X,\quad cost'(X) \leq (cost(a) - C_{min}) * z_{aX}$$

Although this integer program is NP-hard (by Theorem 7.10), its linear relaxation can be solved in polynomial time and remarkably also provides a (possibly) better admissible lower bound for Π^- [44] and hence a better upper bound on the number of levels. A greedy algorithm can be used to approximate the optimal solution to this linear relaxation [39].

8. Theoretical considerations

8.1. Improving the upper bound

It should be observed that an even better bound $MaxLev$ can sometimes be deduced than that given by the optimal solution to the problem OCIS. Consider the two overlapping indispensable sets $X_1 = \{move_{AB}, move_{CB}, move_{EB}\}$ and $X_2 = \{move_{AB}, load_A, move_{AC}\}$: the optimal choice of non-intersecting indispensable sets is $\mathcal{X} = \{X_1\}$, with $\Sigma_{X\in\mathcal{X}}(\text{mincost}(X) - C_{min}) = \text{mincost}(X_1) - 3 = 12$. However, it is easy to show by exhaustion that any set of actions S containing at least one element from each of X_1 and X_2 satisfies $\Sigma_{a\in S}(\text{cost}(a) - C_{min}) \geq 14$. The formal definition of the problem of finding the best bound $MaxLev$ in this way is given below. We again consider indispensable actions as singleton indispensable sets.

Problem: OCSA (Optimal Choice of Set of Actions)

Instance: An optimal-planning problem $\Pi = \langle A, I, G\rangle$, with C_{min} the minimum cost of actions in A, a set J of indispensable sets and a constant M.

Question: Does there exist a subset $S \subseteq A$ such that $\forall X \in J$, $S \cap X \neq \emptyset$ and

$$\Sigma_{a\in S}(\text{cost}(a) - C_{min}) \;\leq\; M$$

If S is an optimal choice of set of actions, i.e. a set of actions which minimizes $M_S = \Sigma_{a\in S}(\text{cost}(a) - C_{min})$, then we can deduce an upper bound $MaxLev = \left\lceil \frac{C^*_k - M_S}{C_{min}} \right\rceil - 1$. Not surprisingly, OCSA is also intractable.

Proposition 8.1 *OCSA is NP-hard.*

This follows by a simple polynomial reduction from Hitting Set; it suffices to show that any collection of sets of actions can actually occur as indispensable sets in a planning problem. Given any hitting set instance, for each of its sets $X = \{a_1, \ldots, a_k\}$, we simply insert a new goal g_i and let actions $a_1, \ldots, a_k$ be the only achievers of g_i, meaning that X is an indispensable set.

Indispensable sets can also be used to determine a lower bound on the cost of a solution plan, an essential ingredient of heuristic search. OCSA is equivalent to determining the best such lower bound from a set of indispensable sets in a planning problem in which the cost of each action a is $\text{cost}(a) - C_{\min}$. Hence, it follows from Proposition 8.1 that finding the best lower bound from a set of indispensable sets is NP-hard.

8.2. Generating new indispensable sets from old

GP-WCSP* uses indispensable sets of actions to improve its estimate of the maximum number of levels that need to be constructed in the planning graph in order to guarantee optimality. In our implementation we employ several distinct sources of information to generate indispensable sets: relaxed versions of the planning problem, indispensable actions and landmarks (both generated from relaxed versions of the planning problem), and the partially constructed planning graph. The number of indispensable sets we generate is $O(n_a + n_f + n_l)$, where n_a is the number of actions, n_f the number of fluents and n_l the number of levels in the planning graph. We do not attempt to generate all indispensable sets, because, in the worst case, their number may be exponential, as we will demonstrate below. However, for theoretical completeness, we now present two lemmas which would, at least in theory, allow us to generate an exponential number of indispensable sets.

Consider, first of all, a simple planning problem consisting of transporting a crate between two cities A and B. There are m different, non-intersecting routes from A to B, each of which is composed of n distinct steps. There are thus exactly m different valid plans, each of which is a sequential plan of length n. The number of minimal indispensable sets is easily seen to be n^m, corresponding to the number of ways of choosing one step from each of the m different routes.

In this particular example, we can, in fact, generate all indispensable sets from the single goal-based indispensable set (Lemma 7.5) by multiple applications of the following lemma.

Lemma 8.2 *Let X be an indispensable set and consider any $a \in X$. Let f be a precondition of a such that $f \notin I$. If add_f is the set of actions $a' \in A$ such that $f \in add(a')$, then $(X - \{a\}) \cup add_f$ is an indispensable set.*

Proof: Suppose, for a contradiction, that P is a valid plan which uses no actions in $(X - \{a\}) \cup add_f$. Since X is an indispensable set, P must use action a. But, since $f \in prec(a)$, $f \notin I$ and P does not use any of the actions $a' \in add_f$ which achieve f, P cannot use action a. ∎

Using a similar argument we can also prove the following lemma.

Lemma 8.3 *Let X be an indispensable set and consider any $a \in X$ such that $add(a) \cap G = \emptyset$. If $X_a \subseteq A$ is the set of actions $a' \in A$ such that $add(a) \cap prec(a') \neq \emptyset$, then $(X - \{a\}) \cup X_a$ is an indispensable set.*

Lemmas 7.6 and 7.7 correspond to the special cases of Lemmas 8.2 and 8.3, respectively, in which $X = \{a\}$.

9. Eliminating actions

We can also detect, during search, actions which have become too costly because any plan containing these actions would have a cost greater than or equal to the cost of the current best plan. Such eliminations reduce the size of the search space. They can also allow us to recalculate, during search, the importance of actions and hence to further reduce the value of $MaxLev$.

Definition 9.1 (too-costly action) *An action a is too costly at level k and at all levels $k' \geq k$ if:*

$$(k-1) * C_{min} + cost(a) \geq C^*_{k-1} \quad (4)$$

Taking into account indispensable sets, the action a is too costly from level k onwards if:

$$(k - |\mathcal{X}| - 1) * C_{min} + \ cost(a) \ + \ \Sigma_{X \in \mathcal{X}, a \notin X} \ mincost(X) \ \geq C^*_{k-1} \quad (5)$$

where $\mathcal{X}$ is a set of non-intersecting indispensable sets.

The elimination of a too-costly action may render other (sets of) actions indispensable. This, in turn, may render other actions too costly. This process clearly converges in polynomial time and may lead to a reduction in the value of $MaxLev$.

In our example, Equations (4) and (5) allow us to deduce that $move_{AB}$ is too costly from level 4 onwards. Once this action has been eliminated, $move_{AC}$ becomes an indispensable action. The optimal length-4 plan P^*_4 and Equation (5) allow us to deduce that all the actions not contained in P^*_4 are too costly from level 5 onwards.

10. Experiments

In this section we report results of experimental trials. We carried out a large number of trials on different benchmark problems from the IPC (International Planning Competition[1]) covering diverse application areas, to which we added a random integer cost in the range 1 to 20 for each action. (Trials on the same problems but with random costs in the range 21 to 40 are also reported in Section 10.3.2.) We used the Toulbar2 library[2] [18] to solve the WCSP derived from the planning graph. Our experimental trials were performed on a 3GHz Pentium IV with 1Gbyte of memory.

10.1. Description of the benchmarks

Some IPC benchmarks are inspired by real-world problems and are highly structured [41]. For each different domain, there is a series of problems of increasing difficulty, this being a function of the number of available actions and the number of fluents in the initial and goal states.

We used nontemporal domains from the IPC, written in PDDL (without ADL conditions), to which we added a random integer cost in the range 1 to 20 for each action. However, certain domains (Airport, Pipesworld, Promela, PSR, Pathways, Tpp, Trucks, Openstacks, Elevators) consist of problems whose size (in terms of number of actions and fluents or the length of solution-plans) implies the construction of a planning graph and then a WCSP which is too costly in memory for our approach. We present results only for those domains in which GP-WCSP solved at least one problem.

The BlocksWorld domain is more constrained than the other domains: given an initial set of towers of blocks and a single arm, the goal is the construction of another set of towers.

The Logistics domain involves transporting packages between locations using planes (between cities) and trucks (within a city).

In the Depots domain, trucks can transport crates around and then the crates must be stacked onto pallets at their destinations. The stacking is achieved using hoists.

[1] http://zeus.ing.unibs.it/ipc-5/

[2] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro

The DriverLog domain involves driving trucks to deliver packages. The trucks require drivers who must walk between trucks in order to drive them. The paths for walking and the roads for driving are given in the form of two different maps.

The Satellite domain is inspired by space applications. It involves planning and scheduling a collection of observation tasks between multiple satellites, each equipped in slightly different ways.

The ZenoTravel domain involves transporting people around in planes, using different modes of movement (fast and slow).

The Rovers domain involves a collection of planetary rovers which navigate on the surface of a planet, finding samples and taking them back to a lander.

The Storage domain involves spatial reasoning. It is concerned with moving a certain number of crates from containers to depots by hoists. Inside a depot, each hoist can move according to a specified spatial map connecting different areas of the depot.

We also used the Scanalyzer-3D and Transport domains from the sequential optimization track of IPC2008.

10.2. The best plan of fixed length

This section describes results of trials of GP-WCSP which finds a cost-optimal plan among parallel plans of minimum length. These trials allowed us to compare different techniques for solving WCSPs, independently of the different methods for calculating $MaxLev$. It is worth noting that the optimal parallel-plan of minimum length was often the globally optimal plan and was found in CPU times ranging from 0.01 seconds to 5019 seconds with a median value of 11.38 seconds. In 90% of the benchmark problems tested, the resolution time of each WCSP built by GP-WCSP is less than 1 second.

