

Planning with Preferences

Jorge A. Baier
and Sheila A. McIlraith

■ Automated planning is a branch of AI that addresses the problem of generating a set of actions to achieve a specified goal state, given an initial state of the world. It is an active area of research that is central to the development of intelligent agents and autonomous robots. In many real-world applications, a multitude of valid plans exist, and a user distinguishes plans of high quality by how well they adhere to the user's preferences. To generate such high-quality plans automatically, a planning system must provide a means of specifying the user's preferences with respect to the planning task, as well as a means of generating plans that ideally optimize these preferences. In the last few years, there has been significant research in the area of planning with preferences. In this article we review current approaches to preference representation for planning as well as overviewing and contrasting the various approaches to generating preferred plans that have been developed to date.

Preferences play a significant role in human decision making, particularly as they relate to deciding how to act. As such, it comes as no surprise that preferences also play a significant role in AI automated planning, providing a means of specifying those properties of a plan that distinguish it as high quality. Given some task to be achieved, users may have preferences over what goals to achieve and under what circumstances. They may also have preferences over *how* goals are achieved—properties of the world that are to be achieved, maintained, or avoided during plan execution, and adherence to a particular way of doing some or all of the tasks at hand. Interestingly, with the exception of Markov decision processes (MDPs), nontrivial user preferences have only recently been integrated into AI automated planning.

In classical planning, the oldest and best-known planning paradigm, we are given a description of a dynamic domain, an initial state, and a description of a goal. The problem is to find a course of action (that is, a plan) that, when performed in the initial state, leads to a state where the goal is achieved. In a typical planning problem, there may be many plans that achieve the goal, but the problem specification provides no additional criteria to distinguish between good and bad plans. Planners are often constructed to find shortest plans first. This has been extended to minimal cost plans where the cost of a plan is the sum of the cost of its constituent actions.

Preference-based planning (PBP) (see for example, Son and Pontelli [2006]; Bienvenu, Fritz, and McIlraith [2006]) is an extension to the well-known classical planning problem. In PBP we are provided with a criterion to determine when a plan is preferred to another. To this end, we normally consider the relative merit of properties that desirable plans would satisfy. By way of illustration, consider a logistic planning domain in which packages can be transported between cities using trucks. In the classical planning setting, one is typically interested in finding a sequence of actions that results in a state where all packages are located at

their destination. Preferences arise naturally in this domain. For instance, a user may assign higher priority to certain packages, and may prefer that high priority packages be delivered first. Moreover, they may have preferences over the trucks that are used in the plan, perhaps preferring to use trucks with lower gas consumption. Users may also have preferences over truck routes, for example, preferring to use freeways at off-peak times. How to specify user preferences in the context of PBP has been one avenue of research recently.

Once we know the user's preferences, we must relate them to preferences over plans. In particular, we must formally define when one plan is preferred to another. In our example, one might consider that satisfying the preference about priority packages is more important than using freeways. Moreover, we might also like to consider the amount of gas consumed by the trucks as a factor to compare two plans. To define our preference relation among plans, we typically need a language that (1) allows us to define preferences and (2) allows us to aggregate different preferences.

With a specification of the PBP problem in hand, the second challenge is to efficiently generate preferred plans and, ideally, those that are optimal with respect to the criterion provided. The already hard classical planning task of finding one plan is now complicated by requiring a preferred plan. Intuitively, the PBP task is much harder (although in most cases the complexity class does not change). Consequently, as in traditional optimization, we might not be able to find an optimal plan, due to limitations in computational power. In such cases we might still be satisfied with a reasonably good plan, given a reasonable time bound.

In this survey we focus our attention on two different aspects related to the PBP problem: planning preference languages, and algorithms for planning with preferences. We start by providing formal definitions of the classical and PBP problems and then follow with a description of languages and algorithms for PBP. We conclude with a discussion and a description of open lines of research.

Classical Planning

In order to define the problem of planning with preferences, we must first review the definition of the classical planning problem. A *classical planning instance* is a tuple $I = (S, s_0, O, G)$, where S is a set of states, in which each state is a collection of *facts*. Moreover, $s_0 \in S$ is an initial state, $G \subseteq S$ is a set of goal states, and O is a set of operators, where each operator maps a state into another one.

The classical planning problem consists of finding a sequence of operators $a_1 a_2 \dots a_n$ which, when applied to the initial state, will produce a state in G . We refer to the sequence of operators as a *plan*.

It is also possible to describe a plan as a sequence of sets of operators $A_1 A_2 \dots A_n$, where each set A_i contains operators that are not mutually exclusive (that is, which can be performed concurrently without any precondition or effect interactions). The makespan of a plan is the minimum number of (concurrent) steps in which a plan can be carried out. Thus, plan $A_1 A_2 \dots A_n$ has a makespan of n .¹

The planning community has developed a variety of languages to define planning instances. In STRIPS (for example, Bylander 1994), the oldest and best known of these languages, operators are described as triples of the form $(pre(o), add(o), del(o))$, where $pre(o)$ are the preconditions—a set of facts that must hold in a plan state prior to applying o , $add(o)$ is the add-list—a set of facts that will be added to the current state after o is applied, and finally $del(o)$ is the *del-list*—a set of facts that will be deleted from the current state after o is applied. Other languages for specifying planning instances include ADL (Pednault 1989), and PDDL, the Plan Domain Definition Language (McDermott 1998), which is currently the de facto standard for specifying planning instances. Planning in the STRIPS formalism is PSPACE-complete (Fikes and Nilsson 1971).

