Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

Handling Goal Utility Dependencies in a Satisfiability Framework

Richard Russell and **Sean Holden**
Computer Laboratory, University of Cambridge
15 J J Thomson Avenue, Cambridge, CB3 0FD, UK
{Richard.Russell, Sean.Holden}@cl.cam.ac.uk

Abstract

Goal utility dependencies arise when the utility of achieving a goal depends on the other goals that are achieved with it. This complicates the planning procedure because achieving a new goal can potentially alter the utilities of all the other goals currently achieved. In this paper, we present an encoding procedure that enables general-purpose Max-SAT solvers to be used to solve planning problems with goal utility dependencies. We compare this approach to one using integer programming via an empirical evaluation using benchmark problems from past international planning competitions. Our results indicate that this approach is competitive and sometimes more successful than an integer programming one – solving two to three times more subproblems in some domains, while being outperformed by only a significantly smaller margin in others.

Introduction

In a classical planning problem, valid plans must achieve every goal; however, when faced with limited resources, it is often useful to relax this constraint and allow the planner to make a trade-off between the benefit and cost of achieving a set of goals. Such planning problems are often called *partial satisfaction* or *oversubscription* problems. There is a growing body of research concerning how to utilize concepts of utility and preference from decision theory in the solution of such problems. This has produced a renewed interest in modeling problems as integer programs due to their facility for explicitly expressing an optimization function over solutions. Despite integer programming (IP) lagging behind satisfiability in performance, the possibility of a satisfiability approach to partial satisfaction problems in classical planning has largely been neglected.

In previous years, satisfiability approaches to planning have enjoyed much success: in the 4th and 5th International Planning Competition (IPC) SATPLAN (Kautz and Selman 1999; Kautz, Selman, and Hoffmann 2006) achieved first and joint first prizes respectively for optimal planning in propositional domains. In this context, an optimal plan is one with the smallest number of discrete time steps, often referred to as the plan's *makespan*. This notion of optimality is not a particularly natural one; instead IPC-6 adopts a collection of different metrics for optimization: number of actions, total action cost and net benefit, where the net benefit of a plan is the total utility of the goals achieved minus the cost of the executed actions. This raises the question of how the total utility of a set of goals should be defined.

The partial satisfaction planning (PSP) net benefit metric (van den Briel et al. 2004) assumes that each goal has an individual utility, and the total utility of a set of goals achieved by a plan can be expressed simply as the sum of their individual utilities. This particular model is worth studying because it is conceptually simple to state but can express a variety of interesting problems: the class of Simple Preferences from PDDL3 – a recent version of the de facto standard for expressing planning problems in the AI community – can be reduced to PSP (Benton, Do, and Kambhampati 2009).

There are scenarios where the utility of achieving a single goal depends upon which other goals are co-achieved. One such example arises in the IPC Rovers domain where it seems desirable to achieve complementary goals, which give good scientific coverage. For example, the utility of taking an image and a rock sample from a single site should be greater than the sum of the utilities of achieving either of those tasks in isolation.

Groups of goals do not always interact positively, and there are scenarios where jointly achieving two or more goals has a reduced utility over achieving any one of them in isolation. For a purchasing problem, the agent may want a single set of chairs, but purchasing more than one set has an adverse effect: the extra sets impair movement and introduce additional hazards through overcrowding, which the agent would rather avoid.

Do et al. (2007) described these types of problems as having *goal utility dependencies*, and they presented a systematic approach for handling them using the *Generalized Additive Independence* (GAI) model of utility and integer programming. Their ıPUD planner finds solutions with maximum net benefit for a bounded makespan horizon, but plans may exist with larger makespans that have a greater net benefit.

Thus, it seems that integer programming has received renewed interest for this type of planning because of its facility for explicitly expressing an optimization function. There has been some work on handling preferences in SAT: SAT-

PLAN has been extended to minimize the cost of executed actions in planning within a bounded horizon (Giunchiglia and Maratea 2007). In principle, it should be possible to extend this to solve PSP net benefit problems. However, maximum satisfiability (Max-SAT), in particular its weighted variants, also offers a facility for explicitly defining an optimization function in a similar manner to integer programming.

Max-SAT is the optimization variant of SAT. The problem is to find a truth assignment to variables that maximizes the number of satisfied clauses. Further variations can be made on the problem, and one such variation that is of particular interest to us is the Weighted Partial Max-SAT problem (WPMax-SAT). These problems can contain clauses of two types: hard or soft. For our purposes, each soft clause has a non-negative integer weight which is its violation cost. The WPMax-SAT problem is to find a truth assignment to variables that satisfies all hard clauses and maximizes the sum of the weights of the satisfied soft clauses.

To our knowledge a system for handling preferences using general-purpose WPMax-SAT solvers has not been explored, nor has a satisfiability approach to handling goal utility preferences been investigated. In this paper we develop these ideas. Our main contributions in this paper are:

- An encoding scheme for representing PSP net benefit problems with goal utility dependencies in WPMax-SAT together with a technical extension, which we call MSATPLAN, to SATPLAN that implements this using a general-purpose Max-SAT solver.
- A thorough empirical comparison between MSATPLAN and IPUD, where we find that MSATPLAN has competitive and often better performance than its IP counterpart.

Encoding Utilities

A classical planning problem is described by a 4-tuple $\langle \mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ where $\mathcal{F}$ is a set of predicate symbols, $\mathcal{I}$ is a set of ground predicates that are true in the initial state, $\mathcal{G}$ is a set of ground predicates that must be true in the goal state and $\mathcal{A}$ is a set of actions. Each action $a \in \mathcal{A}$ has a precondition set *Pre*(a) of ground predicates that must all be true for a to be applicable; it also has a set *Add*(a) of add effects and a set *Del*(a) of delete effects that contain ground predicates which are made true and false respectively if a is executed. We define a plan P as an ordered sequence of actions $(a_1, \ldots, a_m)$ which achieve a subset of the ground predicates in $\mathcal{G}$ when executed in order from the initial state in which only the predicates in $\mathcal{I}$ are true.