Resolution time is an exponential function of parallel-plan length which means, for example, that feeding in the minimum length k by hand would not significantly speed up the search for an optimal parallel-plan of minimum length. To solve the corresponding WCSP it is necessary not only to find an optimal solution but also to prove optimality; hence the fact that the WCSP corresponding to length k is solvable does not make it easier to solve. The total time spent on solving unsolvable WCSPs represents only about 25% of the total time spent on solving WCSPs (i.e. (k-1) unsolvable WCSPs and 1 solvable WCSP) in GP-WCSP. In none of the benchmark problems tested did thus percentage exceed 67%.

We tested different heuristics and algorithms on the set of WCSPs constructed by GP-WCSP* during the search for an optimal solution to different problems from the benchmark domains[3] described above. In order to get a clearer idea of the size of the WCSPs which we are able to solve, in this set of trials we avoided using the cost of an optimal level-k plan to prune the search for an optimal level-$(k+1)$ plan. Table 1 describes the problems and the resulting WCSPs: problem name, characteristics of the largest WCSP solved by GP-WCSP* (number of planning-graph levels, number of variables, geometric mean of domain sizes, number of constraints, search space size of the WCSP encoding).

10.2.1. Variable-ordering heuristics

In preliminary experimental trials we compared different variable-ordering heuristics. It is well known that the order of instantiation of variables can greatly affect the efficiency of search. It should also be noted that the lower bound produced by FDAC at every node of the search tree also depends on the order of the uninstantiated variables [13]. In our trials, the instantiation order of variables was recalculated at each node of the search tree and the calculation of the lower bound by FDAC used the current instantiation order.

We tested several different variable-ordering heuristics, starting with simple heuristics based on a random ordering, the number of constraints or the domain size. We present the comparison between the two variable-ordering heuristics which produced the best results in our tests. Figure 6 presents a comparison in terms of the time to solve WCSPs to optimality between the use of a planning-graph based heuristic and a WCSP-derived heuristic:

– the "level-based" heuristic has been found to perform well on planning benchmarks (without costs): it consists in ordering variables by increasing domain size, with ties being broken according to the inverse order of their appear-

[3] the corresponding WCSP files are available here: http://mulcyber.toulouse.inra.fr/gf/project/toolbar/

Problem	characteristics of the largest WCSP				
	k	Variables	$mean\ d_i$	Constraints	Search space
blocks01	6	179	2.74	2148	2.04E+078
blocks02	20	747	2.82	8160	3.06E+336
blocks03	6	147	2.70	1483	2.96E+063
blocks04	20	1157	2.78	16579	8.69E+513
blocks05	20	1273	2.79	19033	5.23E+566
blocks06	20	1023	2.77	14071	6.63E+452
blocks07	20	1703	2.74	30576	9.01E+744
blocks08	10	781	2.72	14457	6.07E+338
blocks09	20	1287	2.73	20546	1.50E+561
blocks10	20	1515	2.70	27279	1.95E+652
logistics01	13	265	3.16	2941	2.47E+132
logistics02	12	245	3.14	2624	5.05E+121
logistics03	15	330	3.21	3796	2.00E+167
logistics04	12	284	3.13	3353	5.31E+140
logistics05	12	286	3.15	3342	2.87E+142
logistics06	7	98	2.76	646	1.81E+043
storage01	3	11	2.02	23	2.30E+003
storage02	3	11	2.02	23	2.30E+003
storage03	6	118	2.57	981	2.31E+048
driverlog01	16	495	3.59	7871	3.27E+274
driverlog02	12	454	3.68	8554	8.13E+256
driverlog03	13	490	3.68	9664	1.82E+277
driverlog04	11	540	3.67	12396	8.21E+304
driverlog07	9	601	3.77	18219	1.04E+346
driverlog08	9	624	3.74	20113	1.32E+357
zenotravel01	2	10	2.00	20	1.02E+003
zenotravel02	17	1100	2.71	9046	5.50E+256
zenotravel03	6	476	2.79	4572	1.10E+212
satellite01	20	235	5.17	2994	4.91E+167
satellite02	20	326	5.31	5268	2.65E+236
satellite03	16	417	5.86	6725	8.56E+319
satellite04	13	396	6.58	7797	6.82E+323
depot01	16	818	3.05	12675	1.01E+395
depot02	13	1323	2.97	29518	4.47E+058

Table 1

Details of the problems used in our initial trials.

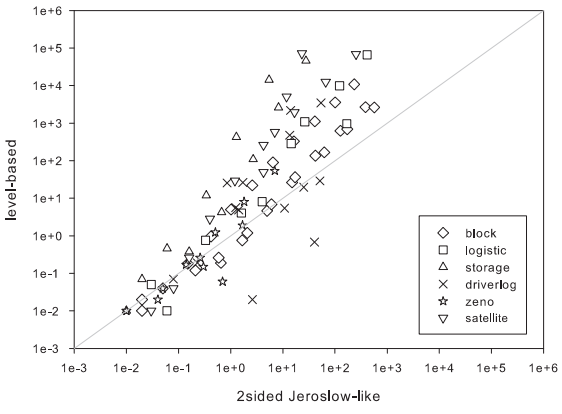

Fig. 6. Comparison of WCSP resolution times for the heuristics: "2-sided Jeroslow-like" and "level-based" (in seconds).

ance in the planning graph. Cayrol et al. [9] and Do & Kambhampati [22] showed that the CSP variable-ordering heuristic "most constrained variable first" outperformed the original GRAPHPLAN "no-op" first heuristic and the domain size heuristic. Fluents which first occur in higher levels of the planning graph tend to be more difficult to obtain and hence the level at which a fluent first appears is a good measure of constrainedness;

– the "2-sided Jeroslow-like" heuristic [19] has been found to perform well on WCSP benchmarks: variables are ordered in increasing size of the ratio between domain size and the sum of the average costs of the unary and binary constraints involving the variable.

Figure 6 clearly shows, on a logarithmic scale, that the "2-sided Jeroslow-like" heuristic outperforms the "level-based" heuristic for solving the

WCSPs derived from planning benchmark problems. For example, the WCSP derived from the 15-level planning graph corresponding to a problem from the Satellite domain was solved to optimality in 23.18 seconds using the "2-sided Jeroslow-like" heuristic compared to 19h49min46sec for the "level-based" heuristic.

We also compared different ordering heuristics for the instantiation of the values within a domain. In all the problems tested, no significant difference was noted between the computation times using the different value-ordering heuristics.

10.2.2. Soft constraint propagation

We tested different soft arc consistency algorithms (NC, FDAC and EDAC described in Section 5). NC, FDAC or EDAC was established at each node of the branch-and-bound search.

Comparing NC and FDAC (Figure 7), we observed that FDAC was on average 17 times faster than NC. The number of nodes visited is always less with FDAC (by a factor of up to 250 times) and, on average, FDAC visits 35 times less nodes than NC. We can conclude that the use of FDAC significantly improves the time to extract an optimal solution from the planning graph coded as a WCSP.

Figure 8 provides a comparison between FDAC and EDAC. We found no significant difference between the two techniques. On average, EDAC visited 5% less nodes but used 6% more CPU time. We can therefore conclude that EDAC, unlike on random problems of similar density [18], does not provide an improvement compared with FDAC on the problems tested.

We also experimented with more sophisticated search algorithms. Since the pathwidth of the constraint graph corresponding to the WCSP was often large, we could not take advantage of tree-decomposition techniques. The recently developed form of soft arc consistency, called virtual arc consistency (VAC) [15], is theoretically stronger than FDAC, but requires greater computation time. It is therefore best suited for solving hard WCSP instances. Maintaining VAC during search did not produce a significant improvement, compared with the simpler propagation technique FDAC, on the optimal-planning problems we tested. The extra work performed at each node of the branch-and-bound search tree was not always compensated by the reduction in the number of nodes visited.

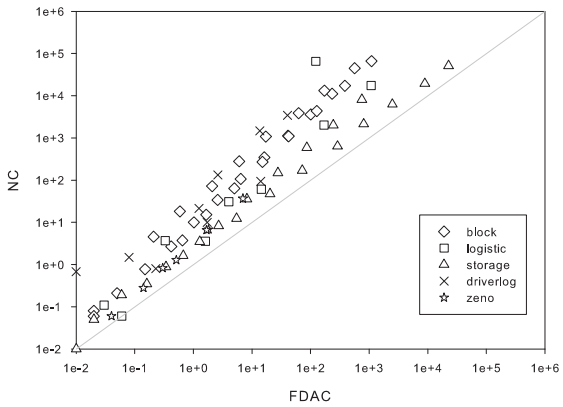

Fig. 7. Comparison of WCSP resolution times between using FDAC or NC (in seconds).

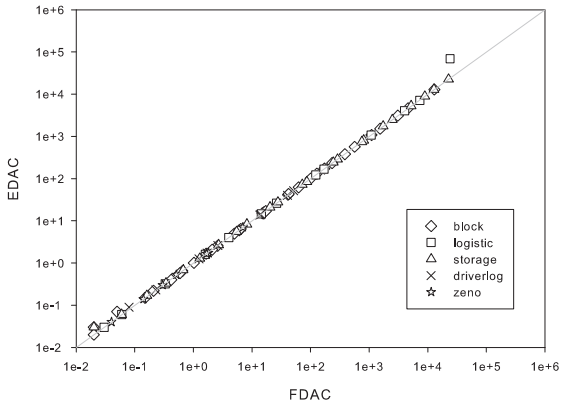

Fig. 8. Comparison of WCSP resolution times between using FDAC or EDAC (in seconds).