Preference-Based Planning (PBP)

In preference-based planning (PBP), we are looking for a most-preferred plan for a planning instance I . To define the notion of a most-preferred plan, we use an ordering relation that specifies when a plan for a planning instance I is preferred to another. The planning instance I does not need to satisfy any restrictions; indeed, in the definitions that follow we do not need the instance I to adhere to any of the traditional planning paradigms.

An instance of the PBP problem is a pair (I, \preceq) , where I is a standard planning instance. Furthermore, \preceq is a transitive and reflexive relation in $\mathcal{P} \times \mathcal{P}$, where \mathcal{P} contains precisely all plans for I . The \preceq relation is the formal mechanism for comparing two plans for I . Intuitively $p_1 \preceq p_2$ stands for “ p_1 is at least as preferred as plan p_2 .” Moreover, we use $p_1 \prec p_2$ to abbreviate that $p_1 \preceq p_2$ and $p_2 \not\preceq p_1$. Thus, $p_1 \prec p_2$ holds true if and only if p_1 is strictly preferred to p_2 . Note that, since \prec is a partial preorder, our definition of preference over plans allows incomparability; that is, there could be two plans p_1 and p_2 such that neither $p_1 \preceq p_2$ nor $p_2 \preceq p_1$ hold true. As we will see later, there are some formalisms that do not allow incomparability. In those cases \preceq is total, that is, for any two plans p_1, p_2 either $p_1 \preceq p_2$ or $p_2 \preceq p_1$.

To illustrate how we define \preceq , consider for example that we are interested in a classical planning setting where instances are extended with a

set of soft goals T . Suppose that a plan p_1 is preferred to another p_2 iff p_1 satisfies more soft goals than p_2 . Then if f_1 and f_2 are the respective final states reached by p_1 and p_2 we would say that $p_1 \prec p_2$ iff the number of soft goals appearing in f_1 is greater than the number of soft goals appearing in f_2 . In this simple case, the \preceq relation is total; that is, there are no incomparable plans. On the other hand, we could have instead preferred p_1 to p_2 if p_1 achieved a strict superset of the soft goals achieved by p_2 . In this case, \preceq would result in a partial relation. In the next section, we elaborate on the properties of \preceq as they relate to different PBP formalisms.

We conclude this section by providing a formal definition of the PBP problem. Given an instance $N = (I, \preceq)$, the preference-based planning problem consists of finding any plan in the set

$$\Lambda_N = \{p \in \mathcal{P} \mid \text{there is no } p' \in \mathcal{P} \text{ such that } p' \prec p\}.$$

Intuitively, the set Λ_N contains all the optimal plans for an instance I with respect to \preceq . Observe that now, as opposed to classical planning, we are interested in any plan that is optimal based on \preceq .

Preference Languages and Formalisms

The definition of the preference-based planning problem provided in the previous section requires that we specify a preference relation \preceq between solutions to a standard planning instance I . However, in general we might be looking for a relation \preceq satisfying a number of properties of interest. Moreover, notice that the relation \preceq ranges over all pairs of plans; we obviously do not want to represent it explicitly.

Most of the research on languages and formalisms for preference specification related to PBP focus on the development of preference languages that provide for a compact definition of \preceq . Languages differ with respect to their expressive power. Most languages provide a means of referring to properties of a plan—generally that a property holds (or does *not* hold) in some or all intermediate states traversed during the execution of the plan, or in the final state of the plan. Other languages additionally provide for preferences over action occurrences, preferring some actions over others, perhaps conditioned on state. Languages may also provide a way to specify the temporal relationship between the occurrence of actions or the satisfaction of certain properties during plan execution.

Most preference languages in the literature have either a quantitative or qualitative nature, but some of them admit a combination of both. In quantitative languages, solutions are associated with numeric values, through some numeric func-

tion $g(\cdot)$. To compare whether or not $p_1 \preceq p_2$ we simply verify whether or not $g(p_1) \leq g(p_2)$. In quantitative approaches, the induced \preceq relation is always total. In qualitative languages, on the other hand, $p_1 \preceq p_2$ is defined compactly in terms of properties of p_1 and p_2 but no numbers need to be assigned to plans. The resulting \preceq relation can be either total or partial.

PDDL3 has recently been extended for preference-based planning with hierarchical task networks (HTN) (Erol, Hendler, and Nau 1994). This extension supports preferences on the occurrence, decomposition, and instantiation of HTN tasks (Sohrabi, Baier, and McIlraith 2008).

Quantitative Languages

In this subsection we describe some of the preference languages and formalisms that have been used in preference-based planning. Some of these languages were not proposed specifically for planning but were later used in planning applications.