For a nonempty set $G \subseteq \mathcal{G}$ of N ground predicates with indexed elements $\{g_1, \ldots, g_N\}$ we will use *pseudo-Boolean functions* of the form $f\colon \mathbb{B}^N \to \mathbb{N}_0$ where $\mathbb{B} = \{1, 0\}$ and $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$. For a subset $S \subseteq G$ of goals achieved by a particular plan, we define a vector $\mathbf{1}_S^G = (x_1, \ldots, x_N)$ such that

$$x_i = \begin{cases} 1 & \text{if } g_i \in S \\ 0 & \text{otherwise.} \end{cases}$$

For a planning problem, we assume we are given a function $u\colon \mathbb{B}^{|\mathcal{G}|} \to \mathbb{N}_0$ that specifies a utility for each possible combination of goals we could achieve. In the worst case, this will require space exponential in the number of goals to describe u; however, we can achieve a more compact representation if we assume that u can be factorized in some way. The *Generalized Additive Independence* (GAI) model (Bacchus and Grove 1995) provides one such method for decomposing a utility function and is used by Do et al. (2007) in their framework for handling goal utility dependencies. Let us split $\mathcal{G}$ into k, not necessarily disjoint, nonempty subsets $G_1, \ldots, G_k$ such that $\bigcup_{i=1,\ldots,k} G_i = \mathcal{G}$. The utility function u has an *additive decomposition* over $G_1, \ldots, G_k$ if u can be expressed as

$$u(\mathbf{1}_S^{\mathcal{G}}) = \sum_{i=1}^{k} f_i(\mathbf{1}_{G_i \cap S}^{G_i}) \tag{1}$$

for a collection of k functions $f_i\colon \mathbb{B}^{|G_i|} \to \mathbb{N}_0$. Hopefully, k and the size of each G_i will be sufficiently small to allow us to represent the function using less space than a single tabular representation of u over $|\mathcal{G}|$, thus leading to a more practical encoding.

The utility of a set of goals achieved by a plan is only one aspect of its quality; there are often many different plans that will achieve a particular set of goals. In order to distinguish between them, we should include their execution cost in our measure of plan quality. For each action $a \in \mathcal{A}$ we associate a cost $c(a) \in \mathbb{N}_0$ for executing that action. By writing the set of goals achieved by a plan P as $\mathsf{Goals}(P)$, we can now precisely define the most preferred plan $P^\star$ as the plan that maximizes the *net benefit*:

$$P^\star = \operatorname*{argmax}_{\substack{\text{valid plans} \\ P=(a_1,\ldots,a_m)}} \left(u\left(\mathbf{1}_{\mathsf{Goals}(P)}^{\mathcal{G}}\right) - \sum_{i=1}^{m} c(a_i) \right). \tag{2}$$

Weighted Partial Max-SAT (WPMax-SAT)

WPMax-SAT is the variation of Max-SAT that we use in our encodings. A weighted clause is a disjunction of literals $\ell_1, \ldots, \ell_q$ with an associated weight $w \in \mathbb{N}_0 \cup \{\top\}$, which we write as

$$\underbrace{(\ell_1 \vee \cdots \vee \ell_q)}_{w}. \tag{3}$$

Weighted clauses are either *hard* or *soft*. A clause is hard if its weight is the special element $\top$; all other clauses are soft. Let Φ be a conjunction of weighted clauses. A *complete* truth assignment to the variables in Φ gives each variable a truth value. A *model* for Φ is a complete truth assignment to the variables in Φ that satisfies all hard clauses. The *cost* of a model for Φ is the sum of the weights of the falsified clauses in Φ produced by that truth assignment. The WPMax-SAT problem is to find a model of minimum cost for Φ. If Φ consists of only soft weighted clauses then we have the weighted Max-SAT problem; if all clauses are soft with weight 1 then we have the Max-SAT problem.

Plan Encoding

We extend the 'thin-gp' encoding from SATPLAN (Kautz, Selman, and Hoffmann 2006) to create a formula φ_h^T in

propositional logic with the property that a satisfying assignment to it allows us to extract a plan with a makespan of at most T from the truth assignment; however, since all goals are soft, we do not include the clauses that force each goal to be true at level T, and we add additional axioms, not included in the original 'thin-gp' encoding, that ensure a goal is not achieved by the extracted plan if it is false at level T.

Algorithm 1: Encoding the planning graph (φ_h^T).

Result: φ_h^T encoding the planning problem.