Notice that the size of the WCSPs solved (Table 1) is an order of magnitude larger than the largest random WCSPs solved to date using the same algorithms [18] (40 variables, domain-size 10). For example, the search space of the largest WCSP derived from problem *blocks*07 is about 9×10^{744}. The fact that we were able to solve such large WCSPs is no doubt due to the structure of the WCSP derived from the planning graph [14].

10.2.3. GP-WCSP

In this section we report results of the experimental trials of GP-WCSP. Table 2 shows the problems which GP-WCSP could solve within a time limit of 27 hours for the resolution of each WCSP. We used the "2-sided Jeroslow-like"

variable-ordering heuristic and maintained FDAC during search for each WCSP that was solved. The Solution columns describe the cost-optimal plan among parallel-plans of minimum length: the number of levels (k) and the number of actions (N_a). The CPU time is the total execution time. The notation $h : m : s$ stands for hours, minutes, and seconds respectively. Otherwise, times are in seconds.

The first conclusion we can draw from the experimental results, given in Table 2, is that the number of problems solved is highest in the domains with the most constrained problems. For example, GP-WCSP solves the first 15 problems of the very constrained Blocks domain (and 13 problems from the Logistic domain) but only 4 problems from the Depots domain. Indeed, a planning graph with a larger number of mutexes gives rise to a WCSP with many crisp constraints which allows pruning to occur early in the branch-and-bound search for an optimal solution. Another general conclusion that came out of these experimental trials was the general rule that GP-WCSP is capable, in most cases, of solving problems containing up to 250 actions and requiring the construction of a planning graph containing up to about 20 levels. Beyond these limits, the graph generally becomes too large to be stored and/or translated into a WCSP which can be solved in less than 27 hours.

10.3. Globally optimal plan

In this section we report results of trials of GP-WCSP* and we compare our approach with the results obtained by other optimal planners. In order to find a globally optimal plan, GP-WCSP* solves a WCSP, whose solution is an optimal length-k parallel-plan, for increasing values of k. This exhaustive search terminates when the number of levels k exceeds the upper bound $MaxLev$. Given the results reported in Section 10.2, we used the "2-sided Jeroslow-like" variable-ordering heuristic and maintained FDAC during search for each WCSP that was solved. The value of $MaxLev$ was calculated using Equation (3). Indispensable sets of actions were detected and a greedy algorithm used to find a set of non-intersecting indispensable sets. At each level k, too-costly actions were detected and indispensable sets recalculated. This dynamic recalculation often led to a dramatic decrease in $MaxLev$.

We used the Toulbar2 library to solve the WCSPs, maintaining FDAC at each node of the branch-and-bound search tree. We computed the set J of all indispensable sets that could be deduced from Lemmas 7.2, 7.4, 7.5, 7.6, 7.7, 7.8 and 7.9. The elements X of J were ordered in decreasing order of mincost(X) and, in case of ties, by increasing order of $|X|$. The set $\mathcal{X}$ of non-intersecting indispensable sets was then built by a greedy algorithm which, thanks to the ordering of J, gives preference to indispensable sets J with the highest minimum cost. It is easy to see that this greedy algorithm includes all singleton indispensable sets in $\mathcal{X}$.

*10.3.1. GP-WCSP**

The results of the experimental trials of GP-WCSP* are given in Table 3. The initial value of $MaxLev$ is calculated using Equation (2), in which all actions are assumed to have a cost of C_{min}, and the final value is calculated using Equation (3), after the detection of indispensable sets. $MaxLev$ is calculated at each level and can decrease from one level to the next. We give the final, and hence the smallest value. N_{ia} is the number of indispensable actions detected by Lemma 7.2 and after the elimination of too-costly actions; $|\mathcal{X}|$ is the number of non-intersecting indispensable sets in the set $\mathcal{X}$ constructed by our greedy algorithm; N_{tc} is the number of too-costly actions detected. The Solution columns describe the best solution-plan: the level of optimality achieved (plan), the number of levels (k) and the number of actions (N_a). The CPU time is the total execution time. The notation $h : m : s$ stands for hours, minutes, and seconds respectively. Otherwise, times are in seconds. If the best plan found is P^* then we have a proof of global optimality. If the best plan found is P^*_k, for some k, then global optimality is not guaranteed, but the plan is nevertheless an optimal plan among parallel plans containing up to k levels.

As already observed in Section 10.2, the WCSP instances that we are able to solve are very significantly larger than the largest random WCSP instances that have been solved to optimality [57,18]. We were able to solve the WCSP instances derived from planning graphs with up to 20 levels. Furthermore, using the cost of the best plan found so far up to level $i-1$, drastically reduces the time to solve the WCSP at level i: for example, the WCSP derived from the reduced 20-level planning graph in *blocks*05 has a search-space size of 5×10^{566} (geo-

Problem	Solution		CPU
	k	N_a	time
blocks01	6	6	11.04
blocks02	10	10	4.45
blocks03	6	6	3.15
blocks04	12	12	0:02:31
blocks05	10	10	0:02:21
blocks06	16	16	0:06:57
blocks07	12	12	0:15:01
blocks08	10	10	0:19:51
blocks09	20	20	0:52:50
blocks10	20	20	1:24:23
blocks11	22	22	25:28:16
blocks12	20	20	06:21:21
blocks13	18	18	15:06:58
blocks14	20	20	31:56:20
blocks15	16	16	07:25:55
logistics01	9	20	1
logistics02	9	19	2.03
logistics03	9	15	1
logistics04	9	27	1.63
logistics05	9	17	6.72
logistics06	3	8	0.05
logistics07	9	25	3.21
logistics08	9	14	3.96
logistics09	9	26	6.8
logistics10	12	38	0:46:57

Problem	Solution		CPU
	k	N_a	time
logistics12	11	31	1:30:08
logistics14	11	36	0:14:24
logistics15	10	31	0:34:52
depot01	5	11	0.18
depot02	8	16	0:01:11
depot03	12	29	3:16:45
depot07	10	24	2:37:34
driverlog01	6	8	0.15
driverlog02	7	18	12.99
driverlog03	7	12	11.58
driverlog04	7	19	0:01:02
driverlog05	8	21	0:02:49
driverlog06	5	11	6.92
driverlog07	6	18	0:01:46
driverlog08	7	27	0:05:46
driverlog09	10	24	0:31:56
rovers01	6	10	9.87
rovers02	4	8	0.53
rovers03	7	12	18.54
rovers04	4	8	0.33
rovers05	8	23	0:09:31
satellite01	8	9	9.55
satellite02	12	13	0:02:06
satellite03	6	13	32.69
satellite04	10	22	0:10:44

Problem	Solution		CPU
	k	N_a	time
satellite05	7	23	0:16:13
satellite06	8	27	0:58:11
satellite07	6	28	0:22:54
satellite09	6	35	4:13:28
zenotravel01	1	1	0.04
zenotravel02	5	6	4.19
zenotravel03	5	6	0:01:02
zenotravel04	5	12	31.87
zenotravel05	5	14	0:01:36
zenotravel06	5	12	0:02:27
zenotravel07	6	16	0:01:43
storage01	3	3	0.01
storage02	3	3	0.02
storage03	3	3	0.04
storage04	8	8	12.44
storage05	6	9	3.32
storage06	6	9	9.7
storage07	14	14	0:46:13
storage08	8	12	0:11:29
scanalyzer01	2	6	5.09
scanalyzer22	5	5	0.91
scanalyzer23	5	5	0.93
scanalyzer24	5	5	0.91
transport01	4	7	0.21
transport02	8	15	0:06:42
transport11	7	13	2.77

Table 2
Results of the experimental trials of GP-WCSP.

metric mean of domain sizes: 2.79; number of variables: 1273; number of constraints: 19005). It was solved in 0.58 seconds when the cost of the best plan found so far was passed as a parameter and in 12,885 seconds without using this parameter.

The results reported in Table 3 demonstrate that the use of WCSPs to find cost-optimal plans is a practical possibility. However, we should point out that GP-WCSP* performs better on some domains than others. In the Scanalyzer domain and the first three problems in the Storage domain (which contains few actions), GP-WCSP* quickly determines an optimal solution. The domains which contain a lot of indispensable actions, such as BlocksWorld or Logistics, also represent a favourable case, even if the last WCSP was often too large to be solved in under 27 hours, and hence a proof of optimality was not obtained. In the Depots, Driverlog, Satellite, ZenoTravel and Transport domains, GP-WCSP* returns the optimal plan but again the value of $MaxLev$ calculated by GP-WCSP* was rarely sufficiently low to provide a guarantee of optimality. For the benchmark problems tested, the optimal plans often correspond to the first plan found by GP-WCSP*.

The value of $MaxLev$ decreased by a significant amount in all the problems tested and, on average, by a factor of 55.6%. This decrease is due to the indispensable sets of actions that were found in all the problems tested. Indispensable actions play a key role in reducing $MaxLev$: an average of 41.7% of the actions in the best plans found were indispensable actions that were rapidly detected by analysis of the relaxed planning graph. For example, in the *logistics*04 problem, we found 25 indispensable actions out of the 27 actions in the level-12 optimal plan.