Decision-Theoretic Planning. Planning problems can be naturally expressed as a Markov decision process (MDP). In MDPs, the state of the world changes as a result of performing actions. Actions have nondeterministic effects. After an action is performed, the agent gets an observation. Moreover, there is a reward function, which associates a reward to every state transition. The reward function can be used to define user preferences, as it quantitatively ranks all possible states. Finally, a policy in the MDP framework is a function that returns an action depending on the history of actions performed and observations received by the agent.

In decision-theoretic planning, one attempts to solve the MDP, obtaining an optimal policy (Boutilier, Dean, and Hanks 1999). The problem of finding optimal policies is thus a form of preference-based planning. Policy π_1 is preferred to policy π_2 if and only if the expected discounted reward obtained by π_1 is greater than the discounted reward obtained with π_2 . A policy π_i implicitly defines a conditional plan p_i such that if policy π_1 is at least as preferred to policy π_2 then $p_1 \preceq p_2$. As user preferences are specified through the reward function, a nontrivial problem is how to generate a utility function that reflects the user's preferences. This problem is known as preference elicitation and is discussed further in this issue (Boutilier and Brazuinas 2008).

Partial Satisfaction Planning (PSP). PSP, also called *oversubscription planning* (for example, Smith [2004] and van den Briel et al. [2004]), is an extension of classical planning. A PSP problem is a tuple $\langle F, A, I, G, U, C \rangle$, where F is a set of facts, $I \subseteq F$ is an initial state, $G \subseteq F$ is a set of goal facts, and A is a set of actions. Moreover, U associates a nonnega-

```

(:constraints
 (and
  ;; truck1 desirably at Toronto between 8 and 9 am
  (preference morning
   (hold-during 8 9 (at truck1 toronto)))
  ;; prefer to deliver priority packages first
  (preference priority
   (forall ?p1 - prio-pck ?p2 - nonprio-pck
    (sometime-before (delivered ?p1)
                      (delivered ?p2))))
  ;; prefer to load packages in economical trucks
  (forall ?p - pck ?t - truck
   (preference in-econ
    (always (implies (loaded ?p ?t) (cheap ?t))))))
  ;; metric function
  (metric minimize (+ (* 10 (is-violated priority))
                      (is-violated in-econ)
                      (* 2 (is-violated morning))))))

```

Figure 1. Preference Priority Is Satisfied in a Plan in Which Priority Packages Are Delivered before Nonpriority Packages.

Preference in-econ defines a preference for loading packages on economical trucks.

tive utility to each fact in F , and C associates a non-negative cost to every action in A . The PSP task usually consists of finding a classical plan p with maximum net benefit, which is defined as the sum of the utilities of the goals reached by p minus the sum of the costs of the actions in p .

Although the literature in PSP typically does not refer to the word *preference*, PSPs can be viewed as planning with preferences. Indeed, if p_1 and p_2 are plans, $p_1 \preceq p_2$ iff p_1 's net benefit is not smaller than p_2 's net benefit.

PSP planning is PSPACE-complete (van den Briel et al. 2004), that is, it has the same complexity as classical planning.

PDDL3. PDDL3 (Gerevini and Long 2005) is an extension of the planning domain definition language, PDDL (McDermott 1998), which provides a rich language for defining user preferences for planning. PDDL3 was designed for the Fifth International Planning Competition (IPC-5),² which is the first international competition that included tracks for preference-based planning.

Preferences in PDDL are formulae that are evaluated over a plan; that is, a preference is satisfied

(respectively violated) by a plan p if it logically evaluates to true (respectively false) in p . Preferences can be *temporal* or *temporally extended*, and they can hold over *preconditions*. Temporal preferences may refer to specific times (for example, I would like the truck to be in Toronto between 8 AM and 9 AM). On the other hand, temporally extended preferences allow the user to express desirable temporal relationships between properties that hold true over some part of the plan execution (for example, I would like priority packages to be delivered *before* nonpriority packages). They are defined in a subset of linear temporal logic (LTL) (Pnueli 1977). Finally, precondition preferences are conditions that are desirable when an action is performed.

In PDDL3, the \preceq relation is determined by a so-called metric function. This is an arithmetic function that receives a plan as an argument and can be dependent on the *is-violated* function, which is a quantitative measure of the level at which preferences are violated.

Figure 1 shows the PDDL3 definitions for some preferences in our example logistics domain. The expression *(is-violated priority)* is equal to 1 if the formula for preference priority is false and is equal to 0 otherwise. On the other hand, *in-econ*, an externally quantified preference, defines a family of preferences, with one individual preference for each package-truck pair. The expression *(is-violated in-econ)* returns the number of individual preferences in the family *in-econ* that violate the LTL formula.

Qualitative Languages

In this subsection we describe various qualitative preferences languages for planning that exist in the literature.

