Exploiting MaxSAT for Preference-Based Planning

Farah Juma and **Eric I. Hsu** and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
{fjuma, eihsu, sheila}@cs.toronto.edu

Abstract

In this paper, we explore the application of partial weighted MaxSAT techniques for preference-based planning (PBP). To this end, we develop a compact partial weighted MaxSAT encoding for PBP based on the SAS$^+$ formalism. Our encoding extends a SAS$^+$ based encoding for SAT-based planning, SASE, to allow for the specification of simple preferences. To the best of our knowledge, the SAS$^+$ formalism has never been exploited in the context of PBP. Our MaxSAT-based PBP planner, MSPLAN, significantly outperformed a STRIPS-based MaxSAT approach for PBP with respect to running time, solving more problems in a few cases. Interestingly, when compared to SGPlan$_5$ and HPLAN-P, two state-of-the-art heuristic search planners for PBP, MSPLAN consistently generated plans with comparable quality, slightly outperforming at least one of these two planners in almost every case. For some problems, MSPLAN was competitive with HPLAN-P with respect to running time. Our results illustrate the effectiveness of our SASE based encoding and suggests that MaxSAT-based PBP is a promising area of research.

1. Introduction

Many real-world planning problems consist of both a set of mandatory goals and an additional set of desirable plan properties. The degree of satisfaction of these desirable properties, or plan *preferences*, determines the quality of a plan. Preference-based planning (PBP) (e.g., (Baier and McIlraith 2008)) extends the well-known classical planning problem in order to generate plans that achieve problem goals while maximizing the satisfaction of other preferred properties of the plan. In so doing, they allow a planner to generate plans of high quality, often under situations with conflicting preferences.

PBP has been the subject of substantial research in recent years. The 2006 International Planning Competition (IPC-2006) initiated a track on this topic which resulted in the extension of the standardized Planning Domain Description Language (PDDL) to support the specification of preferences (Gerevini et al. 2009). In PDDL3, desirable properties of a plan are expressed as preference formulae. These formulae may describe properties of the final state as well as properties that hold over intermediate states visited during plan execution. The relative importance associated with not violating these preference formulae is reflected in a metric function, a weighted linear combination of preferences whose violation is minimized by the planner. At IPC-2008, this family of planning problems was extended to include action costs. The objective of these so-called *net benefit* problems is to maximize the sum of the utilities of the goals and preferences that have been achieved, minus total costs. Action costs can be incorporated into our partial weighted MaxSAT-based PBP approach and it is something that we are exploring, but do not address it in this paper.

To date, the most effective techniques for PBP have been based on heuristic search (e.g., *Yochan*$^{\mathcal{PS}}$ (Benton, Kambhampati, and Do 2006), SGPlan$_5$ (Hsu et al. 2007), and HPLAN-P (Baier, Bacchus, and McIlraith 2009)). There have also been several planners that have used SAT, CSP, or Answer Set solvers (e.g., SATPLAN(P) (Giunchiglia and Maratea 2007), PREFPLAN (Brafman and Chernyavsky 2005), CPP (Tu, Son, and Pontelli 2007)). Most recently, Giunchiglia and Maratea (2010) explored a partial weighted MaxSAT-based approach to solving PBP problems by modifying a version of SATPLAN (Kautz 2006). For the purposes of this paper, we refer to this as GM. While all of these latter systems show promise, performance has generally been inferior to heuristic search.

In this paper, we characterize the problem of computing preference-based plans as a partial weighted MaxSAT problem. A major focus of our work is on how to construct an effective encoding. To this end, we propose a SAS$^+$ based (Bäckström and Nebel 1995) encoding of PBP as MaxSAT that is both compact and correct. Our encoding builds on the success of Huang, Chen, and Zhang (2010)'s SASE, a SAS$^+$ based encoding recently developed for SAT-based planning. To the best of our knowledge, the SAS$^+$ formalism has never been used in the context of PBP despite its effectiveness in classical planning. Exploiting our characterization of PBP as a partial weighted MaxSAT problem, we develop a system called MSPLAN that employs our SASE based encoding.

We experimentally evaluated our system by comparing it to Giunchiglia and Maratea's GM on *Simple Preferences* problems from IPC-2006. MSPLAN consistently outperformed GM with respect to running time, in some cases by an order of magnitude, solving some problems that GM could not solve. In all cases, plan quality was comparable. We also compared the performance of MSPLAN, run with two different MaxSAT solvers, to state-of-the-art heuris-

tic search planners for PBP. MSPLAN generated plans of comparable plan quality, sometimes slightly out- or underperforming the best of the heuristic search planners we employed. However, consistent with our expectations, MSPLAN was not able to solve as many problems as the heuristic search planners. While in some instances, one of the MSPLAN systems significantly outperformed HPLAN-P with respect to running time, the heuristic search planners were generally faster. Analysis showed that a significant part of MSPLAN's running time was attributed to the incremental construction of makespans, rather than the search for a solution. Given the consistent quality of MSPLAN solutions, consistent superior performance to GM, and the variability in the performance of all of these systems, we deem our SASE based encoding to be effective compared to a STRIPS encoding, and the use of MaxSAT and SAS$^+$ encodings for PBP to be promising areas of research.