When too-costly actions exist, they help considerably in reducing the size of the search space. For example, during the resolution of the *logistics*06 problem, 70 too-costly actions were detected. The total time spent on the detection of indispensable actions and too-costly actions, during the resolution of each of the 55 problems in Table 3, was on average only 0.1 seconds (and never more than 0.5 seconds). These computation times allow us to envisage the detection of indispensable actions and too-costly actions in other algorithms for cost-optimal planning.

It is also possible to use indispensable sets during the extraction of an optimal parallel plan of a

Problem	MaxLev					Solution			CPU
	init.	final	N_{ia}	$\lvert\mathcal{X}\rvert$	N_{tc}	plan	k	N_a	time
blocks01	51	5	6	6	0	P^*	6	6	11.17
blocks02	61	31	6	6	10	P^*_{20}	10	10	0:10:12
blocks03	37	5	6	6	0	P^*	6	6	3.15
blocks04	96	50	8	8	0	P^*_{20}	12	12	0:47:36
blocks05	93	21	7	7	37	P^*_{20}	10	10	1:21:09
blocks06	111	73	9	9	0	P^*_{20}	16	16	0:25:19
blocks07	84	26	11	11	34	P^*_{20}	12	12	2:57:45
blocks08	132	9	10	10	0	P^*	10	10	0:19:44
blocks09	139	88	11	11	0	P^*_{20}	20	20	0:51:54
blocks10	145	71	13	13	0	P^*_{20}	20	20	1:23:08
logistics01	148	21	19	19	27	P^*_{13}	9	20	0:56:52
logistics02	138	27	17	17	13	P^*_{12}	9	19	0:59:01
logistics03	145	34	13	13	5	P^*_{15}	9	15	2:26:05
logistics04	234	46	25	25	0	P^*_{12}	9	27	2:23:47
logistics05	128	21	15	15	26	P^*_{12}	9	17	2:03:15
logistics06	51	7	8	8	70	P^*	3	8	0.92
logistics07	178	44	23	23	0	P^*_{12}	9	25	2:54:25
logistics08	123	30	13	13	7	P^*_{13}	9	14	8:38:32
logistics09	245	37	23	23	0	P^*_{12}	11	25	25:46:43
logistics10	404	64	31	33	0	P^*_{13}	13	36	5:34:09
depot01	93	33	4	8	0	P^*_{16}	8	10	11:35:13
depot02	123	73	5	6	0	P^*_{13}	11	17	9:05:22
driverlog01	60	41	0	3	0	P^*_{16}	6	8	18:54:12
driverlog02	141	121	0	9	0	P^*_{12}	7	18	3:21:26
driverlog03	110	86	0	6	0	P^*_{13}	7	12	3:55:33
driverlog04	179	112	0	11	0	P^*_{11}	10	21	2:48:22
driverlog05	144	124	0	6	0	P^*_{11}	8	21	3:06:13
driverlog06	106	78	0	8	0	P^*_{9}	5	11	3:37:47
driverlog07	146	88	0	6	0	P^*_{9}	8	18	4:45:11
driverlog08	224	165	0	14	0	P^*_{9}	9	27	6:41:08
rovers01	82	41	3	7	0	P^*_{12}	5	9	7:25:29
rovers02	82	41	3	7	0	P^*_{12}	5	9	7:22:06
rovers03	112	40	3	9	0	P^*_{12}	8	11	3:31:20
rovers04	63	37	2	8	0	P^*_{11}	5	8	31:59:42
rovers05	141	75	7	14	0	P^*_{9}	9	23	1:50:38
satellite01	60	23	5	8	22	P^*_{20}	8	9	1:18:41
satellite02	80	28	4	13	17	P^*_{20}	15	16	1:36:32
satellite03	128	61	0	4	0	P^*_{16}	11	14	2:03:20
satellite04	124	89	0	6	0	P^*_{13}	11	24	3:08:22
satellite05	170	131	0	1	0	P^*_{9}	9	20	08:05:06
zenotravel01	18	6	0	1	0	P^*	3	3	4.93
zenotravel02	52	30	2	3	11	P^*_{17}	6	7	6:32:49
zenotravel03	49	26	0	4	0	P^*_{11}	6	7	6:55:47
storage01	15	2	3	3	0	P^*	3	3	0.01
storage02	38	2	1	3	0	P^*	3	3	0.03
storage03	32	4	1	3	53	P^*	3	3	1.54
storage04	49	35	3	6	0	P^*_{19}	8	8	18:13:37
storage05	50	25	0	3	0	P^*_{15}	6	9	30:00:44
storage06	61	47	0	2	0	P^*_{11}	6	9	15:23:33
storage07	126	87	4	8	0	P^*_{19}	14	14	23:31:22
scanalyzer22	12	8	0	2	0	P^*	5	5	8.61
scanalyzer23	12	8	0	2	0	P^*	5	5	8.65
scanalyzer24	12	8	0	2	0	P^*	5	5	8.61
transport01	61	24	0	2	0	P^*_{17}	5	5	20:44:15
transport11	82	54	0	1	0	P^*_{13}	10	11	5:17:18

Table 3

Results of the experimental trials of GP-WCSP*.

given length k, as well as in the calculation of the upper bound $MaxLev$. For each indispensable set $X = \{a_1, \ldots, a_t\}$, we can impose a new constraint in the WCSP at level k which says that at least one of these actions must occur in the solution-plan. (In practice, since the constraint propagation algorithms are specifically tailored for binary constraints, we replaced this new constraint of arity $m = O(tk)$ by an extra selector variable of domain size m together with m binary constraints.) We repeated our experimental trials using these new constraints in the WCSPs. Unfortunately, on average, the resulting pruning did not compensate the extra work created by the introduction of more variables and constraints.

10.3.2. Comparison with different costs of actions

Table 3 describes trials in which action-costs were random integers in the range 1 to 20. In order to test the influence of this range of costs on the efficiency of our approach, we repeated our experiments on a sample of 51 problems in which we simply added 20 to the cost of each action, so that action-costs ranged from 21 to 40. Table 4 gives the results. Compared to the results in Table 3, this increase in the minimum cost C_{min} led to a significant decrease in the values of $MaxLev$ (initial values ranging from 4 to 36 instead of from 12 to 404, and final values ranging from 2 to 27 instead of from 2 to 165). This allowed us to obtain a proof of optimality in 32 out of the 51 problems. For example, the problem *driverlog*01 which was not solved to optimality after nearly 19 hours when $C_{min} = 1$, was solved to optimality in less than 2 seconds when $C_{min} = 20$. These trials confirm that the efficiency of our approach is strongly related, through the value of $MaxLev$, to the range of action-costs. A better lower bound on the number of levels is obtained when the ratio between the average and the minimum costs of actions decreases.

10.3.3. Comparison with suboptimal planners

In another series of experiments we compared our approach with the results obtained by two planners which are suboptimal in the sense that they do not guarantee optimality. To this end, we used two versions of our planner: GP-WCSP which returns the first solution-plan found, which is necessarily optimal at this level, and GP-WCSP* which returns a globally optimal plan or a plan which is optimal at a certain level (as given in the column "Solution-plan" of Table 3). LPG [26] is a planner based on a local search method. We tested two versions: LPG-speed returns the first solution found and LPG-quality tries to optimize the quality of the solution. Since the heuristic employed by the two versions has a random component, we restarted the tests as many times as possible within the time used by our planner on the same problem. In order to differentiate between the two versions, we calculated the average cost of the plans found by LPG-speed and the cost of the best plan found by LPG-quality. SGPLAN [10] uses landmarks, that it orders so as to divide the problem into subproblems. It always returns the same solution-plan, whatever the number of iterations.

On average, LPG and SGPLAN solve the test problems in less than 1 second. Figure 9 gives the results of the comparison between the planners. It shows that the plans returned by LPG-speed and SGPLAN are rarely optimal. LPG-speed, even though restarted a large number of times (several hundred), failed to find an optimal solution to a third of the problems. For the same problem, the costs of the solution-plans that it returns are very variable (the worst being up to three times the cost of the best). The solutions returned by SGPLAN are rarely optimal: on average their cost is 24% greater than the cost of an optimal plan (and 89% greater in the case of *blocks*09). Although LPG-quality often finds an optimal-cost plan, no guarantee of optimality accompanies these optimal solutions. For certain problems (*blocks*07, *driverlog*02 and *driverlog*03), neither SGPLAN nor LPG-quality find as good solution-plan as GP-WCSP*. In the thirty-three problems tested, the solution returned by GP-WCSP* is always at least as good, and often better than the solution returned by LPG or SGPLAN. This is the case even when GP-WCSP* does not complete the search, in the sense that it does not provide us with a proof of optimality.