Languages Based on Ranked Knowledge Bases. Ranked knowledge bases (RKBs) provide a means of defining qualitative preferences over problem solutions that correspond to interpretations in classical logic. They were originally proposed for default reasoning (Brewka 1989, Pearl 1990). Related work is described elsewhere in this issue by Brewka, Niemelä, and Truszcynski (2008). An RKB can be represented as a finite set of pairs (f, r) , where f is a logical formula and r is the rank associated to f . Consider, for example, the following RKB:

$$K = \{(at-t1-l1 \wedge at-t2-l1, 3), (at-t1-l1, 2), (at-t2-l1, 1)\}.$$

Brewka (2004) defines several orderings in which K can be used to represent preferences on final states achieved for plans. For example, under the *maxsat* ordering for K , the maximum rank among all satisfied formulae are used to compare two states. Thus, states that satisfy both *at-t1-l1* and *at-t2-l1* (which have rank 3) are strictly preferred to those that satisfy *at-t1-l1* but not *at-t2-l1* (which have

rank 2), and, in turn, the latter are strictly preferred to those that satisfy $at-t2-l1$ but not $at-t1-l1$. Furthermore, Brewka (2004) defines several ways in which preferences induced by two or more RKBs can be aggregated. Feldmann, Brewka, and Wenzel (2006) have proposed two extensions to the PDDL language that allow defining preferences using RKBs. In both of these extensions, preferences are on the final state reached by the plan.

Conditional Preference Networks. Conditional preference networks (CP-nets) (Boutilier et al. 2004) are compact graphical representations for specifying user preferences. Conditional preference relations among variables are represented in a fashion that resembles conditional probability tables in Bayesian networks. For example, one can establish that, *all other things being equal* (that is, *ceteris paribus*), a user prefers eating cookies over cake if he or she is having coffee, but prefers cake over cookies if he or she is having dessert. A CP-net induces a partial ordering over complete assignments of the variables it refers to. Thus, CP-nets are also an effective tool when representing preferences between different planning states, and, in particular, between different goal states. Thus, we say $p_1 \preceq p_2$ if the goal state achieved by p_1 is at least as preferred to the goal state achieved by p_2 . This \preceq relation is partial, since two states can be incomparable given a CP-net.

CP-nets have been extended to represent more expressive notions of preferences, namely trade-offs between certain variables (Brafman, Domshlak, and Shimony 2006). In the resulting networks—called TCP-nets—one can refer to the relative importance of certain variables over others. For example, in our logistics domain, one could say that delivery of priority packages is more important than fuel economy. Hence, if we assume that delivering priority packages first is preferred to delivering them after nonpriority packages, and that a plan with low fuel consumption is preferred to one with high fuel consumption, then a plan that delivers priority packages first with high fuel consumption is still preferred to one that does not deliver the priority packages first with low fuel consumption. TCP-nets also allow two states being incomparable.

Temporal and Temporally Extended Preferences. A variety of formalisms can be found in the literature that enable the user to define qualitative temporal and temporally extended preferences. As we saw earlier, a temporal preference allows the user to talk about the temporal relationship of aspects of the execution of the plan, rather than just to the final state reached by such a plan. Temporal preferences can be expressed in a variety of ways.

Delgrande, Schaub, and Tompits (2007) propose a framework for the representation of temporal

preferences, expressed in a Boolean language extended with arithmetic operators that allows quantification over time steps of the plan. Formulae in this language can refer, in a simple way, to properties of the states traversed or to actions that have occurred in the plan. For example, if $deliver-p$ denotes the action of delivering a package p , the formula:

$$\forall i, j. deliver-p1(i) \wedge deliver-p2(j) \supset i < j$$

can be used to express that package $p1$ is delivered before package $p2$ (if $p1$ and $p2$ are delivered respectively at time steps i and j , then $i < j$). In this framework, the preference relation $p_1 \preceq p_2$ is defined by a logical temporal formula that is evaluated in p_1 and p_2 . The resulting \preceq relation can be partial, allowing incomparability of plans. Although their formalism enables the representation of several different preferences, their work does not focus on providing a rich language for aggregating those preferences. Limited forms of preference aggregation are available, however. Son and Pontelli (2006), on the other hand, develop a language, called \mathcal{PP} , that enables both the representation of temporal preferences and the aggregation of such preferences. Temporal preferences are expressed here in a version of LTL. Later, Bienvenu, Fritz, and McIlraith (2006) developed LPP , an extension of the \mathcal{PP} language with a different semantics, which allows quantification and further aggregation. To get a feel for the expressiveness of these languages consider the following temporal preferences.