2. PBP as Partial Weighted MaxSAT

In this section, we overview the correspondence between PBP and partial weighted MaxSAT. For the purposes of this paper, we restrict our attention to simple preferences – preferences over properties of the final state of the plan.

Definition 1 (Planning domain) Let X be a set of state variables. A planning domain is a tuple $D = (S, A, \gamma)$, where S is the set of all possible states; A is the set of actions; and γ is the transition function. A state $s \in S$ is a set of assignments to all of the variables in X. An action $a \in A$ is described by a tuple $(\mathit{pre}(a), \mathit{eff}(a))$, which denotes the action's preconditions and effects. A transition $\gamma(s, a)$ modifies the values of the state variables mentioned in $\mathit{eff}(a)$.

Definition 2 (Simple PBP problem) A simple PBP problem is a tuple $P = (D, s_I, s_G, \mathit{Pref}, W)$, where $D = (S, A, \gamma)$ is the planning domain; $s_I \in S$ is the (complete) initial state; s_G represents the goal and is a partial set of assignments to the state variables; and *Pref*, the preferences, is a partial set of assignments to the state variables. Optionally, each element in *Pref* may have a positive weight associated with it, W: *Pref* $\rightarrow \mathbb{R}_{\geq 0}$, to capture the relative importance of preferences.

We now define the partial weighted MaxSAT problem. Let ϕ denote a CNF propositional formula over a set V of boolean variables and let $\{C_1, \ldots, C_m\}$ denote the clauses of ϕ.

Definition 3 (MaxSAT and (Partial) Weighted MaxSAT) The *MaxSAT problem* is to find an assignment of values for V that maximizes the number of satisfied clauses in ϕ. Given a weight w_i for each clause C_i in ϕ, the *weighted MaxSAT problem* is to find an assignment of values for V that maximizes the total weight of the satisfied clauses in ϕ. When some clauses in ϕ are designated as *hard* clauses and other clauses in ϕ are designated as *soft* clauses and we are given a weight w_i for each *soft* clause C_i in ϕ, the *partial weighted MaxSAT problem* is to find an assignment of values for V that satisfies all of the designated *hard* clauses in ϕ and maximizes the total weight of the satisfied *soft* clauses in ϕ.

If *Pref*, the preferences in a simple PBP problem, are encoded as soft clauses, and s_I, s_G, and D are encoded as hard clauses, and weights drawn from W are assigned to the soft clauses to indicate the relative importance of the preferences, then from Definition 3, it follows that finding a solution to the resulting partial weighted MaxSAT problem is equivalent to finding a plan for the original PBP problem that achieves all of the hard constraints while maximizing the weight of the satisfied preferences. How we actually encode these clauses is a challenge and is one of the contributions of this paper.

3. Preference-Based Planning

3.1 A SAS$^+$ Based Encoding

The SAS$^+$ formalism (Bäckström and Nebel 1995) has been increasingly exploited in the context of *classical* planning. Unlike a STRIPS-based encoding, which consists of actions and facts and represents a state using a set of facts, a SAS$^+$ based encoding consists of transitions and multi-valued state variables and represents a state using a set of assignments to all of the state variables. A transition represents a change in the value of a state variable. For example, consider a transportation domain in which trucks can move packages between locations with certain restrictions. To represent the possible locations of a truck, a STRIPS-based encoding might include a fact for each such location (e.g., `(at truck1 loc1)`, `(at truck1 loc2)`). On the other hand, a SAS$^+$ based encoding might include a state variable `truck1` whose domain consists of all of the possible truck locations. Moving `truck1` from `loc1` to `loc2` would be represented by an action in a STRIPS-based encoding. However, in a SAS$^+$ based encoding, this would be represented by a transition that changes the value of the state variable `truck1` from `loc1` to `loc2`. In general, an action might result in changes in the values of multiple state variables. In a SAS$^+$ based encoding, this would be represented by multiple transitions. Because the number of transitions in an encoding based on the SAS$^+$ formalism is normally much less than the number of actions in a STRIPS-based encoding, the search space in a transition-based encoding is smaller than the search space in an action-based encoding (Huang, Chen, and Zhang 2010). The compactness of the SAS$^+$ formalism along with the rich structural information that can be derived from it are the major advantages of this formalism. As demonstrated by its use in (Helmert 2006), many state-of-the-art planners have been adopting SAS$^+$ based encodings.

The major limitation of previous STRIPS-based SAT encodings for classical planning problems has been the large number of clauses that are produced. The SAS$^+$ formalism was recently used for the first time in the context of a SAT-based classical planning approach with an impressive reduction in the size of the SAT encoding (Huang, Chen, and Zhang 2010). However, to the best of our knowledge, the SAS$^+$ formalism has never been used in the context of PBP despite its effectiveness in classical planning. We explore this here.