1 **begin**
2 $\quad\varphi \leftarrow$ **true**
3 $\quad$**foreach** $\rho \in \mathcal{I}$ **do**
4 $\quad\quad\varphi \leftarrow \varphi \wedge (V_{pred}(\rho, 0))$
5 $\quad$**for** $t \leftarrow 1$ **to** T **do**
6 $\quad\quad$**foreach** $a \in \mathcal{A}'$ *applicable at level* t **do**
7 $\quad\quad\quad$**foreach** $\rho \in Pre(a)$ **do**
8 $\quad\quad\quad\quad\varphi \leftarrow \varphi \wedge (\neg V_{act}(a, t) \vee V_{pred}(\rho, t-1))$
9 $\quad\quad$**foreach** $\rho \in \mathcal{F}$ *reachable at level* t **do**
10 $\quad\quad\quad C \leftarrow \neg V_{pred}(\rho, t)$
11 $\quad\quad\quad$**foreach** $a \in \mathcal{A}'$ *applicable at level* t *such that* $\rho \in Add(a)$ **do**
12 $\quad\quad\quad\quad C \leftarrow C \vee V_{act}(a, t)$
13 $\quad\quad\quad\varphi \leftarrow \varphi \wedge C$
14 $\quad\quad$**foreach** $a_1, a_2 \in \mathcal{A}'$ *applicable at level* t **do**
15 $\quad\quad\quad$**if** `Mutex` (a_1, a_2, t) **then**
16 $\quad\quad\quad\quad\varphi \leftarrow \varphi \wedge (\neg V_{act}(a_1, t) \vee \neg V_{act}(a_2, t))$
17 $\quad\quad$**foreach** $\rho_1, \rho_2 \in \mathcal{F}$ *reachable at level* t **do**
18 $\quad\quad\quad$**if** `Mutex` (ρ_1, ρ_2, t) **then**
19 $\quad\quad\quad\quad\varphi \leftarrow \varphi \wedge (\neg V_{pred}(\rho_1, t) \vee \neg V_{pred}(\rho_2, t))$
20 $\quad\quad$**foreach** $g \in \mathcal{G}$ *reachable at level* t **do**
21 $\quad\quad\quad$**foreach** $a \in \mathcal{A}'$ *applicable at level* t *such that* $g \in Add(a)$ **do**
22 $\quad\quad\quad\quad\varphi \leftarrow \varphi \wedge (V_{pred}(g, t) \vee \neg V_{act}(a, t))$
23 $\quad\quad\quad$**if** g *is reachable at level* $t-1$ **then**
24 $\quad\quad\quad\quad C \leftarrow V_{pred}(g, t) \vee \neg V_{pred}(g, t-1)$
25 $\quad\quad\quad\quad$**foreach** $a \in \mathcal{A}'$ *applicable at level* t *such that* $g \in Del(a)$ **do**
26 $\quad\quad\quad\quad\quad C \leftarrow C \vee V_{act}(a, t)$
27 $\quad\quad\quad\quad\varphi \leftarrow \varphi \wedge C$
28 $\quad\varphi_h^T \leftarrow \varphi$
29 **end**

A summary of the steps involved in producing this encoding can be found in Algorithm 1. Lines 3–19 are from the original 'thin-gp' encoding; we write $\mathcal{A}'$ to denote $\mathcal{A}$ extended to include NOOP actions for each ground predicate. For more details the reader should refer to the above paper.

A planning graph (Blum and Furst 1997) of makespan T is built from domain and problem PDDL files. For $0 \leq t \leq T$, a binary variable $V_{pred}(\rho, t)$ is created for each ground predicate ρ that is reachable at time step t. For $1 \leq t \leq T$, a binary variable $V_{act}(a, t)$ is created for each action a that has reachable and non-mutex preconditions at level $t-1$.

A satisfying truth assignment, $\mathcal{S}$, to the variables in φ_h^T corresponds to a valid plan $\mathsf{Plan}(\mathcal{S})$; however, it is no longer necessary for a valid plan to achieve all the goals in $\mathcal{G}$. For each goal predicate $\rho_i \in \mathcal{G}$, if ρ_i is in the planning graph at level T then the variable $V_{pred}(\rho_i, T)$ occurs in φ_h^T and we say that ρ_i *is coded for as a goal in* φ_h^T. If ρ_i has been coded for as a goal then ρ_i is achieved by executing $\mathsf{Plan}(\mathcal{S})$ from the initial state iff $\mathcal{S}[V_{pred}(\rho_i, T)]$ is true; in all other cases, including those for which ρ_i has not been coded for as a goal, $\mathsf{Plan}(\mathcal{S})$ does not achieve ρ_i. The 'only if' condition is ensured by the addition of the following axiom to the encoding:

$$\neg V_{pred}(g, t) \Rightarrow \left(\neg V_{pred}(g, t-1) \vee \bigvee_{\substack{a \in \mathcal{A} \\ \text{s.t. } g \in Del(a)}} V_{act}(a, t) \right) \wedge \bigwedge_{\substack{a \in \mathcal{A} \\ \text{s.t. } g \in Add(a)}} \neg V_{act}(a, t), \ (\forall g \in \mathcal{G}, 1 \leq t \leq T), \quad (4)$$

which produces the clauses as described in lines 20–27 in Algorithm 1.

Optimization Function

So far, we have presented a method for creating a clausal formula φ_h^T in propositional logic, consisting only of hard clauses, from which we can extract a valid plan from any satisfiable truth assignment to that formula. In order to guide the search procedure to plans of high net benefit, we need to specify an optimization function over solutions. Our approach is to construct a clausal formula φ_s^T in propositional logic, consisting only of weighted soft clauses, with the property that an optimal satisfiable truth assignment $\mathcal{S}^\star$ to the WPMax-SAT formula $\Phi^T = \varphi_h^T \wedge \varphi_s^T$ gives a plan $\mathsf{Plan}(\mathcal{S}^\star)$ of maximum net benefit over all possible plans of makespan less than or equal to T.