$$\varphi_1 \doteq \text{always}(\neg \text{occ}(\text{drive}(t_1, A, B))),$$

$$\varphi_2 \doteq \text{eventually}(\text{at}(t_2, A)),$$

$$\varphi_3 \doteq \text{until}(\neg \text{delivered}(p_2), \text{delivered}(p_1)),$$

$$\varphi_4 \doteq \text{until}(\neg \text{delivered}(p_3), \text{delivered}(p_2)),$$

where φ_1 expresses that the action $drive(t_1, A, B)$ never occurs in the plan, φ_2 that truck t_2 must be at A at some point in the plan, and φ_3 (respectively φ_4) express a preference for delivering p_1 (respectively p_2) before delivering p_2 (respectively p_3). A so-called *atomic preference formula* (APF) can be used to express the relative importance of different preferences. APFs explicitly refer to the level of satisfaction that is attained by the user. Let us assume that $\mathcal{V} = \{\text{excellent}, \text{good}, \text{ok}, \text{bad}\}$ is an ordered set containing the user's levels of satisfaction (where *excellent* is the best and *bad* is the worst). Then if:

$$\psi_1 \doteq \varphi_1 \wedge \varphi_2[\text{excellent}] \gg \varphi_2[\text{good}] \gg \varphi_1[\text{ok}],$$

$$\psi_2 \doteq \varphi_3[\text{excellent}] \gg \varphi_4[\text{good}],$$

ψ_1 expresses that it is “excellent” to satisfy both φ_1 and φ_2 whereas it is still “good” to satisfy φ_2 and “ok” to satisfy φ_1 . Moreover, LPP preferences can be aggregated using conjunctive, disjunctive, and conditional connectives. For example $\psi_1 \& \psi_2$ means that the user would like to satisfy both ψ_1 and ψ_2 as much as possible; thus, it is “excellent” if

φ_1 , φ_2 , and φ_3 are satisfied by the plan but only “good” if φ_1 , φ_2 , φ_4 , but not φ_3 are satisfied by the plan. Preference formulae in *LPP* evaluated on a plan always evaluate to a value in the set of user satisfaction levels \mathcal{V} , and thus the relation induced by an *LPP* formula is total. This is not the case in the \mathcal{PP} language, which does allow incomparability. *LPP* was recently extended to *LPH*, a language that enables the expression of qualitative preferences for HTN planning. It maintains the expressive power of *LPP*, augmenting it with preferences over the occurrence, decomposition, and instantiation of HTN tasks (Sohrabi and McIlraith, 2008).

Qualitative and Quantitative Combined

Some of the languages we have mentioned above are not exclusively qualitative. Many of them can integrate quantitative criteria into the preference language. In the formalism of Delgrande, Schaub, and Tompits (2007) we can use a number of aggregation functions, which enables us to define the preference relation \preceq in quantitative terms. For example, the aggregator $\text{count}[\varphi]$ returns the number of times the property φ is true during the execution of the plan. One could then establish that a plan p_1 is preferred to another plan p_2 if the number of times some fluent f is true in p_1 does not exceed the number of times f is true in p_2 . In a similar way, *LPP* allows resorting to preference aggregators that could sum up the levels of satisfaction achieved by different input preference formulae to compare two plans.

Fritz and McIlraith (2006) provide a clean integration of qualitative preferences, in the aforementioned *LPP* language, and quantitative preferences, represented through a utility function over plan states. Their work builds upon the DT-Golog agent programming language (Boutilier et al. 2000), whose semantics is designed in such a way that, given a program δ , the interpreter searches for the best possible execution of δ for the given a utility model. Given a preference formula ψ in *LPP*, a DT-Golog program can be synchronized with another program generated from ψ in such a way that the interpreter will search for the quantitatively best plan among the set of most preferred plans given by ψ . If ψ cannot be satisfied at its maximum level, then the interpreter will iterate through the remaining levels of satisfaction, searching for the qualitatively best solution in each level.

Algorithms for PBP

In this section, we describe different approaches to addressing the PBP problem, and in particular how to generate most-preferred preference-based plans. Although one way of achieving PBP is to do classical planning on each possible instance that results

from some combination of the user’s preferences, and then choose the best, the focus of current research is on *efficient* PBP. Like their classical planning counterparts, many of the top-performing preference-based planners perform some form of domain-independent heuristic search.

Although PBP planners focus on efficiency not all of them guarantee optimality (that is, finding a *most-preferred* plan). Some algorithms, for example, return a succession of potentially suboptimal plans of increasing quality, until an optimal plan is eventually found or resources are exhausted. Because of this, there are other properties, besides optimality, that we are going to be interested in analyzing.

We start by the definition of optimality, which essentially defines that an algorithm solves the PBP problem. Note that we require that the algorithm eventually returns an optimal. We do not preclude it from returning other suboptimal plans.

Definition 1 (Optimal). An algorithm A is optimal for the PBP instance (I, \preceq) if it eventually outputs an optimal plan with respect to \preceq .

Some of the algorithms we describe below will not be able to achieve optimality, basically because they are not designed to search for an arbitrarily long plan. Some of these algorithms are able to achieve a restricted notion of optimality, which we call k -optimality.

Definition 2 (k -optimal). Given a positive integer k , an algorithm A is k -optimal iff given any PBP instance (I, \preceq) it outputs an optimal plan with respect to \preceq from all those plans whose makespan is at most k , if such a plan exists.

Finally, an incremental planner is one that returns a sequence of plans with increasing quality.

Definition 3 (Incremental). An algorithm A is incremental iff there exists a nonempty family \mathcal{F} of PBP instances such that for each instance $(I, \preceq) \in \mathcal{F}$, it outputs a nonempty and nonsingleton sequence of plans for $I, p_1 p_2 \dots$, such that $p_i \prec p_j$ for all i, j such that $1 \leq i < j$.