Translating PBP Problems to SAS$^+$ We consider PBP problems from the *Simple Preferences* track of IPC-2006. STRIPS problems can be compiled into the SAS$^+$ representation using the parser created for the Fast Downward plan-

ner (Helmert 2006). Since IPC-2006 *Simple Preferences* problems are non-STRIPS problems, we first need to compile them into STRIPS problems. We do so using the compilation technique developed in (Giunchiglia and Maratea 2010). The compilation proceeds as follows. For each simple preference, a so-called *dummy* action is created. The precondition of this dummy action is the simple preference itself and the effect of this action is a *dummy* literal that represents the simple preference. This dummy literal is then added to the definition of the goal for the problem instance and the definition of the simple preference is then removed from the problem. The execution of a dummy action indicates that the corresponding simple preference has been satisfied. For example, consider the following preference that is expressed in PDDL3:

```
(preference time
  (or (delivered pck1 loc3 t1)
      (delivered pck1 loc3 t2)))
```

The `time` preference specifies that `pck1` should be delivered to `loc3` at `t1` or `t2`. The compilation technique replaces this preference with the following dummy action:

```
(:action dummy-time
 :parameters ()
 :precondition (or (delivered pck1 loc3 t1)
                   (delivered pck1 loc3 t2))
 :effect (dummy-goal-time))
```

The literal `dummy-goal-time` represents the satisfaction of the `time` preference. After introducing dummy actions for each simple preference, the resulting problems are compiled into STRIPS using Gerevini and Serina's ADL2STRIPS tool. Note that we must also maintain the information about which goals in the resulting STRIPS problems correspond to simple preferences. These STRIPS problems can then be translated into the SAS$^+$ formalism.

After translating these STRIPS problems into SAS$^+$, there is a SAS$^+$ goal variable that corresponds to each dummy literal (i.e., simple preference). We will refer to these SAS$^+$ variables as preference variables. We will use s_P to denote the set of desired assignments to preference variables in the final state. As such, $s_P(x) = p$ indicates that in the final state, the desired value of the preference variable x is p.

Encoding the Clauses To encode a SAS$^+$ based PBP problem as a partial weighted MaxSAT problem, we modify the SAS$^+$ based SAT encoding, SASE, first proposed in (Huang, Chen, and Zhang 2010). SASE is made up of transition variables U, action variables V, and eight classes of clauses. We show these classes of clauses below. Note that N denotes the number of time steps in the plan, $\tau(x)$ denotes the set of possible transitions for x, $M(a)$ denotes the transition set of action a, R is the set of all prevailing transitions, $\delta_{f \to f'}$ represents a change in the value of x from f to f', $U_{x,f,f',t}$ is a transition variable that indicates that variable x changes from the value of f to a value of f' at time step t, and $V_{a,t}$ is an action variable that indicates that the action a is executed at time t.

1. Initial state - $(\forall x, s_I(x) = f)$: $\bigvee_{\forall \delta_{f \to g} \in \tau(x)} U_{x,f,g,1}$
2. Goal - $(\forall x, s_G(x) = g)$: $\bigvee_{\forall \delta_{f \to g} \in \tau(x)} U_{x,f,g,N}$
3. Transition's progression - $(\forall \delta^x_{f \to g} \in \tau$ and $t \in [2, N])$: $U_{x,f,g,t} \to \bigvee_{\forall \delta^x_{f' \to f} \in \tau(x)} U_{x,f',f,t-1}$
4. Transition mutex - $(\forall \delta_1 \forall \delta_2$ such that δ_1 and δ_2 are transition mutex): $U_{\delta_1,t} \to \neg U_{\delta_2,t}$
5. Existence of transitions - $(\forall x \in X)$: $\bigvee_{\forall \delta \in \tau(x)} U_{\delta,t}$
6. Composition of actions - $(\forall a \in O)$: $V_{a,t} \to \bigwedge_{\forall \delta \in M(a)} U_{\delta,t}$
7. Action's existence - $(\forall \delta \in \tau \backslash R)$: $U_{\delta,t} \to \bigvee_{\forall a, \delta \in M(a)} V_{a,t}$
8. Non-interference of actions - $(\forall a_1 \forall a_2$ such that $\exists \delta$, $\delta \in T(a_1) \cap T(a_2)$ and $\delta \notin R) : V_{a_1,t} \to \neg V_{a_2,t}$

These clauses encode the initial state, the goal state, the transitions that are allowed to occur at various time steps, the relationship between transitions and actions, and the fact that mutually exclusive actions cannot be executed simultaneously. After finding a sequence of transitions that achieves the goal, from the initial state, a corresponding action plan is found.