We assume that our utility function u has an additive decomposition over $G_1, \ldots, G_k$ as given in Equation 1. We introduce a measure which we call the *residual utility* resulting from a truth assignment to the arguments of a function f_i in the additive decomposition of u. The residual utility is the amount of utility that we failed to secure by choosing this truth assignment over one that would maximize the utility for this factor. More precisely, let the maximum of the function be

$$\bar{f}_i = \max_{\mathbf{v} \in \mathbb{B}^{|G_i|}} f_i(\mathbf{v}). \quad (5)$$

Define the function $r_i : \mathbb{B}^{|G_i|} \rightarrow \mathbb{N}_0$ that calculates the residual utility of a truth assignment $\mathbf{v} \in \mathbb{B}^{|G_i|}$ to the arguments of f_i as

$$r_i(\mathbf{v}) = \bar{f}_i - f_i(\mathbf{v}). \quad (6)$$

Using this measure, and assuming that all predicates in $\mathcal{G}$ are coded for as goals in φ_h^T, we construct φ_s^T such that it satisfies the property that for every complete satisfiable assignment, $\mathcal{S}$ to Φ^T, the following holds:

1. For each action a that is executed in $\mathsf{Plan}(\mathcal{S})$, a unique clause is violated in φ_s^T with weight $c(a)$.

2. For each G_i in the additive decomposition of u, a unique clause is violated in φ_s^T with weight $r_i(\mathbf{1}^{G_i}_{G_i \cap S})$ where S is the set of goals achieved by $\mathsf{Plan}(\mathcal{S})$.
3. No other clauses are violated.

If φ_s^T satisfies this property, then the sum of weights of violated clauses for such a truth assignment $\mathcal{S}$ will be given by

$$\sum_{i=1}^{k} r_i\Big(\mathbf{1}^{G_i}_{G_i \cap \mathsf{Goals}(\mathsf{Plan}(\mathcal{S}))}\Big) + \sum_{i=1}^{m} c(a_i), \tag{7}$$

where $\mathsf{Plan}(\mathcal{S}) = (a_1, \ldots, a_m)$. An optimal WPMax-SAT solver applied to Φ^T will find the truth assignment $\mathcal{S}^\star$ that minimizes this quantity which is equivalent to maximizing its negative. Therefore, $\mathcal{S}^\star$ is given by

$$\mathcal{S}^\star = \underset{\substack{\text{Satisfiable T.A.} \\ \mathcal{S} \text{ to } \Phi^T}}{\operatorname{argmax}} \left[\sum_{i=1}^{k} f_i\Big(\mathbf{1}^{G_i}_{G_i \cap \mathsf{Goals}(\mathsf{Plan}(\mathcal{S}))}\Big) - \sum_{i=1}^{m} c(a_i) \right], \tag{8}$$

where the a_i and m are dependent on $\mathcal{S}$ such that $\mathsf{Plan}(\mathcal{S}) = (a_1, \ldots, a_m)$. Note that the quantity being maximized is a form of Equation 2; thus, a WPMax-SAT solver applied to Φ^T will find a valid plan that is optimal up to the makespan T with regard to maximizing the net benefit metric.

Encoding the Optimization Function

We have seen that if φ_s^T satisfies the property we outlined above, then the plans produced maximize net benefit for a fixed makespan. Now we discuss the details of how such a formula is constructed according to our procedure shown in Algorithm 2.

The first part of the property is encoded in lines 4–6 where a clause is added to φ_s^T for each action that is applicable at each level up to the makespan T. If an action a is executed at level t in a plan extracted from a truth assignment, then $V_{act}(a,t)$ is necessarily true from the definition of φ_h^T. Consequently, the clause $(\neg V_{act}(a,t))$ with weight $c(a)$ is violated and $c(a)$ is added to the cost of the truth assignment. If the action is not executed, then $V_{act}(a,t)$ is false and its corresponding clause is satisfied and makes no contribution to the cost of the assignment.

The second part of the property is ensured by lines 7–17; the objective is to produce, for each G_i and each truth assignment to predicates in G_i, a soft clause, weighted by the residual utility, that is violated iff the predicates in G_i take on that truth assignment. Our procedure is made more complicated by accounting for situations where one or more predicates in $\mathcal{G}$ are not coded for as goals in φ_h^T.

If a particular G_i is being processed, for each truth assignment, the predicates in G_i are split into two sets, π^+ and π^-, depending on whether the predicate is assigned true or false respectively (lines 9–10). We then check to see if the truth assignment might be possible on line 11 by checking if any pair of predicates in π^+ is known to be mutex at the final level of the plan. If this is true then the truth assignment will never satisfy φ_h^T so there is no need to add a clause for this particular truth assignment to G_i to φ_s^T.

Algorithm 2: Encoding the optimization function (φ_s^T).

Result: φ_s^T encoding the optimization function.

1 **begin**
2 $\quad \Omega_T \leftarrow \{\, \rho_j \in \mathcal{G} \mid \rho_j$ is coded for as a goal in $\varphi_h^T \,\}$
3 $\quad \varphi \leftarrow$ **true**
4 $\quad$ **for** $t \leftarrow 1$ **to** T **do**
5 $\quad\quad$ **foreach** *action* $a \in \mathcal{A}$ *applicable at level* t **do**
6 $\quad\quad\quad \varphi \leftarrow \varphi \wedge \underbrace{(\neg V_{act}(a,t))}_{c(a)}$
7 $\quad$ **for** $i \leftarrow 1$ **to** k **do**
8 $\quad\quad$ **foreach** $\mathbf{v} \in \mathbb{B}^{|G_i|}$ **do**
9 $\quad\quad\quad \pi^+ \leftarrow \{\rho_j \in G_i \mid v_j = 1\}$
10 $\quad\quad\quad \pi^- \leftarrow \{\rho_j \in G_i \mid v_j = 0\}$
11 $\quad\quad\quad$ **if** `MutexFree(`π^+`,`T`)` **and** $\pi^+ \subseteq \Omega_T$ **then**
12 $\quad\quad\quad\quad L \leftarrow \{\neg V_{pred}(\rho, T) \mid \rho \in \pi^+\}$
13 $\quad\quad\quad\quad L \leftarrow L \cup \{V_{pred}(\rho, T) \mid \rho \in \pi^- \cap \Omega_T\}$
14 $\quad\quad\quad\quad$ **if** $L \neq \emptyset$ **then**
15 $\quad\quad\quad\quad\quad \varphi \leftarrow \varphi \wedge \underbrace{\left(\bigvee_{\ell \in L} \ell\right)}_{r_i(\mathbf{v})}$
16 $\quad \varphi_s^T \leftarrow \varphi$
17 **end**

If at least one predicate in π^+ is not coded for as a goal, then this tells us that this predicate cannot be achieved by any plan of makespan less than or equal to T; thus, this truth assignment and its corresponding clause should be ignored. This is the reason for the check $\pi^+ \subseteq \Omega_T$ on line 11.