The cost of plans returned by our non-optimal planner GP-WCSP are, on average, only 5.7% greater than the cost of an optimal plan. As in many optimization problems, obtaining a proof of optimality is far more time-consuming than finding a good solution. We have, nevertheless demonstrated that our planning-graph based approach can produce a proof of optimality for certain non-trivial optimal-planning problems. Sub-

Problem	MaxLev					Solution			CPU
	init.	final	N_{ia}	$\|\mathcal{X}\|$	N_{tc}	plan	k	N_a	time
blocks01	8	5	6	6	0	P^*	6	6	11.40
blocks02	12	11	6	6	22	P^*	10	10	13.33
blocks03	7	5	6	6	0	P^*	6	6	3.23
blocks04	16	13	8	8	38	P^*	12	12	0:04:26
blocks05	13	10	7	7	0	P^*	10	10	0:02:24
blocks06	20	18	9	9	34	P^*	16	16	0:14:35
blocks07	15	12	11	11	0	P^*	12	12	0:15:28
blocks08	15	9	10	10	0	P^*	10	10	0:20:14
blocks09	25	23	11	11	0	P^*_{20}	20	20	0:53:41
blocks10	25	22	13	13	27	P^*_{20}	20	20	1:25:53
logistics01	26	20	19	19	0	P^*_{14}	9	20	14:28:39
logistics02	26	19	17	17	0	P^*_{12}	9	19	3:19:44
logistics03	21	15	13	13	63	P^*	9	15	1:33:41
logistics04	36	27	25	25	0	P^*_{12}	9	27	3:37:09
logistics05	22	17	15	15	0	P^*_{12}	9	17	5:01:10
logistics06	10	7	8	8	70	P^*	3	8	2.17
logistics07	32	25	23	23	0	P^*_{12}	9	25	6:34:06
logistics08	19	14	13	13	64	P^*_{13}	9	14	5:00:01
logistics09	34	25	23	23	0	P^*_{12}	11	25	23:10:16
depot01	14	10	7	12	65	P^*	8	10	0:03:00
depot02	21	18	5	6	0	P^*_{13}	12	15	17:12:11
driverlog01	10	7	1	5	28	P^*	7	7	1.43
driverlog02	23	21	0	3	0	P^*_{12}	12	17	5:59:49
driverlog03	14	12	8	16	112	P^*	7	12	15:51:50
driverlog04	24	21	0	11	0	P^*_{10}	7	18	2:27:29
driverlog05	23	22	0	7	0	P^*_{11}	8	18	2:59:30
driverlog06	14	13	0	2	0	P^*_9	5	11	2:32:05
driverlog07	25	16	0	3	0	P^*_9	7	13	3:31:35
driverlog08	35	27	0	14	0	P^*_{10}	9	22	7:42:56
rovers01	12	9	9	14	54	P^*	8	10	0:01:57
rovers02	11	8	7	9	46	P^*	4	8	0:13:13
rovers03	16	11	9	15	67	P^*	10	12	19:13:00
rovers04	11	7	8	14	78	P^*	4	8	0:23:59
rovers05	27	15	7	14	0	P^*_9	8	22	8:49:28
satellite01	11	8	8	11	51	P^*	8	9	15.93
satellite02	16	12	12	21	98	P^*	12	13	0:04:04
satellite03	18	13	1	6	82	P^*	10	11	0:21:26
satellite04	25	24	0	6	0	P^*_{12}	12	20	7:47:45
satellite05	29	23	0	1	0	P^*_9	8	18	14:55:38
storage01	4	2	3	3	0	P^*	3	3	0.00
storage02	4	2	1	3	0	P^*	3	3	0.02
storage03	4	2	3	5	57	P^*	3	3	0.09
storage04	9	8	6	9	52	P^*	8	8	36.37
storage05	10	8	2	7	40	P^*	6	9	0:03:59
storage06	11	8	4	7	170	P^*	8	8	1:04:35
storage07	19	17	8	14	121	P^*	14	14	3:35:03
scanalyzer22	6	4	4	6	34	P^*	5	5	3.23
scanalyzer23	6	4	4	6	34	P^*	5	5	3.23
scanalyzer24	6	4	4	6	34	P^*	5	5	3.23
transport01	8	4	4	7	100	P^*	5	5	17.63
transport11	18	10	4	8	136	P^*	9	9	0:01:49

Table 4
Results of the experimental trials of GP-WCSP* with costs of actions in the range 21 to 40.

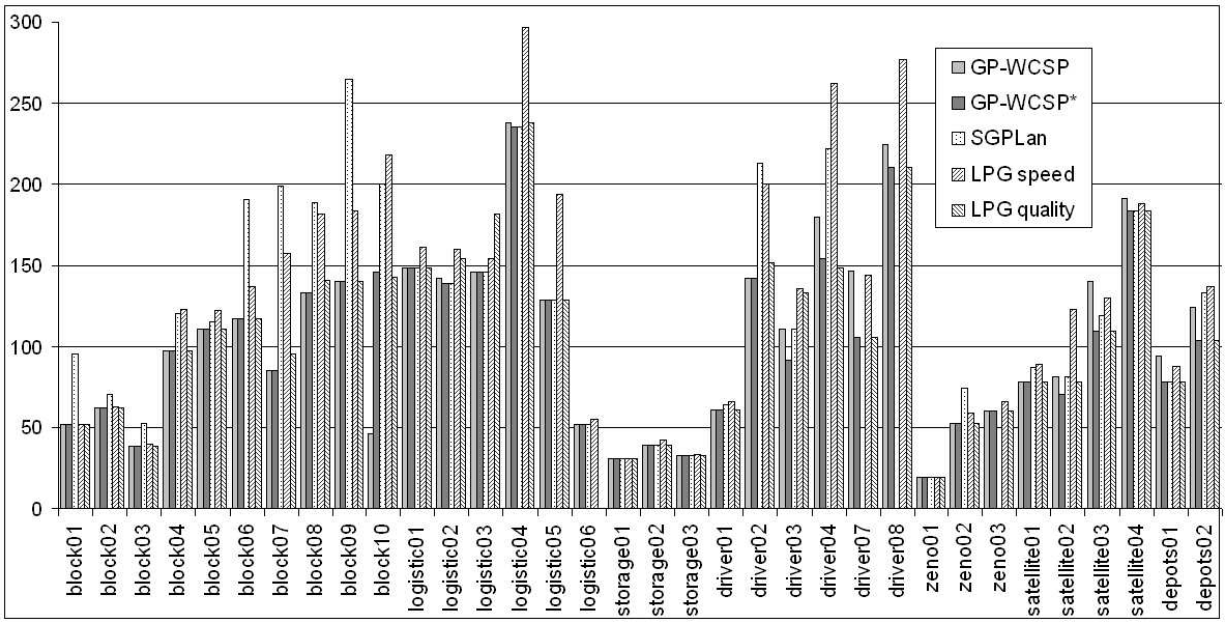

Fig. 9. Comparison of the costs of the solution-plans returned by GP-WCSP, GP-WCSP*, SGPLAN, LPG-speed and LPG-quality.

optimal planners, on the other hand, are essential for much larger problems for which an exhaustive search is not a practical possibility.

10.3.4. Comparison with optimal planners

In another set of experiments we compared GP-WCSP* with three state-of-the-art optimal planners, namely two versions of HSP* [35] (which performs a state-space search) and the baseline planner in the sequential optimization track of IPC2008, called SEQ-OPT-BASE[4]. HSP^*_0 (also known as TP4) performs a regression search, using IDA*, whereas HSP^*_a performs a regression search using IDAO*. SEQ-OPT-BASE performs a uniform cost search and implements an A* search with a zero heuristic. All three of these planners guarantee that if a solution-plan is returned then it is optimal. However, they are not specifically designed to have an anytime behaviour; if the execution of the program exceeded 27 hours or exceeded 950Mb of memory we stopped the program and no solution was returned.

Table 5 gives the results of our experiments on the same problems as in Table 3 except the Storage domain for HSP* because it does not accept problems from this domain. A dash means that the program was stopped after 27 hours or exceeded 950Mb of memory. (For some unexplained reason, in all the test problems from the rovers domain, HSP did not function correctly since it halted without finding a solution-plan even though the other planners found at least one.) For each problem, the plans returned by the different planners were of identical cost except three problems for which GP-WCSP* did not find the optimal plan and returned solutions whose costs were greater than the optimal by 2% for *blocks*10 and *driver*02, and 8% for *driver*04. As already indicated in Table 3, for many problems GP-WCSP* was not able to provide a complete proof of optimality due to excessive computation time, but instead produced a proof of optimality among parallel plans of length k.

For all problems solved by GP-WCSP*, one or other of the HSP*-based planners solved the problem to optimality (except *driver*08). Furthermore, GP-WCSP* was, on average, considerably slower than the other planners. Indeed, in most cases, GP-WCSP takes longer to find a cost-optimal plan among minimum-length parallel plans than the baseline planner SEQ-OPT-BASE takes to find a globally optimal plan. This is probably due to the fact that the search space of the largest WCSP that is solved by GP-WCSP* is much larger than the search space explored by a simple state-space