The key challenge is determining how to encode the preferences. We modify SASE to handle preference variables (i.e., variables that we would *like* to achieve a certain value in the final state) in addition to goal variables (i.e., variables that *must* achieve a certain value in the final state). To this end, we create a new class of clauses that represents the preferences. Specifically, for each preference variable x, we create a clause that is the disjunction of all transitions which result in the desired value for x in the final state. Using the notation described above, this new class of clauses can be defined as follows:

Preferences - $(\forall x, s_P(x) = p)$: $\bigvee_{\forall \delta_{f \to p} \in \tau(x)} U_{x,f,p,N}$

It is important to note that once a dummy action is executed, the dummy goal literal it produces persists indefinitely, i.e., no action deletes a dummy goal literal. Because the execution of a dummy action is meant to indicate that a simple preference is satisfied in the *final* state of the plan, we must ensure that once a dummy action has been executed, no other action that can invalidate the precondition of this dummy action can occur after it. As such, we restrict the execution of dummy actions, denoted by O_{dummy}, to the final time step of the plan by only defining dummy action variables that correspond to this time step, i.e.,

Dummy action variables - $V_{a,N}, \forall a \in O_{dummy}$

Because preference variables do not necessarily have to achieve their desired value in the final state, unlike hard goal variables, we treat all preference clauses as soft clauses. We treat the clauses that encode the initial state, goal state, and the planning domain, as hard clauses.

Weighting the Clauses We assign weights to all of the soft clauses using the PDDL3 metric function. This metric function is a linear function defined over simple preferences and is used to determine the quality of a plan. An example of

a PDDL3 metric function over two preferences named `time` and `loc` is shown below:

```
(:metric minimize(+
                  (* 10 (is-violated time))
                  (*  5 (is-violated loc))))
```

Note that the `is-violated` function returns 1 if the preference with the given name does not hold in the final state of the plan and returns 0 otherwise. This metric function indicates that satisfying the `time` preference is twice as important as satisfying the `loc` preference. An IPC-2006 *Simple Preferences* task can thus be viewed as the problem of finding a plan that satisfies all of the hard goals while minimizing the total weight of the preferences that are not satisfied (i.e., the task is to minimize the metric function). Since each simple preference is represented by one soft clause in our encoding, we assign a weight to this soft clause based on the weight assigned to this preference in the PDDL3 metric function from the original IPC-2006 problem instance. A similar approach was used to weight clauses in (Giunchiglia and Maratea 2010).

3.2 Planning with MaxSAT

With our SAS$^+$ based partial weighted MaxSAT encoding for PBP in hand, the next step is to determine how to find a plan using such an encoding. As in SAT-based planning, an incremental construction of makespans is required, i.e., for a specific value n for the makespan (the number of time steps in the plan), we must encode a given PBP problem into our SAS$^+$ based partial weighted MaxSAT encoding with a makespan of n, attempt to solve the problem using a partial weighted MaxSAT solver, and continue on in this manner, trying increasing values of n, until a solution is found. However, determining when a solution is found is not trivial, as discussed in the next subsection.

Stopping Conditions Because the task of PBP involves finding a high-quality plan, and not just a plan with the minimum number of time steps (as in SAT-based planning), determining when to stop trying increasing values for the makespan is more difficult. One possibility we consider is to stop trying increasing values for the makespan after the first satisfiable partial weighted MaxSAT formula is found. The solution to this formula corresponds to a plan with optimal makespan and optimal plan quality for that particular makespan. However, this plan is not guaranteed to be globally optimal. Consequently, there could still be a plan with a larger number of time steps but with better plan quality. We also consider the case where a time limit is given and an incremental construction of makespans is carried out until the allotted time expires, at which point the plan returned is the plan with the best quality that was found during the given amount of time. We will refer to this algorithm as the BEST algorithm.

Properties of the Plan We can show that for any fixed makespan, our approach is guaranteed to return a solution with optimal plan quality with respect to that particular makespan and furthermore, we can show that when restricted to plans with makespan bounded by k, our approach is guaranteed to return a solution that is k-optimal.

Lemma 1 *For any fixed makespan n, the solution to the partial weighted MaxSAT problem encoded with makespan n yields a plan with optimal quality with respect to the set of all plans with makespan n, if such a plan exists.*

Proof: Follows directly from the definition of the partial weighted MaxSAT problem (Definition 3). □

From Lemma 1, we can conclude that any plan P that is returned by our approach has optimal quality with respect to the set of all plans with the same makespan as P.

In certain cases, we may want to restrict our attention to plans with a makespan that is bounded by some value.

Definition 4 (k-Optimality) We can say that a partial weighted MaxSAT-based PBP algorithm is k-optimal if it is always able to find a plan that is optimal, in terms of quality, with respect to the set of all plans with makespan $i \leq k$.