At lines 12 and 13 we gather the set of literals for the clause. We negate variables corresponding to predicates in π^+ and leave as positive literals the variables corresponding to predicates in $\pi^- \cap \Omega^T$. If the truth assignment is made, then all literals will be false and the clause will be violated. Notice how we exclude any predicates that are in $\pi^- \setminus \Omega_T$ because they are unreachable at the final level and cannot be achieved by any plan of makespan less than or equal to T, consequently they are fixed to false. The check at line 14 handles the special case where the truth assignment assigns false to all predicates in G_i and none of these predicates are coded for as goals in φ_h^T. This results in an empty clause that is always violated; therefore, we need not include it in the encoding since it will not affect the minimization. Finally, at line 15 the clause is added with weight set to the residual utility of the truth assignment to the predicates in G_i.

We implemented this procedure on top of the SATPLAN06 system. We modified the parser and lexer to read in a specification of action costs and a description of the utility function. To represent the utility function we use a tree-like structure called a UCP-net[1] (Boutilier, Bacchus, and Braf-

[1]We ignore the *ceteris paribus* condition that ensures the dominance property at each node because we compile the function to WPMax-SAT where it is not exploited by the solver.

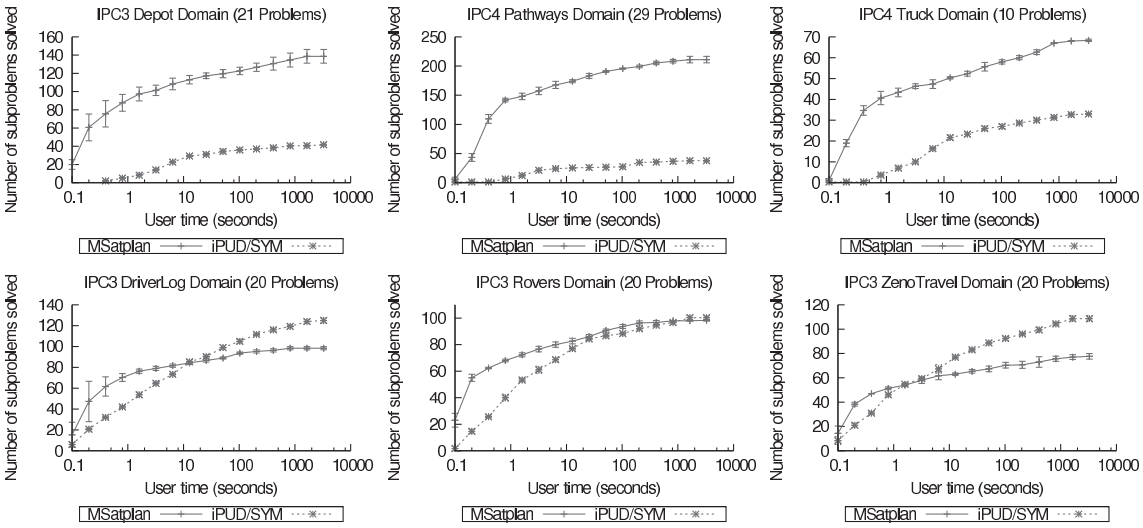

Figure 1: Comparison of MSATPLAN with IPUD/SYM on the number of subproblems solved as each system is given more time. Each point shows the mean number of subproblems solved within the corresponding time limit. Error bars show $\pm\sigma$, calculated over three runs. A lack of an error bar indicates that there was no observed variance in that measurement.

man 2001). Each node in this tree represents the value of a single goal predicate from $\mathcal{G}$. If there is a directed link from node Y to node X then Y is a *parent* of X, and we denote the set of parents of a node X as $\mathsf{Parents}(X)$. Each node X also contains a *conditional preference table* which we specialize to a tabular representation of a pseudo-Boolean function $f_X : \mathbb{B}^{|\mathsf{Parents}(X)|+1} \rightarrow \mathbb{N}_0$ since all variables in the tree are boolean-valued. f_X represents X's contribution to the utility of a plan dependent on its value and those of its parents.

Experimental Results

We compared our system MSATPLAN with IPUD using a *one state change* (1SC) encoding over a collection of problems derived from past International Planning Competition (IPC) benchmarks: *DriverLog*, *Depot*, *ZenoTravel* and *Rovers* from IPC3; and *Truck* and *Pathways* from IPC4. We wrote a Java program to parse untyped STRIPS problems and attach randomly generated action costs and utility functions over goals. This process is described as follows. For each action in a domain, a cost is generated randomly according to a discrete uniform distribution over the values $\{x \in \mathbb{N} \mid 1 \leq x \leq 30\}$. For each problem, a random utility function is generated in two stages. Firstly, a DAG with a restriction on heuristic induced width is randomly generated according to a method that is used to generate random Bayesian networks (Ide et al. 2004). Given this DAG, a conditional preference table (CPT) is generated for each node. For each truth assignment T for a node X and its parents, if $T[X]$ is false then an entry of 0 is made in the CPT for that truth assignment; otherwise, a value is randomly generated according to a discrete uniform distribution over the values $\{x \in \mathbb{N} \mid 100 \leq x \leq 200\}$, and this is entered in the CPT for the truth assignment T. The numbers in these ranges are somewhat arbitrary; they were selected by experimenting to find values that tended to allow valid plans of increasing net benefit as the makespan was increased from 1 to 10. If the action costs are too high or the utilities too low it precludes the existence of any nonempty plan with positive net benefit; the optimal plan would be empty in these cases.