[4]http://ipc.informatik.uni-freiburg.de/Planners

Problem	GP-WCSP*	HSP^*_0	HSP^*_a	SEQ-OPT-BASE
blocks01	11.17	0.01	0.01	**0**
blocks02$^+$	0:10:12	0.01	0.01	**0**
blocks03	3.15	0.01	0.01	**0**
blocks04$^+$	0:47:36	0.03	0.08	**0**
blocks05$^+$	1:21:09	0.06	0.16	**0**
blocks06$^+$	0:25:19	0.03	0.35	**0**
blocks07$^+$	2:57:45	0.15	1.17	**0.01**
blocks08	0:19:44	0.30	1.13	**0.04**
blocks09$^+$	0:51:54	0.37	3.72	**0.04**
blocks10$^+$	1:23:08	0.31	42.16	**0.27**
logistics01$^+$	0:56:52	0:01:51	0.87	**0.12**
logistics02$^+$	0:59:01	0:07:11	1.46	**0.09**
logistics03$^+$	2:26:05	0.19	**0.06**	0.1
logistics04$^+$	2:23:47	-	0:00:50	**1.42**
logistics05$^+$	2:03:15	0:01:02	0.91	**0.25**
logistics06	0.92	0.06	0.04	**0.01**
logistics07$^+$	2:54:25	-	27.58	**4.64**
logistics08$^+$	8:38:32	97.54	**0.36**	0.83
logistics09$^+$	25:46:43	-	0:02:48	**8.2**
driverlog01$^+$	18:54:12	0.02	0.03	**0**
driverlog02$^+$	3:21:26	11:58:24	**0:00:24**	0.74
driverlog03$^+$	3:55:33	7.10	1.75	**0.16**
driverlog04$^+$	2:48:22	-	0:04:32	**26.61**
driverlog05$^+$	3:06:13	-	0:08:35	**0:02:03**
driverlog06$^+$	3:37:47	3:12:00	**24.42**	0:01:39
driverlog07$^+$	4:45:11	-	**0:06:06**	-
driverlog08$^+$	**6:41:08**	-	-	-
rovers01$^+$	7:25:29	no solution	no solution	**0.26**
rovers02$^+$	7:22:06	no solution	no solution	**0.26**
rovers03$^+$	3:31:20	no solution	no solution	**0.26**
rovers04$^+$	-	no solution	no solution	**0.26**
rovers05$^+$	01:50:38	no solution	no solution	**0.26**
satellite01$^+$	1:18:41	0.01	0.01	**0**
satellite02$^+$	1:36:32	0.80	0.14	**0.02**
satellite03$^+$	2:03:20	0:39:54	1.47	**0.44**
satellite04$^+$	3:08:22	-	0:02:29	13.39
satellite05$^+$	08:05:06	-	-	-
zenotravel01	4.93	0.01	0.01	**0**
zenotravel02$^+$	6:32:49	0.04	0.03	**0**
zenotravel03$^+$	6:55:47	1.43	1.39	**0.28**
storage01	0.01			**0**
storage02	0.03			**0**
storage03	1.54			**0**
storage04$^+$	18:13:37			**0**
storage05$^+$	-			**0**
storage06$^+$	15:23:23			**0.01**
storage07$^+$	23:31:22			**0.01**
depot01$^+$	11:32:33	0.05	0.09	**0**
depot02$^+$	9:07:34	0:00:57	-	**0.25**
scanalyzer22	8.61	0.02	-	**0**
scanalyzer23	8.65	0.02	-	**0**
scanalyzer24	8.61	0.02	-	**0**
transport01$^+$	20:44:15	0.1	-	**0**
transport11$^+$	5:17:18	0.1	-	**0**

Table 5

Comparison of the cpu time required by GP-WCSP*, HSP^*_0, HSP^*_a and SEQ-OPT-BASE. For the problems marked "+", GP-WCSP* only provided a proof of optimality up to k levels for the value of k given in column "plan" of Table 3. The best times are shown in bold type.

planner. The number of nodes explored by a simple A* algorithm is bounded by the total number of states. As a concrete example, in zenotravel03, there are only $5\times5\times5\times5\times3\times3\times7\times7 = 275,625$ states (since each of 4 travelers is in one of 3 cities or 2 planes, each of the 2 planes is one of 3 locations and has one of 7 fuel levels). By comparison, GP-WCSP needs to search a space whose size is greater than 10^{212} simply to find a first solution (see Table 1).

11. Conclusion

For certain problems, the quality of a plan is of primary importance. We have presented a new method for the extraction of an optimal solution-plan from a planning graph using Weighted CSP techniques. Experimental trials have demonstrated the considerable speed-up provided by the propagation of weighted constraints and the use of an appropriate variable-ordering heuristic. We were able to solve WCSPs an order of magnitude larger than random benchmark problems, due to the specific structure of the constraint graphs of the resulting WCSPs.

The number of planning-graph levels to construct in order to guarantee finding a globally optimum plan is unknown in advance. Our algorithm first produces a plan which has minimum cost among shortest-length parallel plans. Indeed, in certain applications this could be the type of optimality which is required. The cost of this plan is then used to find an upper bound $MaxLev$ on the number of levels to construct to ensure global optimality. Our experimental trials indicate that this upper bound can be significantly improved by detection of indispensable sets (sets of actions at least one of which must occur in a valid plan).

Extensive experimental trials have demonstrated that the search for a globally-optimal plan via construction of the planning graph is a practical possibility for non-trivial planning problems of moderate size from diverse domains. Nonetheless, in terms of computation time, our planner was simply not competitive with optimal planners based on a state-space search. This is probably due to the much larger search space that our planner had to explore.

However, the notions that we have defined in this article could be used by other algorithms (not necessarily based on the planning graph) to prune the search space when looking for an optimal-cost plan. In particular, we have presented several methods, based on the relaxed planning graph, for detecting indispensable sets of actions. Even on large benchmark planning problems, the detection of indispensable sets of actions was achieved in a fraction of a second. The sum of the minimum costs of actions in non-intersecting indispensable sets provides a natural lower bound on the cost of a solution-plan which could be used, for example, in a state-based search. The notion of too-costly actions can also be adapted to other algorithms provided we have a lower bound on the number of actions in a solution-plan.

Another possible avenue of future research is the investigation of automatic techniques for bounding the number of occurrences of actions, and in particular the minimum-cost action since this would help to tighten the bound $MaxLev$ on the number of planning-graph levels that need to be explored. As a simple example, an action which produces all the goal fluents cannot occur twice in an optimal plan. Another research direction is the calculation of a lower bound on the number of times an action is required in a plan via linear programming. Detecting indispensable actions which must occur more than once can again improve the bound $MaxLev$.

12. Acknowledgments

We would like to thank Cyril de Runz and Sylvain Cussat-Blanc for their important contributions to the construction of the GP-WCSP planner. We would also like to thank the anonymous referees for their numerous suggestions which greatly improved both the content and the presentation of this paper.

Appendix

A. Proof of NP-hardness of OCIS

Let $\phi : \{0,1\}^2 \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be the function

$$\phi(x,y) \;=\; \begin{cases} \infty \text{ if } x = y = 1 \\ 0 \quad \text{otherwise} \end{cases}$$

and let $\psi : \{0,1\} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be the function $\psi(0) = 1$, $\psi(1) = 0$.

Let WCSP(Γ) denote the set of instances of the Weighted Constraint Satisfaction Problem in which all the cost functions belong to Γ and costs lie in $\mathbb{R}_{\geq 0} \cup \{\infty\}$. In the case of boolean domains, there are only eight tractable classes of cost functions [12]. The set $\{\phi, \psi\}$ is a subset of none of these eight tractable classes, which implies that WCSP($\{\phi, \psi\}$) is NP-hard.

Let P be an instance of WCSP($\{\phi, \psi\}$) on the variables $V = \{v_1, \ldots, v_n\}$, represented by a graph $G_P = \langle V, E_P \rangle$, a subset $V_P \subseteq V$ and a constant K. The question is to determine whether there exists an assignment $s : V \rightarrow \{0, 1\}$ such that

$$\sum_{\{v_i, v_j\} \in E_P} \phi(s(v_i), s(v_j)) + \sum_{v_i \in V_P} \psi(s(v_i)) \leq K$$

Observe that any variable $v_i \notin V_P$ can be assigned the value 0 without incurring any cost in P. Such variables can hence be eliminated in a preprocessing step. Therefore, without loss of generality, in the following we assume $V_P = V = \{v_1, \ldots, v_n\}$.

To prove the theorem, it suffices to give a polynomial reduction from the problem WCSP($\{\phi, \psi\}$) to OCIS. For each variable v_i, we create a fluent B_i. For each edge $\{v_i, v_j\} \in E_P$, we create an action $a_{\{i,j\}}$ of cost 2 and such that $add(a_{\{i,j\}}) = \{B_i, B_j\}$. We also create an action a_0 of cost 1. Let $A = \{a_0\} \cup \{a_{\{i,j\}} : \{v_i, v_j\} \in E_P\}$. For each variable $v_i \in V_P$, let $X_i = \{a_{\{i,j\}} : \{v_i, v_j\} \in E_P\}$ be the set of actions which add the fluent B_i. Let $J = \{X_i : 1 \leq i \leq n\}$ and

$$G \;=\; \bigwedge_{1 \leq i \leq n} B_i$$

Let Π be the optimal-planning problem $\langle A, \emptyset, G \rangle$. By Lemma 7.5, the indispensable sets of Π are the sets X_i $(1 \leq i \leq n)$.

There is a bijection between the assignments $s : V \rightarrow \{0, 1\}$ such that $\forall \{v_i, v_j\} \in E_P$, $\phi(s(v_i), s(v_j)) = 0$ and the choices $\mathcal{X} = \{X_i : s(v_i) = 1, 1 \leq i \leq n\}$ of indispensable sets such that $\forall X_i, X_j \in \mathcal{X}$, $X_i \cap X_j = \emptyset$, since X_i, X_j intersect iff $\{v_i, v_j\} \in E_P$. Let $L = n - K$. Since $C_{min} = 1$ and $\forall X \in \mathcal{X}$, mincost$(X) = 2$,

$$\Sigma_{X \in \mathcal{X}}(\text{mincost}(X) - C_{min}) \;=\; |\mathcal{X}|$$

By definition of the function ψ,

$$\Sigma_{v_i \in V_P} \psi(s(v_i)) \;=\; n \;-\; |\mathcal{X}|$$

Hence $|\mathcal{X}| \geq L$ if and only if

$$\Sigma_{\{v_i, v_j\} \in E_P} \phi(s(v_i), s(v_j)) + \Sigma_{v_i \in V_P} \psi(s(v_i)) = \Sigma_{v_i \in V_P} \psi(s(v_i)) \leq \; n - L = K$$

Therefore, the solutions $\mathcal{X}$ to the instance of OCIS defined by Π correspond exactly to the solutions s to the Valued Constraint Satisfaction Problem P. This reduction is clearly polynomial.