Theorem 1 *If the search is restricted to plans with makespan bounded by k and the* BEST *algorithm is run long enough for the PBP problem to be encoded into a partial weighted MaxSAT formula using each makespan $i \leq k$, then the* BEST *algorithm is k-optimal.*

Proof: For each makespan $i \leq k$, the PBP problem will be encoded into a partial weighted MaxSAT formula with makespan i and if a solution to this formula is found, the quality of the resulting plan will be determined. From Lemma 1, each such plan will have optimal quality with respect to the set of all plans with makespan i. Now, the result follows since the BEST algorithm returns the plan with the best quality among the plans that were found. □

4. Implementation and Evaluation

We implemented our planner, MSPLAN, by extending the SASE planner. In order to compare the performance of MSPLAN to GM, we tried two of the partial weighted MaxSAT solvers which GM was evaluated with, namely, MiniMaxSAT v1.0 (Heras, Larrosa, and Oliveras 2007) and SAT4J v2.1 (Berre and Parrain 2010). We also attempted a comparison using MSUncore (Marques-Silva 2009), another partial weighted MaxSAT solver, but were unable to get a version of the system from the developers that would run on our hardware.

Most partial weighted MaxSAT solvers do not allow real-valued weights to be assigned to clauses but the PDDL3 metric function does allow for real-valued weights to be assigned to preferences. Thus, we must multiply real-valued weights in our encoding by an appropriate power of 10 in order to remove decimals from the weights. Additionally, many partial weighted MaxSAT solvers require a special weight to be specified for hard clauses in addition to weights for soft clauses. Following the convention used to specify this special weight, we assign a weight to each hard clause that exceeds the sum of the weights of all of the soft clauses.

Our evaluation of MSPLAN was motivated by two objectives. Specifically, we wanted to: (1) compare the performance of our planner to a previous partial weighted MaxSAT-based approach; and (2) compare our planner to state-of-the-art heuristic search planners for PBP. In doing so, we also hoped to gain some insight into the impact of the

underlying MaxSAT partial weighted MaxSAT solver on the performance of MSPLAN.

To support comparison with GM, the domains we evaluated were limited to those used in (Maratea 2010). Four domains from the IPC-2006 *Simple Preferences* track were used in our evaluation: trucks, storage, pathways, and openstacks. The trucks and openstacks domains contain both simple preferences and hard goals. However, the pathways and storage domains contain only simple preferences and it is generally not possible to satisfy all of these preferences. MSPLAN requires STRIPS-encodings of these problem instances as input. Since such encodings were also used in (Maratea 2010), we were able to obtain the problems from the *Simple Preferences* track which Giunchiglia and Maratea have been able to compile into STRIPS. The trucks and storage domains consist of 7 problems, the pathways domain consists of 20 problems, and the openstacks domain consists of 1 problem.

We compared the performance of MSPLAN to GM. GM uses a STRIPS-based partial weighted MaxSAT encoding as opposed to a SAS^+ based encoding. We also compared the performance of MSPLAN to that of the top heuristic search planners for PBP from IPC-2006, namely, $SGPlan_5$ (Hsu et al. 2007) and HPLAN-P (Baier, Bacchus, and McIlraith 2009). $SGPlan_5$ was the winner of all of the IPC-2006 *Simple Preferences* tracks. HPLAN-P did not formally compete in this track and came in 2nd place in the *Qualitative Preferences* track. Nevertheless, experiments performed in (Baier, Bacchus, and McIlraith 2009) show that HPLAN-P would have outperformed all entrants in the *Simple Preferences* track, other than $SGPlan_5$.

All of our experiments were performed on an AMD Opteron 1GHz processor. The memory usage in our experiments did not exceed 1GB. We ran our experiments with a timeout of 60 minutes. We were not able to obtain a copy of GM. As such, the results for these experiments are taken from (Maratea 2010). Different machines have thus been used in our comparison. In (Maratea 2010), experiments were performed on a Pentium IV 3.2GHz processor with 1GB of RAM, a faster machine than ours. Note that (Maratea 2010) gives only the time required for different partial weighted MaxSAT solvers to find a solution to the first satisfiable partial weighted MaxSAT formula. The total amount of time required to find a solution plan does not appear in (Maratea 2010).

Table 1 shows the performance of MSPLAN compared to GM when evaluating the time required for the solver to find a solution to the first satisfiable partial weighted MaxSAT formula when using MiniMaxSAT v1.0 and SAT4J v2.1 as the underlying solvers. The results show that for all problem instances in the trucks and storage domains which could be solved using both GM and MSPLAN, the time required to find a solution to the first satisfiable partial weighted MaxSAT formula was less, by an order of magnitude in many cases, when using MSPLAN than when using GM, regardless of the solver used. The value of the plan quality metric does not appear in (Maratea 2010) and we were unable to obtain the corresponding quality values from the authors. However, from a different encoding given to us by the authors, we were able to generate an upper bound on the quality values. This information indicated that the plan quality was comparable in all cases. This information also indicated that the number of clauses in our encoding was often significantly smaller, by a large constant factor. While this comparison is not precise, it is the best that could be done with the available data and gives an indication of the positive properties of our approach. There were problems in both the trucks and storage domains which could be solved by MSPLAN with a particular solver but could not be solved by GM when using the same solver. In fact, MSPLAN was able to optimally solve three problems in the trucks domain which GM could not solve using any of the partial weighted MaxSAT solvers evaluated in (Maratea 2010). Neither MSPLAN nor GM could generate a plan for the one problem instance in the openstacks domain. Finally, we were not able to do a direct comparison between these two planners for the pathways domain because we were not able to obtain information about the running time of GM on these problems.