We do not use the original IPUD system because it uses a commercial linear program solver called CPLEX 10.0[2], which we did not have access to; instead, we implemented IPUD's encoding scheme by extending the 1SC encoding[3] of IPPLAN (van den Briel, Vossen, and Kambhampati 2008) and modifying it to use an open-source mixed integer program solver SYMPHONY 5.1 (Ralphs and Guzelsoy 2005). We refer to this implementation as IPUD/SYM to avoid ambiguity. We believe this remains a reasonably fair test as we do not use a commercial WPMax-SAT solver and the algorithms for SYMPHONY are published and open for inspection. MSATPLAN uses MiniMaxSat 1.0 (Heras, Larrosa, and Oliveras 2008) to solve the WPMax-SAT encodings. MiniMaxSat was chosen because of its strong performance across domains in the Max-SAT-2008 evaluation.

Since all goals are soft, a (possibly empty) plan exists at every makespan, and consequently it is not very interesting to search for the plan with smallest makespan. Instead, we split each problem up into subproblems parameterized by a makespan variable d. Each subproblem is to find a plan of makespan d that solves the original problem with optimal net benefit over all other plans of makespan d. For each problem in a domain benchmark, we derive subproblems for $d = 1, \ldots, 10$. Each planning system is given 30 minutes to solve each subproblem, and is aborted if it fails to do so.

In summary, each run of our experiment consists of gener-

[2] http://www.ilog.com/products/cplex

[3] We use the 1SC encoding instead of the G1SC one used in Do et al.'s original paper because it allows the same amount of parallelism as the SAT-based encoding that MSATPLAN extends.

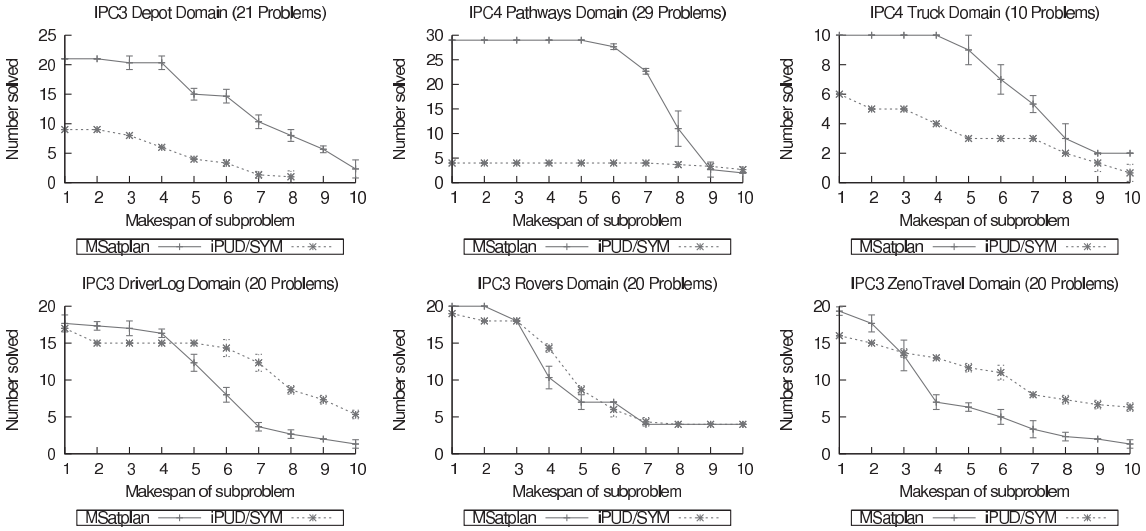

Figure 2: Comparison of the successful search depth between MSATPLAN and IPUD/SYM over six problem domains taken from IPC-3 and IPC-4. Each point shows the mean number of subproblems, of a particular makespan, that were solved by each system. Error bars show $\pm\sigma$, which is calculated over the three runs. A lack of an error bar indicates that there was no observed variance in that measurement.

ating a random utility tree and action costs for each problem in each domain, splitting each problem into 10 subproblems and attempting to solve each subproblem with both solvers. We conduct three runs using randomly generated utility trees and action costs each time. Where appropriate, we computed the sample mean and sample variance over these different runs to investigate how the utility tree and action costs affect solution times. All experiments were conducted on a Linux machine with an Intel 2.4 GHz quad core CPU (although neither program is multithreaded) and 2 GB of memory; however, we limit the memory resource available to each program to 1.5 GB to reduce paging.

When examining these results, it should be noted that IPUD/SYM uses a translation step, taken from the *Fast Downward Planner* (Helmert 2006), that converts PDDL2.2 (Edelkamp and Hoffmann 2004) files to the SAS$^+$ formalism (Backstrom and Nebel 1995), which is used to represent multi-valued planning tasks. Inspection of several experiments revealed that for problems 06–30 from the Pathways domains, this translation step almost always failed to complete within 30 minutes. We found that this was also the case for the original translation tool applied to the original PDDL2.2 files taken from the Pathways domain from IPC-4; the reason for this failure remains unclear.