References

[1] F. Bacchus and Q. Yang, Downward Refinement and the Efficiency of Hierarchical Problem Solving, *Artificial Intelligence* **71**(1) (1994), 43–100.

[2] A. Blum and M. Furst, Fast Planning Through Planning Graph Analysis, *Artificial Intelligence* **90**(1-2) (1997), 281–300.

[3] B. Bonet and H. Geffner, Planning as heuristic search, *Artificial Intelligence* **129**(1-2) (2001), 5–33.

[4] B. Bonet and H. Geffner, Heuristics for Planning with Penalties and Rewards using Compiled Knowledge, in: *International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, P. Doherty, J. Mylopoulos and C.A. Welty, eds, AAAI Press, Lake District (UK), 2006, pp. 452–462.

[5] R.I. Brafman and Y. Chernyavsky, Planning with Goal Preferences and Constraints, in: *International Conference on Automated Planning and Scheduling (ICAPS 2005)*, S. Biundo, K.L. Myers and K. Rajan, eds, AAAI Press, Monterey, California, USA, 2005, pp. 182–191.

[6] R.I. Brafman and C. Domshlak, Introducing Variable Importance Tradeoffs into CP-Nets, in: *Conference in Uncertainty in Artificial Intelligence (UAI 2002)*, A. Darwiche and N. Friedman, eds, Morgan Kaufmann, Edmonton, Alberta, Canada, 2002, pp. 69–76.

[7] A. Bundy, F. Giunchiglia, R. Sebastiani and T. Walsh, Computing Abstraction Hierarchies by Numerical Simulation, in: *Conference on Artificial Intelligence (AAAI 1996)*, volume 1, AAAI Press / The MIT Press, Portland, Oregon, 1996, pp. 523–529.

[8] T. Bylander, The Computational Complexity of Propositional STRIPS Planning, *Artificial Intelligence* **69**(1-2) (1994), 165–204.

[9] M. Cayrol, P. Régnier and V. Vidal, Least commitment in GRAPHPLAN, *Artificial Intelligence* **130**(1) (2001), 85–118.

[10] Y. Chen, C. Hsu and B.W. Wah, SGPlan: Subgoal Partitioning and Resolution in Planning, in: *International Conference on Automated Planning and Scheduling (ICAPS 2004)*, S. Zilberstein, J. Koehler and S. Koenig, eds, AAAI Press, Whistler, British Columbia, Canada, 2004, pp. 30–33.

[11] Y. Chen, X. Zhao and W. Zhang, Long Distance Mutual Exclusion for Propositional Planning, in: *International Joint Conference on Artificial Intelligence (IJCAI 2007)*, M.M. Veloso, eds, International Joint Conference on Artificial Intelligence, Hyderabad, India, 2007, pp. 1840–1845.

[12] D. Cohen, M.C. Cooper, P. Jeavons and A. Krokhin, The complexity of soft constraint satisfaction, *Artificial Intelligence* **170**(11) (2006), 983–1016.

[13] M.C. Cooper, Reduction operations in fuzzy or valued constraint satisfaction, *Fuzzy Sets and Systems* **134**(3) (2003), 311–342.

[14] M.C. Cooper, S. Cussat-Blanc, M. de Roquemaurel and P. Régnier, Soft arc consistency applied to optimal planning, in: *International Conference on Principles and Practice of Constraint Programming (CP 2006)*, F. Benhamou, eds, Springer, LNCS 4204, Nantes, France, 2006, pp. 680–684.

[15] M.C. Cooper, S. De Givry, M. Sanchez, T. Schiex and M. Zytnicki. Virtual arc consistency for Weighted CSP, in: *Conference on Artificial Intelligence (AAAI 2008)*, D. Fox and C.P. Gomes, eds, AAAI Press, Chicago, 2008, pp. 253–258.

[16] M.C. Cooper, S. de Givry and T. Schiex, Optimal soft arc consistency, in: *International Joint Conference on Artificial Intelligence (IJCAI 2007)*, M.M. Veloso, eds, International Joint Conference on Artificial Intelligence, Hyderabad, India, 2007, pp.68–73.

[17] M.C. Cooper and T. Schiex, Arc consistency for soft constraints, *Artificial Intelligence* **154**(1-2) (2004), 199–227.

[18] S. de Givry, F. Heras, M. Zytnicki and J. Larrosa, Existential arc consistency: Getting closer to full arc consistency in weighted CSPs, in: *International Joint Conference on Artificial Intelligence (IJCAI 2005)*, L. Pack Kaelbling and A. Saffiotti, eds, Professional Book Center, Edinburgh, Scotland, 2005, pp. 84–89.

[19] S. de Givry, J. Larrosa, P. Meseguer and T. Schiex, Solving Max-SAT as Weighted CSP, in: *International Conference on Principles and Practice of Constraint Programming (CP 2003)*, F. Rossi, eds, Springer, LNCS 2833, Kinsale, Ireland, 2003, pp. 363–376.

[20] C. de Runz, Utilisation de CSP valués pour la recherche de solutions dans les graphes de planification, Master's thesis, Université Paul Sabatier, Toulouse, France, 2005

[21] Y. Dimopoulos, B. Nebel and J. Koehler, Encoding Planning Problems in Nonmonotonic Logic Programs, in: *European Conference on Planning (ECP 1997)*, S. Steel and R. Alami, eds, Springer, LNCS 1348, Toulouse, France, 1997, pp. 169–181.

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

[23] S. Edelkamp, External Symbolic Heuristic Search with Pattern Databases, in: *International Conference on Automated Planning and Scheduling (ICAPS 2005)*, S. Biundo, K. L. Myers and K. Rajan, eds, AAAI, Monterey, California, USA, 2005, pp. 51–60.

[24] M. Fox and D. Long, The Automatic Inference of State Invariants in TIM, *Journal of Artificial Intelligence Research* **9** (1998), 367–421.

[25] M. Fox and D. Long, Utilizing Automatically Inferred Invariants in Graph Construction and Search, in: *International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, S. Chien, S. Kambhampati and C.A. Knoblock, eds, AAAI, Breckenridge, Colorado, USA, 2000, pp. 102-111.

[26] A. Gerevini, A. Saetti and I. Serina, Planning with Numerical Expressions in LPG, in: *European Conference on Artificial Intelligence (ECAI 2004)*, R. López de Mántaras and L. Saitta, eds, IOS Press, Valencia, Spain, 2004, pp. 667–671.

[27] A. Gerevini and L.K. Schubert, Inferring State Constraints for Domain-Independent Planning, in: *Conference on Artificial Intelligence (AAAI 1998)*, AAAI Press / The MIT Press, Madison, Wisconsin, USA, 1998, pp. 905–912.

[28] A. Gerevini and L.K. Schubert, Discovering State Constraints in DISCOPLAN: Some New Results, in: *Conference on Artificial Intelligence (AAAI 2000)*, AAAI Press / The MIT Press, Austin, Texas, USA, 2000, pp. 761–767.

[29] E. Giunchiglia and M. Maratea, Planning as Satisfiability with Preferences, in: *Conference on Artificial Intelligence (AAAI 2007)*, AAAI Press, Vancouver, British Columbia, Canada, 2007, pp. 987–992.

[30] E. Giunchiglia and M. Maratea, SAT-Based Planning with Minimal-#actions Plans and "soft" Goals, in: *Congress of the Italian Association for Artificial Intelligence (AI*IA 2007)*, R. Basili and M.T. Pazienza, eds, Springer, LNCS 4733, Rome, Italy, 2007, pp. 422–433.

[31] S. Grandcolas and C. Pain-Barre, Filtering, Decomposition and Search Space Reduction for Optimal Sequential Planning, in: *Conference on Artificial Intelligence (AAAI 2007)*, AAAI Press, Vancouver, British Columbia, Canada, 2007, pp. 993–998.

[32] S. Grandcolas and C. Pain-Barre, A Filtering and Decomposition Approach To Optimal Sequential Planning, in: *Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems, ICAPS 2006*, The English Lake District, Cumbria, UK, 2006, pp. 15-22.

[33] E. Guere and R. Alami, One action is enough to plan, in: *International Joint Conference on Artificial Intelligence (IJCAI 2001)*, B. Nebel, eds, Morgan Kaufmann, Seattle, Washington, USA, 2001, pp. 439–444.

[34] E.A. Hansen and R. Zhou, Anytime Heuristic Search, *Journal of Artificial Intelligence Research* **28** (2007), 267–297.

[35] P. Haslum, Improving Heuristics Through Relaxed Search - An Analysis of TP4 and HSP*a in the 2004 Planning Competition, *Journal of Artificial Intelligence Research* **25** (2006), pp. 233–267.

[36] P. Haslum, B. Bonet and H. Geffner, New Admissible Heuristics for Domain-Independent Planning, in: *Conference on Artificial Intelligence (AAAI 2005)*, M.M. Veloso and S. Kambhampati, eds, AAAI Press / The MIT Press, Pittsburgh, Pennsylvania, USA, 2005, pp. 1163–1168.