	MiniMaxSAT		SAT4J	
Instance	GM	MSPLAN	GM	MSPLAN
trucks1	7.7	1.18	359.17	1.95
trucks2	308.92	44.803	-	24.868
trucks3	-	89.15*	-	446.578*
trucks4	-	128.904*	-	-
trucks5	-	652.877*	-	-
trucks6	-	-	-	-
trucks7	-	-	-	-
storage1	0.21	0.008	0.32	0.171
storage2	0.44	0.032	0.65	0.21
storage3	0.59	0.032	1.45	0.503
storage4	0.71	0.084	2.8	0.667
storage5	58.79	13.721	16.35	1.228
storage6	-	43.059	70.6	2.564
storage7	-	-	365.53	6.232

Table 1: Performance of MSPLAN compared to GM. Entries indicate the time required by MiniMaxSAT and SAT4J to find a solution to the first satisfiable partial weighted MaxSAT formula, in seconds. Dash entries indicate that the problem could not be solved during the given time. Starred entries indicate that the plan generated was optimal in terms of quality.

Table 2 shows the performance of MSPLAN compared to the two top heuristic search PBP planners from IPC-2006. In this set of experiments, we ran MSPLAN with the BEST algorithm that was described in Section 3.2i.e., the plan returned by MSPLAN was the highest quality plan found during a 60 minute period. We evaluated the total running time of all three planners and the quality of the plans found. The results show that for all but one of the problems that all three planners were able to solve, the quality of the plan generated by MSPLAN was equal to or was superior to the quality of the plan returned by at least one of $SGPlan_5$ and HPLAN-P, regardless of the partial weighted MaxSAT solver that was used. As expected, $SGPlan_5$ generated plans significantly faster than MSPLAN in all cases. However, MSPLAN solved some problems more quickly than HPLAN-P, while returning a plan of equal or better quality.

The significance of the $SGPlan_5$ comparison needs to be evaluated carefully. SGPlan has been shown to have in-

consistent performance on domains, depending on the encoding (Haslum 2007). Further, the version of SGPlan that participated in IPC-2008 was hand-tuned to the IPC problem encodings[1] and it is believed that previous versions of SGPlan were similarly hand-tuned. If this is the case, then the comparison between MSPLAN and SGPlan_5 is best interpreted as a comparison between a domain-independent MaxSAT-based preference-based planner and a manually domain-tuned heuristic search preference-based planner, and the generally (but not universally) superior performance of the manually domain-tuned system is to be expected. As such, a more reliable comparison is between MSPLAN and HPLAN-P since this heuristic search preference-based planner outperformed all other IPC-2006 Simple Preferences track competitors (Baier, Bacchus, and McIlraith 2009). Our comparison shows that MSPLAN is competitive with HPLAN-P (what we believe to be the top domain-independent heuristic search preference-based planner). We are currently undertaking a comparison with LAMA (Richter, Helmert, and Westphal 2008), a cost-optimal planner, by exploiting a translation of soft goals into action costs (Keyder and Geffner 2009).

As in SAT-based planning, a bottleneck in partial weighted MaxSAT-based planning appears to be the iteration required to determine the makespan for the solution. In our experiments, we encoded the problem using each possible makespan until the plan with the best quality was found. This is clearly a worst-case scenario. Many SAT-based planners first generate an estimate for the appropriate makespan and use this as a starting point for incremental search. As seen in Table 2, the time required for MSPLAN to generate a plan when given, by an oracle, the makespan that yields the plan with the best quality was typically much smaller than the time required by MSPLAN to generate a plan when this makespan was not known *a priori*. If an incremental approach could be created, whereby the partial weighted MaxSAT encoding of a problem with makespan k is extended, instead of being encoded from scratch, in order to generate the partial weighted MaxSAT encoding with makespan $k+1$, the running time of MSPLAN could likely be improved. A similar incremental approach has recently been investigated in the context of compiling PDDL planning problems into answer set programs (Knecht 2009).

Our planner appeared to be sensitive to the underlying MaxSAT solver. With MiniMaxSAT, MSPLAN was able to solve 12 of the 20 problems in the pathways domain but when SAT4J was used instead, 17 of the 20 problems were solved. Most of the pathways problem instances which could not be solved by MiniMaxSAT had a relatively large number of simple preferences (usually between 35 to 50 simple preferences). Since partial weighted MaxSAT solvers are believed to be very sensitive to the ratio of soft constraints versus hard constraints (Ansótegui, Bonet, and Levy 2009), the relatively larger proportion of soft constraints in these problems might explain the superior performance of SAT4J in the pathways domain. In contrast to the branch-and-bound framework of MiniMaxSAT, SAT4J forgoes the overhead of attempting to prune the search space by computing lower bounds. This decision pays off on underconstrained problems where such prunings are unlikely to trigger.