The data were used to calculate for each run, and exponentially increasing time limits, the number of subproblems solved within those time limits. The results for each time limit were then averaged over the three runs. Figure 1 shows a plot of the results. MSATPLAN shows a clear advantage over the 1SC encoding of IPUD/SYM for the Depot and Truck domains by solving 233% and 107% more subproblems respectively. MSATPLAN also performs very well on the Pathways domain; however, comparing this with IPUD/SYM is not possible because of a problem with the translation step as described above. IPUD/SYM solves 27%, 40%, and 2% more problems than MSATPLAN for the DriverLog, ZenoTravel and Rovers domains respectively.

How the two systems compare over the range of fixed makespans can be seen in Figure 2. The percentage of subproblems that are solved drops off with an increase in makespan, as one would expect; however, the gradient of this decrease differs quite substantially between domains. For the Depot and DriverLog domains, the decrease is smooth with a reasonably consistent gradient for both systems. For the Truck and Pathways domains, there is very little decrease initially and almost all the decrease occurs within 2–4 makespans for MSATPLAN.

How each system performed on an individual problem can be seen in Figure 3. It is somewhat surprising that MSATPLAN's performance is maintained across the Pathways domain. For other domains the performance tends to degrade as the problem number increases, since these are considered harder problems. It is also surprising that MSATPLAN solves a large number of subproblems from the higher problem numbers in the DriverLog domain, but not for problem numbers 04–10, which would normally be considered easier to solve.

Discussion

The leading approaches to finding an optimal solution to a WPMax-SAT problem usually involve a depth-first branch-and-bound or branch-and-cut search; the same can be said for finding an optimal solution to an integer program. These algorithms calculate an exact upper bound *ub* on the cost of the optimal solution, which is the cost of the best model found so far in the search, and an underestimation of best cost that can be achieved by extending the current partial assignment to a model. A branch of the search tree is pruned

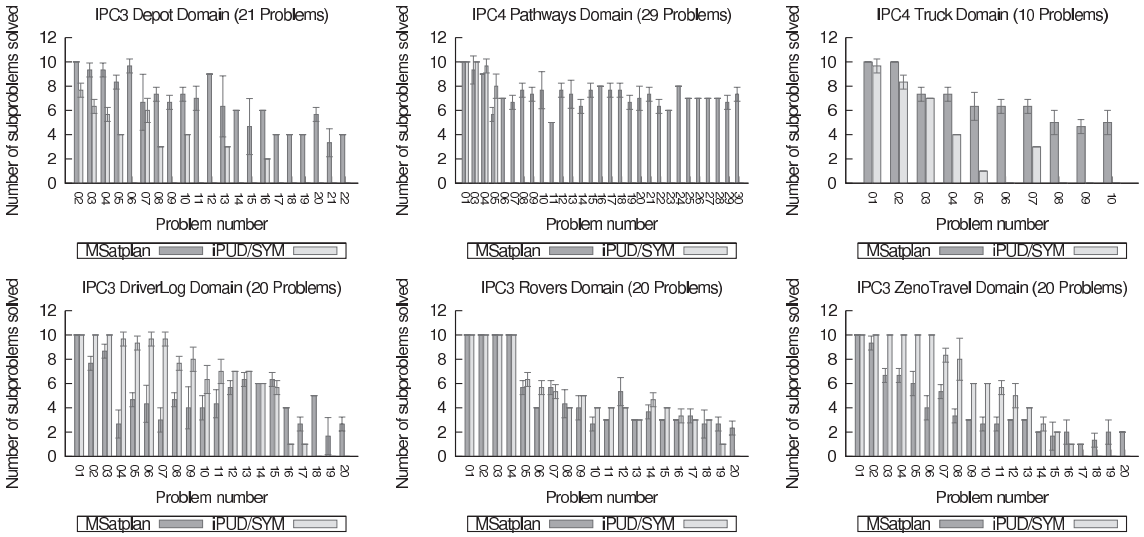

Figure 3: Comparison of MSATPLAN with IPUD/SYM on each problem over a total of six domains from IPC-3 and IPC-4. The height of a bar indicates the mean number of subproblems solved by that system for that problem. Error bars indicate $\pm\sigma$, calculated over three runs. A lack of an error bar for a column indicates that there was no observed variance in that measurement.

when its lower bound is found to be greater than or equal to *ub*. Soft clauses can also be promoted to hard clauses during the search if their weight is found to be greater than *ub*, which allows us to use this clause together with other hard clauses for efficient procedures such as unit-propagation.

If we incrementally increase the plan makespan and find optimal solutions for each one, we can trivially extend an optimal solution for a makespan of $d-1$ to a solution with makespan d by executing appropriate NOOP actions between layers $d-1$ and d. From this, we can produce a nontrivial lower bound on the best net benefit obtainable at makespan d and thus produce a nontrivial value for *ub* for the problem at makespan d. Hopefully, this will increase the number of pruning and promotion events, as described above, that occur early on in the branch-and-bound search.

It is worth noting that these branch-and-bound searches keep track of the best solution encountered during search. This can produce a solution to a planning problem at any time – before the first non-trivial solution is found, this would return the empty plan. In our experiments we terminated searches that lasted for longer than 30 minutes and recorded no solution. Alternatively, after 30 minutes, we could have returned the best solution found so far in the branch-and-bound search and compared this to the results obtained by a heuristic search planner; however, MiniMaxSat did not support this feature.

Related Work

Using Max-SAT to model hard and soft constraints in optimization problems has been studied for Steiner trees (Jiang, Kautz, and Selman 1995), although this did not use the partial weighted Max-SAT variant, so hard constraints were modelled with very high numeric weights.