There is a subtle difference between incremental and *anytime* planners. An anytime planner must guarantee a solution at any time during the computation. This is not possible when the planning problem has hard goals, since a solution will only be available when one plan has been found. If the planning problem lacks hard goals, any incremental planner is also anytime.

We now describe some of the most important approaches to planning with preferences in the literature. We divide the planners into two groups: those that use general-purpose solvers as the planning engine, and those that adapt a search algorithm for planning without preferences.

Search-Based PBP

Search-based planners utilize standard search algorithms for the purpose of planning with preferences. In this category we include algorithms that use off-the-shelf search algorithms using specialized heuristics for preferences.

Many of the PBP algorithms in the literature can be thought of as applying some kind of best-first search with branch-and-bound pruning. Figure 2 shows PREF-SEARCH, a generic example of such an algorithm, which searches for a solution that minimizes the *quality* of the plan (note that it can be trivially modified to maximize the quality). The algorithm does not actually require that quality be a numeric value but rather that there is a well-defined (partial) ordering between the qualities assigned to different plans. This feature allows it to work for both qualitative and quantitative approaches.

PREF-SEARCH stores in the variable *bestQuality* the quality of the best plan found so far. Moreover, for each node *current* considered, PREF-SEARCH computes an estimated lowerbound on the quality of all plans that visit the *current* state (line 5). The algorithm expands a node (line 7) only if this lower bound is better than *bestQuality*, which allows pruning states that are not going to yield plans that are better than those already found by the algorithm. To sort the search frontier, it uses the evaluation function EVALFN().

PREF-SEARCH is an incremental algorithm, since it can output a sequence of plans depending on the termination condition, given by the function TERMINATE?(). Note that if TERMINATE?() is true only when *OpenList* is empty, we obtain an exhaustive incremental planner. There are two conditions that are key to the optimality of PREF-SEARCH: (1) EVALFN() underestimates the actual value of the quality of the best plan that visits *current* (that is it is an admissible function), and (2) the ESTIMATE-LOWER-BOUND function effectively returns a lower bound on the quality of any plan that visits *current*. Condition 1, as in standard best-first search, guarantees optimality because search is guided towards an optimal. On the other hand, condition 2 guarantees that no state that leads to an optimal will be pruned from the search space. If condition 1 does not hold but condition 2 holds, then PREF-SEARCH can still be optimal if it exhausts the search space. Now we turn our attention to some of the search-based PBP planners that have been proposed in the literature. A summary of the features of the planners described is shown in table 1.

Final State Preferences. There are a number of PBP planners that are limited to considering preferences over properties of the final state of the plan. One of the first approaches to planning for PSP problems is the one proposed by Smith (2004) for NASA applications. In this approach, the cost of

```

1: function PREF-SEARCH(PBPinstance I)
2:   OpenList ← INITFRONTIER(init) ▷ initialize search frontier
3:   bestQuality ← Qualityupperbound
4:   while not TERMINATE?() do
5:     current ← Extractbestelementfrom OpenList
6:     lbound ← ESTIMATE-LOWER-BOUND(current)
7:     if lbound < bestQuality then ▷ pruning by bounding
8:       if PLAN?(current) and current.quality < bestQuality
9:         then
10:           Outputplan current
11:           bestQuality ← current.quality
12:         endif
13:         succ ← successorsof current
14:         ComputeEVALFN() onelementsin succ
15:         OpenList ← merge succ into OpenList
16:       endif
17:   end while
end function

```

Figure 2. Generic Forward-Chaining, Best-First Search Algorithm with Branch-and-Bound Pruning for Finding Minimal-Quality Plans.³

subgoals is estimated from a relaxed plan (Hoffmann and Nebel 2001) over an abstracted version of the original planning problem, which ignores the cost of some actions. Using the information of the costs for each subgoal, an orienteering problem is constructed and then solved using standard search techniques. The solution to the orienteering problem is then used to feed a partial order planner with a sequence of goals to achieve.

Sapa^{PS} (van den Briel et al. 2004) is a forward-chaining best-first incremental planner for PSP problems. Its evaluation function estimates the maximum net benefit that can be obtained from the current state. To compute such an estimate for a state *s*, it builds a relaxed plan for all (soft) goals from *s*, and estimates the subgoals that achieve maximum net benefit in the relaxed plan using a greedy algorithm. The evaluation function is also used to prune states that look unpromising; *Sapa^{PS}* will prune a node if its evaluation function is worse than the net benefit of the best plan found so far. The evaluation function in *Sapa^{PS}* is inadmissible, and therefore the search might miss an optimal plan, because a state leading to an optimal might be pruned from the search space. *Yochan^{PS}* (Benton, Kambhampati, and Do 2006), one of the IPC-5 competitors, casts PDDL3 planning problems with final-state and precondition preferences into PSPs by interpreting precondition preferences as conditional action costs and by associating rewards to final-state preferences, which are obtained from the PDDL3 metric function to each final-state pref-