[37] P. Haslum and H. Geffner, Admissible Heuristics for Optimal Planning, in: *International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*, S. Chien, S. Kambhampati and C.A. Knoblock, eds, AAAI, Breckenridge, Colorado, USA, 2000, pp. 140-149.

[38] M. Helmert, Complexity results for standard benchmark domains in planning, *Artificial Intelligence* **143**(2) (2003), 219–262.

[39] M. Helmert and C. Domshlak, Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?, in: *International Conference on Automated Planning and Scheduling (ICAPS 2009)*, A. Gerevini, A. E. Howe, A. Cesta and I. Refanidis, eds, Thessaloniki, Greece, 2009.

[40] M. Helmert, P. Haslum and J. Hoffmann, Flexible Abstraction Heuristics for Optimal Sequential Planning, in: *International Conference on Automated Planning and Scheduling (ICAPS 2007)*, M.S. Boddy, M. Fox and S. Thiébaux, eds, AAAI, Providence, Rhode Island, USA, 2007, pp. 176–183.

[41] J. Hoffman, Where 'Ignoring Delete Lists' Works: Local Search Topology in Planning Benchmarks, *Journal of Artificial Intelligence Research* **24** (2005), 685–758.

[42] J. Hoffmann, J. Porteous and L. Sebastia, Ordered Landmarks in Planning, *Journal of Artificial Intelligence Research* **22** (2004), 215–278.

[43] P. Johnsson and A. Krokhin, Recognizing frozen variables in constraint satisfaction problems, *Theoretical Computer Science* **329**(1-3) (2004), 93–113.

[44] E. Karpas and C. Domshlak, Cost-Optimal Planning with Landmarks, in: *International Joint Conference on Artificial Intelligence (IJCAI 2009)*, C. Boutilier, ed., Pasadena, CA, USA, 2009, pp. 1728–1733.

[45] H.A. Kautz and B. Selman, Planning as Satisfiability, in: *European Conference on Artificial Intelligence (ECAI 1992)*, B. Neumann, eds, John Wiley and Sons, Vienna, Austria, 1992, pp. 359-363.

[46] H.A. Kautz and B. Selman, Pushing the Envelope: Planning, Propositional Logic and Stochastic Search, in: *Conference on Artificial Intelligence (AAAI 1996)*, volume 2, AAAI Press / The MIT Press, Portland, Oregon, 1996, pp. 1194–1201.

[47] H.A. Kautz and B. Selman, Unifying SAT-Based and Graph-Based Planning, in: *International Joint Conference on Artificial Intelligence (IJCAI 1999)*, T. Dean, ed., Morgan Kaufmann, Stockholm, Sweden, 1999, pp. 318–325.

[48] H.A. Kautz, B. Selman and J. Hoffmann, SatPlan: Planning as Satisfiability, in: *Abstracts of 5th International Planning Competition, ICAPS 2006*, The English Lake District, Cumbria, UK, 2006.

[49] H.A. Kautz and J.P. Walser, State-space Planning by Integer Optimization, in: *Conference on Artificial Intelligence (AAAI 1999)*, AAAI Press / The MIT Press, Orlando, Florida, USA, 1999, pp. 526–533.

[50] G. Kelleher and A.G. Cohn, Automatically Synthesising Domain Constraints from Operator Descriptions, in: *European Conference on Artificial Intelligence (ECAI 1992)*, B. Neumann, eds, John Wiley and Sons, Vienna, Austria, 1992, pp. 653–655.

[51] C.A. Knoblock, Automatically Generating Abstractions for Planning, Artificial Intelligence **68**(2) (1994), 243–302.

[52] J. Koehler, Solving Complex Planning Tasks Through Extraction of Subproblems, in: *International Conference on Artificial Intelligence Planning Systems (AIPS 1998)*, R.G. Simmons, M.M. Veloso and S.Smith, eds, AAAI, Pittsburgh Pennsylvania, USA, 1998, pp. 62–69.

[53] J. Koehler and J. Hoffmann, On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm, *Journal of Artificial Intelligence Research* **12** (2000), 338–386.

[54] J. Koehler and J. Hoffmann, On the Instantiation of ADL Operators Involving Arbitrary First-Order Formulas, in: *Proceedings ECAI-00 Workshop on New Results in Planning, Scheduling and Design*, 2000

[55] J. Kvarnström, Applying Domain Analysis Techniques for Domain-Dependent Control in TALplanner, in: *International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*, M. Ghallab, J. Hertzberg, P. Traverso, eds, AAAI, Toulouse, France, 2002, pp. 101–111.

[56] J. Larrosa, Node and Arc Consistency in Weighted CSP, in: *Conference on Artificial Intelligence (AAAI 2002)*, AAAI Press, Edmonton, Alberta, Canada, 2002, pp. 48–53.

[57] J. Larrosa and T. Schiex, In the quest of the best form of local consistency for Weighted CSP, in: *International Joint Conference on Artificial Intelligence (IJCAI 2003)*, G. Gottlob, T. Walsh, eds, Morgan Kaufmann, Acapulco, Mexico, 2003, pp. 239-244.

[58] A. Lopez and F. Bacchus, Generalizing GraphPlan by Formulating Planning as a CSP, in: *International Joint Conference on Artificial Intelligence (IJCAI 2003)*, G. Gottlob, T. Walsh, eds, Morgan Kaufmann, Acapulco, Mexico, 2003, pp. 954–960.

[59] P. Meseguer, F. Rossi and T. Schiex, Soft Constraints, in: *Handbook of Constraint Programming*, F. Rossi, P. van Beek and T. Walsh, eds, Elsevier, 2006, chapter 9.

[60] I. Miguel, P. Jarvis and Q. Shen, Flexible Graphplan, in: *European Conference on Artificial Intelligence (ECAI 2000)*, W. Horn, eds, IO Press, Berlin, Germany, 2000, pp. 506–510.

[61] I. Miguel, Q. Shen and P. Jarvis, Efficient Flexible Planning via Dynamic Flexible Constraint Satisfaction, *Engineering Applications of Artificial Intelligence* **14**(3) (2001), pp. 301–327.

[62] S. Mittal and B. Falkenhainer, Dynamic Constraint Satisfaction Problems, in: *Conference on Artificial Intelligence (AAAI 1990)*, AAAI Press / The MIT Press, Boston, Massachusetts, 1990, pp. 25–32.

[63] J. Porteous, L. Sebastia and J. Hoffmann, On the Extraction, Ordering, and Usage of Landmarks in Planning, in: *Recent Advances in AI Planning. European Conference on Planning (ECP 2001)*, Toledo, Spain, 2001, pp. 37–48.

[64] J. Rintanen, A Planning Algorithm not based on Directional Search, in: *International Conference on Principles of Knowledge Representation and Reasoning (KR 1998)*, A.G. Cohn, L.K. Schubert, S.C. Shapiro, eds, Morgan Kaufmann, Trento, Italy, 1998, pp. 617–625.

[65] J. Rintanen, An Iterative Algorithm for Synthesizing Invariants, in: *Conference on Artificial Intelligence (AAAI 2000)*, AAAI Press / The MIT Press, Austin, Texas, USA, 2000, pp. 806–811.

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

[67] T. Schiex, H. Fargier and G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: *International Joint Conference on Artificial Intelligence (IJCAI 1995)*, Morgan Kaufmann, Montréal, Québec, Canada, 1995, pp. 631–637.

[68] R. Tsuneto, J.A. Hendler and D.S. Nau, Analyzing External Conditions to Improve the Efficiency of HTN Planning, in: *Conference on Artificial Intelligence (AAAI 1998)*, AAAI Press / The MIT Press, Madison, Wisconsin, USA, 1998, pp. 913–920.

[69] P. van Beek and X. Chen, CPlan: A Constraint Programming Approach to Planning, in: *Conference on Artificial Intelligence (AAAI 1999)*, AAAI Press / The MIT Press, Orlando, Florida, USA, 1999, pp. 585–590.

[70] M. van den Briel, T. Vossen and S. Kambhampati, Reviving Integer Programming Approaches for AI Planning: A Branch-and-Cut Framework, in: *International Conference on Automated Planning and Scheduling (ICAPS 2005)*, S. Biundo, K.L. Myers and K. Rajan, eds, AAAI Press, Monterey, California, USA, 2005, pp. 310–319.

[71] M.H.L. van den Briel, J. Benton, S. Kambhampati & T. Vossen, An LP-Based Heuristic for Optimal Planning, *Proc. International Conference on Principles and Practice of Constraint Programming (CP)* (2007) pp. 651–665.

[72] D.S. Weld, Recent advances in AI planning, *AI Magazine* **20** (1999), 93–123.

[73] Q. Yang, J.D. Tenenberg and S. Woods, On the implementation and evaluation of ABTWEAK, *Computational Intelligence* **12**(2) (1996), 295–318.

[74] R. Zhou and E.A. Hansen, Breadth-First Heuristic Search, in: *International Conference on Automated Planning and Scheduling (ICAPS 2004)*, S. Zilberstein, J. Koehler and S. Koenig, eds, AAAI Press, Whistler, British Columbia, Canada, 2004, pp. 92–100.

[75] R. Zhou and E.A. Hansen, Breadth-first heuristic search, *Artificial Intelligence* **170**(4-5) (2006), 385–408.