[1] See for example the function `search_ops_modal` in the file `Parser/inst_utils.c` in the SGPlan code located at `http://ipc.informatik.uni-freiburg.de/Planners` which contains hand-tuning for IPC domains.

5. Concluding Remarks

In this paper, we characterized the PBP problem as a partial weighted MaxSAT problem. We developed a compact encoding of PBP as partial weighted MaxSAT by building on the success of a SAS^+ based SAT encoding. To the best of our knowledge, this is the first time that the SAS^+ formalism has been used in the context of PBP. Our experimental evaluation showed that our MSPLAN system (with our SAS^+ encoding), consistently outperformed an existing MaxSAT-based planner (with a STRIPS encoding) with respect to running time, while generating plans of comparable quality. Remarkably, when run with two different MaxSAT solvers, MSPLAN generated plans with comparable quality to those generated by state-of-the-art heuristic search planners for PBP. Although the heuristic search planners were generally faster, there were problem instances for which at least one of the two MSPLAN systems ran significantly faster than HPLAN-P. The impressive performance of MSPLAN serves to illustrate the effectiveness of our SAS^+ encoding and suggests that both MaxSAT and SAS^+ encodings for PBP are worthy areas of continued exploration.

In this paper, we focused on simple preferences to support comparison with GM. Following the compilation technique described in (Baier and McIlraith 2006), it is possible to translate the full suite of PDDL3 qualitative preferences, including temporally extended preferences, into simple preferences. It is also possible to extend our work to net benefit problems by creating negated unary clauses that represent that an action is not executed at a particular time step and associating the satisfaction of these clauses with a weight corresponding to an action's cost. These are topics of current investigation. A current limitation of MSPLAN is that it is k-optimal rather than optimal. (SGPlan_5 is neither optimal nor k-optimal whereas HPLAN-P has the capacity to be k-optimal with respect to plan length and optimal if run to completion with certain restrictions on metric functions (Baier and McIlraith 2008).) Optimality may be achievable with a MaxSAT-based approach by exploiting a translation of soft goals into action costs (Keyder and Geffner 2009) and a cost-optimal planner based on partial weighted MaxSAT (e.g., (Robinson et al. 2010)).

Acknowledgements

We thank Enrico Giunchiglia and Marco Maratea for providing us with their STRIPS-encodings of the IPC-2006 *Simple Preferences* problems. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Ministry of Innovation, and MITACS.

Instance	MSPLAN: MiniMaxSAT				MSPLAN: SAT4J				HPLAN-P		$SGPlan_5$	
	Time (s)				Time (s)							
	Soln	Oracle	Total	M	Soln	Oracle	Total	M	Time (s)	M	Time (s)	M
trucks1	2.16	5.08	14.01	0	1.78	5.18	21.22	0	5.58	0	0.01	1
trucks2	36.51	41.3	165.91	0	36.04	37.66	173.63	0	92.28	0	0.04	0
trucks3	89.15	111.54	304.1	0	527.21	553.62	861.07	0	660.1	0	0.05	0
trucks4	128.9	426.26	596.18	0	-	-	-	-	563.93	3	0.08	0
trucks5	652.88	675.59	2553.01	0	-	-	-	-	1015.36	0	0.13	0
trucks6	-	-	-	-	-	-	-	-	-	-	0.27	0
trucks7	-	-	-	-	-	-	-	-	-	-	0.63	8
storage1	0.01	1.31	1.66	3	0.17	2.15	2.37	3	0.07	3	0.0	8
storage2	4.9	27.62	34.18	5	4.3	27.15	43.58	5	2.93	5	0.01	16
storage3	708.84	730.15	1192.33	18	1061.36	1083.39	2769.18	7	38.06	6	0.03	41
storage4	23.71	34.51	47.34	38	532.2	637.1	1063.3	24	183.37	9	0.06	49
storage5	70.08	123.29	214.77	107	677.02	705.96	937.9	76	76.57	94	0.13	136
storage6	43.06	71.49	73.19	173	1573.75	1605.23	1978.41	150	459.93	141	0.22	189
storage7	-	-	-	-	127.5	168.81	183.76	277	1280.43	160	0.32	242
pathways1	0.05	1.48	1.6	2	0.18	2.56	3.09	2	3.88	2	0.01	2
pathways2	0.07	0.99	1.29	3	0.46	1.99	3.34	3	186.34	4	0.0	3
pathways3	0.36	3.12	4.95	3	0.59	5.0	7.42	3	115.6	3.7	0.03	3
pathways4	0.36	5.33	5.85	2	0.78	6.25	8.66	2	324.82	2	0.03	2
pathways5	6.47	36.59	45.26	6	2.08	35.39	42.73	6	413.42	9	0.14	6.5
pathways6	434.02	797.73	1468.57	6.4	114.92	103.81	296.1	6.4	0.04	12.9	1.92	7
pathways7	1251.16	1466.24	1710.57	11.5	1378.79	1406.88	2325.46	10.3	0.05	12.5	1.93	10.4
pathways8	201.349	229.51	823.23	18.2	993.28	1020.99	1370.69	18	0.05	20.2	0.92	12.9
pathways9	-	-	-	-	2.33	31.50	32.31	15.7	0.07	15.7	1.4	10.6
pathways10	1182.02	1209.2	1744.95	12.9	775.77	803.84	1692.19	10.1	0.06	16.8	10.94	13.4
pathways11	5.56	32.88	37.85	11.8	1302.28	1332.93	2062.62	9.6	0.0	12.5	2.27	9
pathways12	37.29	47.72	62.02	18.8	2.81	28.77	29.88	18.8	0.04	18.8	14.93	15.4
pathways13	-	-	-	-	-	-	-	-	0.03	22	12.6	16
pathways14	55.28	62.81	80.5	20.7	942.9	1008.96	1092.89	20	0.03	20.7	7.15	15.6
pathways15	-	-	-	-	-	-	-	-	0.06	20.9	0.57	14.5
pathways16	-	-	-	-	660.60	693.78	1060.91	25.7	0.11	25.7	18.4	18.5
pathways17	-	-	-	-	582.82	603.99	615.12	22.3	0.1	22.3	41.94	20.3
pathways18	-	-	-	-	2.26	30.35	30.75	22.8	0.1	22.8	27.68	20
pathways19	-	-	-	-	-	-	-	-	0.05	26.5	41.24	22
pathways20	-	-	-	-	8.31	40.08	41.64	24.7	0.08	24.7	6.53	15
openstacks1	-	-	-	-	-	-	-	-	128.14	6	0.13	13