Planner	Input Language	Optimal	k -Optimal	Incremental
<i>Sapa</i> ^{PS}	PSP	No	No	Yes
<i>Yochan</i> ^{PS}	PDDL3	No	No	Yes
BBOP-LP	PSP	Yes	No	Yes
BG-KCOMP	A variation of PSP	When actions have no conditional costs	No	No
PPLAN	<i>LPP</i>	No	Yes	No
HPLAN-P	PDDL3	Dependent on metric function	Yes	Yes
HPLAN-QP	<i>LPP</i>	Yes	Yes	Yes
MIPS-XXL	PDDL3	Yes	No	Yes
MIPS-BDD	PDDL3	Yes	No	No
SGPlan5	PDDL3	No	No	No
ASPLAN	\mathcal{PP}	No	Yes (*)	No
CPP	\mathcal{PP}	No	Yes (*)	No
PRIOMFF	Extended RKBs	Yes	No	Yes
PREFPLAN	TCP-nets	No	Yes	No
SATPLAN(P)	PDDL3	No	Yes	No

Table 1. Features of PBP Planners.

Note that most planners that are optimal can be easily made k -optimal if search is limited to plans of makespan of at most k . (*): Finds optimal plans of length *equal* to k .

erence. It invokes *Sapa*^{PS} for solving the resulting PSP. *Yochan*^{PS} obtained a distinguished performance award in the simple (final-state or precondition) preferences track of IPC-5.

Recently, Benton, van den Briel, and Kambhampati (2007) proposed BBOP-LP, an incremental branch-and-bound algorithm, which uses a linear programming (LP) solution to an integer programming encoding of a relaxation of the original problem to obtain search heuristics. Specifically, the LP solution is used to bias the computation of relaxed plans which in turn is used to estimate the maximum net benefit that can be achieved by states evaluated by the search. The evaluation function used by BBOP-LP is admissible, and, as in *Sapa*^{PS}, it is used to prune unpromising states.

A quite different approach to PSP planning is taken by the *AltAlt*^{PS} planner, also by van den Briel et al. (2004). This is a backward-chaining planner, which also uses a relaxed plan to greedily determine a subset of the goals that are estimated to

produce a maximum net benefit. Then, a standard planning algorithm is used to plan for such a subset of goals. Because the estimation of the subset of goals to achieve is heuristic, the algorithm is not guaranteed to find an optimal.

Bonet and Geffner (2006) have proposed a framework for planning in the presence of action with costs and where costs and rewards can be associated with fluents that become true at some point during the execution of the plan. Their cost model can represent PSPs as well as the simple preferences subset of PDDL3. They propose admissible heuristics, obtained from compiling a relaxed instance of the problem into a d-DNNF representation. They also propose an optimal algorithm for planning under this model based on best-first search. The approach does not scale very well for large planning instances, in part because of its need to employ an admissible heuristic. We refer to this planner as BG-KCOMP.

Finally, Feldmann, Brewka, and Wenzel (2006)

propose an incremental approach to planning with preferences on the final-state. (Since this planner does not have a formal name, we refer to it here as `PRIOFF`.) The planner repeatedly invokes the `METRIC-FF` planner (Hoffmann 2003) to obtain plans of increasing quality. It does so by representing the quality of plan in a new fluent *val*, which is added to the input domain. Then, if in the *i*-th iteration it obtains a plan of quality V_i , in the (*i* + 1)-th iteration it adds $val > V_i$ as a new hard goal. Feldmann, Brewka, and Wenzel (2006) define two extensions of PDDL, which they use to represent planning problems. One of them, `PDDLq`, enables the specification of final-state preferences in the style of RKBs.

Temporally Extended Preferences (TEPs). There are a number of planners in the literature that are able to plan with TEPs. Temporal preferences are compelling, but much of the work in the area was motivated by the interest in the new features in PDDL for the 2006 planning competition.

`PPLAN` (Bienvenu, Fritz, and McIlraith 2006) is a forward-chaining best-first search planner for the `LPP` language. Its planning algorithm, also an instance of `PREF-SEARCH`, searches the space of plans whose length is at most a parameter *k*. It searches using an admissible evaluation function that uses an optimistic estimator of the plan quality and then a pessimistic estimator for breaking ties. In a nutshell, the optimistic estimator regards as satisfiable all preferences that could still be made true. For example, it considers that **eventually**(φ) is always satisfiable, while **always**(φ) is regarded violated if and only if the plan has made φ false at some state visited by the current plan. `PPLAN` is *k*-optimal. Descendants of `PPLAN` include `GOLOG-PREF`, a version of `PPLAN` that supports procedural domain control knowledge (Sohrabi, Prokoshyna, and McIlraith 2006), and `HTNPLAN` a reimplementation of the `PPLAN` search algorithm within an HTN planner (Sohrabi and McIlraith 2008). Both extensions were motivated by the task of web service composition.