There have been several recent improvements to the planning as satisfiability paradigm. Londex constraints generalize mutex links to indicate mutual exclusion across time steps, which reduces planning time in many benchmark problems (Chen, Zhao, and Zhang 2007). $\exists$-step semantics have been incorporated into encodings for planning as satisfiability (Wehrle and Rintanen 2007). This style of semantics allows parallel execution of operators if at least one total ordering of them is consistent, which allows for shorter parallel plan lengths. The *generalized one state change* (G1SC) encoding, found in IPUD, provides this level of parallelism (van den Briel, Vossen, and Kambhampati 2008). Lifted encodings for planning as satisfiability that allow more scope for parallelism have been investigated (Robinson et al. 2008).

Although preferences are receiving an increasing amount of attention from the planning community, there has been little work examining how the planning as satisfiability paradigm can handle preferences. SATPLAN(P) handles quantitative and qualitative preferences using a custom DPLL solver that branches according to a preference order (Giunchiglia and Maratea 2007). In particular, for a problem with quantitative preferences, the value of the optimization function is encoded as a sequence of bits, and the preference order prefers higher/lower order bits to be set depending on whether the optimization function is to be maximized/minimized. Their experimental results only cover the cases where either (1) each goal is soft with a utility of 1 and there are no action costs or (2) all goals are hard and all actions have cost 1. Thus, it remains to be seen how their approach scales for optimization functions that are more flexible.

The heuristic search planners AltAlt and Sapa have been extended to solve PSP problems (van den Briel et al. 2004); the former heuristically selects a set of goals to plan for, and

Sapa uses an A* search with a heuristic that estimates the extra net benefit available from extending the current partial plan. Sapa has also been extended to handle goal utility dependencies (Do et al. 2007). Its heuristic calculation first greedily constructs a relaxed plan that supports all reachable goals; it then encodes a problem in IP to find the most beneficial plan contained in the relaxed plan.

An alternative method for selecting goals to plan for is to represent an abstracted part of the planning problem as an orienteering problem (Smith 2004). This is motivated by oversubscription planning problems relevant to the Mars rover where the cost of achieving goals depends strongly on the order in which they are achieved. The aim is to model the cost dependencies between achieving goals, but this ignores the idea of goal utility dependencies. If such goal dependencies are sufficiently localized so that none exist between 'cities' in the orienteering graph, then a system such as IPUD or MSATPLAN might find use in producing reward estimates for each city provided that the subproblems are small enough and the computation time constraints are sufficiently generous.

Concluding Remarks

We have demonstrated a system, MSATPLAN, for solving planning problems with goal utility dependencies using an optimization variant of propositional satisfiability, known as weighted partial Max-SAT. This system is guaranteed to produce plans that are optimal up to a given makespan. We compared our implementation against a successful integer programming based encoding, implemented as IPUD/SYM, using past benchmark problems from the International Planning Competition. Our results showed that MSATPLAN is competitive with IPUD/SYM, solving as many as 107% and 233% more subproblems for the Truck and Depots domains; it also demonstrated consistently good performance on the Pathways domain. When IPUD/SYM outperformed MSATPLAN in our experiments, it did so by a smaller margin: 27%, 40% and 2% more problems solved for the DriverLog, ZenoTravel and Rovers domains.

We obtained our results for MSATPLAN using a general-purpose weighted partial Max-SAT solver to find a plan. An area for future work is to investigate how we can specialize the algorithms used in these solvers to exploit the regular structure that is found in plan encodings.

Acknowledgements

We would like to thank Ashutosh Mahajan, Malte Helmert and Menkes van den Briel for their valuable assistance in working with various software programs mentioned in this paper. This work was supported by the UK Engineering and Physical Sciences Research Council under grant EP/P50385X/1.

References

Bacchus, F., and Grove, A. 1995. Graphical models for preference and utility. In *Proc. UAI*.

Backstrom, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11:625–655.

Benton, J.; Do, M.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence* 173:562–592.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial intelligence* 90:281–300.

Boutilier, C.; Bacchus, F.; and Brafman, R. 2001. UCP-Networks: A directed graphical representation of conditional utilities. In *Proc. UAI*.

Chen, Y.; Zhao, X.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In *Proc. IJCAI*.

Do, M.; Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. Planning with goal utility dependencies. In *Proc. IJCAI*.

Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Universität Freiburg, Institut für Informatik.

Giunchiglia, E., and Maratea, M. 2007. Planning as satisfiability with preferences. In *Proc. AAAI*.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient weighted Max-SAT solver. *JAIR* 31:1–32.

Ide, J.; Cozman, F.; Ramos, F.; et al. 2004. Generating random Bayesian networks with constraints on induced width. In *Proc. ECAI*.

Jiang, Y.; Kautz, H.; and Selman, B. 1995. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proc. of the Intl. Joint Workshop on Artificial Intelligence and Operations Research*.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. IJCAI*.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. Satplan: Planning as satisfiability. In *Booklet of 5th IPC*.

Ralphs, T., and Guzelsoy, M. 2005. The SYMPHONY callable library for mixed integer programming. In *Proc. of the Conf. of the INFORMS Computing Society*.

Robinson, N.; Gretton, C.; Pham, D.; and Sattar, A. 2008. A compact and efficient SAT encoding for planning. In *Proc. ICAPS*.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proc. ICAPS*.

van den Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective approaches for partial satisfaction (over-subscription) planning. In *Proc. AAAI*.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2008. Loosely coupled formulations for automated planning: an integer programming perspective. *JAIR* 31:217–257.

Wehrle, M., and Rintanen, J. 2007. Planning as satisfiability with relaxed $\exists$-step plans. In *Proc. of the Australian Joint Conf. on Artificial Intelligence*.