Table 2: Performance of MSPLAN when run with the BEST algorithm compared to the top PBP heuristic search planners from IPC-2006. Let B be the makespan that results in the plan with the best quality within the time limit. Soln denotes the time required for MiniMaxSAT and SAT4J to find a solution to the partial weighted MaxSAT formula with makespan B. This includes only the solver time to find the solution to the formula. Oracle denotes the total running time of MSPLAN when given, *a priori*, the best makespan B. This time includes the time for the translation to SAS^+, the time for encoding the partial weighted MaxSAT formula with makespan B, and the solver time. Total denotes the total running time when this best makespan B is not known ahead of time and must be determined through an incremental construction of makespans. This time includes the time for the translation to SAS^+, the time for repeatedly encoding a partial weighted MaxSAT formula with increasing makespans, and the total solver times. The total running time of HPLAN-P and $SGPlan_5$ is denoted by Time. M is the value of the plan metric, the total value of the violated preferences in the plan found. (Low is good.) A dash indicates timeout.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT*, 427–440.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 788–795.

Baier, J. A., and McIlraith, S. A. 2008. Planning with preferences. *AI Magazine* 29(4):25–36.

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

Benton, J.; Kambhampati, S.; and Do, M. B. 2006. YochanPS: PDDL3 simple preferences and partial satisfaction planning. In *IPC-2006*, 54–57.

Berre, D. L., and Parrain, A. 2010. The SAT4J library, release 2.2. In *Journal on Satisfiability, Boolean Modeling and Computation*, volume 7, 59–64.

Brafman, R., and Chernyavsky, Y. 2005. Planning with goal preferences and constraints. In *ICAPS*, 182–191.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Giunchiglia, E., and Maratea, M. 2007. Planning as satisfiability with preferences. In *AAAI*, 987–992.

Giunchiglia, E., and Maratea, M. 2010. A pseudo-boolean approach for solving planning problems with IPC simple preferences. In *COPLAS*, 23–31.

Haslum, P. 2007. Quality of solutions to IPC5 benchmark problems: Preliminary results. In *ICAPS*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Heras, F.; Larrosa, J.; and Oliveras, A. 2007. MiniMaxSAT: a new weighted Max-SAT solver. In *SAT*, 41–55.

Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2007. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *IJCAI*, 1924–1929.

Huang, R.; Chen, Y.; and Zhang, W. 2010. A novel transition based encoding scheme for planning as satisfiability. In *AAAI*, 89–94.

Kautz, H. A. 2006. Deconstructing planning as satisfiability. In *AAAI*, 1524–1526.

Keyder, E., and Geffner, H. 2009. Soft goals can be compiled away. *JAIR* 36:547–556.

Knecht, M. 2009. Efficient domain-independent planning using declarative programming. Master's thesis, Hasso Plattner Institute, University of Potsdam.

Maratea, M. 2010. An experimental evaluation of Max-SAT and PB solvers on over-subscription planning problems. In *RCRA*, volume 616.

Marques-Silva, J. 2009. The MSUncore MaxSAT solver.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *AAAI*, 975–982.

Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2010. Partial weighted MaxSAT for optimal planning. In *PRICAI*, 231–243.

Tu, P. H.; Son, T. C.; and Pontelli, E. 2007. CPP: A constraint logic programming based planner with preferences. In *LPNMR*, volume 4483 of *LNCS*, 290–296. Springer.