`HPLAN-P` (Baier, Bacchus, and McIlraith 2007) is also a forward-chaining branch-and-bound planner for TEPs. Its input language is `PDDL3`, though it supports a broader LTL subset than that supported in `PDDL3` (in particular, it allows nesting of temporal operators). There are a number of evaluation functions implemented in `HPLAN-P`. Given the planning state *current*, most of them attempt to estimate the quality (`PDDL metric`) of *current* by expanding a relaxed planning graph. As opposed to other planners, like *Sapa^{PS}*, the heuristics do not make an analysis of the goals that minimize the metric but rather weight soft goals depending on the estimated difficulty to achieve them. It also provides two functions to compute lower bounds. These functions will not prune a state from the

search space if it can lead to an optimal, under certain conditions of the metric function (which are satisfied in the IPC-5 benchmarks). Key to this approach is the compilation of TEPs into standard, final-state preferences using a transformation into nondeterministic, parametrized finite-state nondeterministic automata that avoids grounding the preferences and therefore avoids size blowups in the output domains. This compilation is fairly independent of the planning algorithm and therefore could be exploited by other planners for final-state preferences. `HPLAN-P` obtained a distinguished performance award in IPC-5. It has been extended to plan with a subset of the `LPP` language (Baier and McIlraith 2007). The heuristic search techniques developed for `HPLAN-P` were the inspiration for the heuristic search in `HTNPLAN-P`, an HTN planner that performs heuristic search to find preferred HTN plans from a version of `PDDL3` extended with HTN task preferences (Sohrabi, Baier, and McIlraith 2008). Of related note, Lin et al. (2008) recently developed a preference-based HTN planner that processes vanilla `PDDL3` preferences, while the `ASPEN` system (Rabideau et al. 2000) performs a simple form of PBP focused mainly on preferences over resources, but with the facility to exploit an HTN-like task decomposition.

`MIPS-BDD` (Edelkamp 2006) is an optimal planner for the `PDDL3` language. It applies a bidirectional breadth-first search approach that encodes sets of states into binary decision diagrams (BDDs). To handle temporally extended preferences it compiles them into final-state preferences through grounded Buchi automata. In the resulting problem specification, the quality of a plan is encoded within a new numerical fluent of the problem. `MIPS-XXL` (Edelkamp, Jabbar, and Naizih 2006), on the other hand, uses the same compilation to implement a heuristic planner based on enforced hill climbing (Hoffmann and Nebel 2001). `MIPS-XXL` repeatedly invokes a modified version of `METRIC-FF` (following an approach similar to that of `PRIOFF`) to find plans with increasing quality. An interesting feature is that `MIPS-XXL` uses disk space to store search nodes if main memory is not sufficient.

Finally, `SGPlan5` (Hsu et al. 2007) is the search-based PBP planner that won the IPC-5 planning competition in all preference tracks. Unlike the planners described above, `SGPlan5` searches for a plan by using constraint partitioning, decomposing the original planning problem into several subproblems. This technique stems from treating the PBP problem as a standard optimization problem, where the objective function is to minimize the makespan of the plan. Using the so-called extended saddle-point condition, an optimization problem can be partitioned into subproblems, and a local minimum or maximum to the global prob-

lem can be found by solving the subproblems. $SGPlan_5$ does the partitioning in such a way that the resulting problems are planning problems and therefore can be solved with a state-of-the-art planner. In particular, it uses a modified version of METRIC-FF to solve the subproblems. $SGPlan_5$ optimizes the plan quality by formulating different optimization problems (that is different sets of soft goals and temporal constraints) to be solved by constraint partitioning. The enumeration strategy for generating the different optimization problems is determined using a heuristic algorithm.⁴ $SGPlan_5$ is not optimal; this is both because of the solving strategy and due to the fact that the search heuristics used are not admissible. Finally, it is not incremental.

PBP through General-Purpose Solvers

There are a number of preference-based planning approaches that use general-purpose solvers such as SAT, CSP or ASP solvers. We describe some of them here.

Son and Pontelli (2006) describe ASPlan, a PBP planner for their \mathcal{PP} language. In the approach taken by ASPlan one converts a general preference formula and the planning task into a logic program. This program is such that, when solved with the answer-set-programming solver Smodels (Niemelä 1999), it will generate a most-preferred plan of length k . The key idea is to encode a weight function that will return a numeric value depending on how much the preferences are achieved by a certain plan. The weight function is then maximized using a standard Smodels maximization construct. Tu, Son, and Pontelli (2007) later proposed the CPP planner, which uses a similar translation for the \mathcal{PP} language but runs under GNU-Prolog and performs orders of magnitude faster in some domains. Both of these planners are k -optimal.

Brafman and Chernyavsky (2005) propose PREFPLAN, which uses a solver for constraint satisfaction problems (CSPs) as the planning engine. PREFPLAN plans for user preferences on the final state encoded using a TCP-net (Brafman, Domshlak, and Shimony 2006). Given an integer k it encodes a Graphplan planning graph of depth k into a CSP. Then, it uses the TCP-net to generate variable and value orderings for the goal variables mentioned in the TCP-net. These orderings are then plugged into the CSP solver, forcing it to check most-preferred solutions first. PREFPLAN is k -optimal.

SATPLAN(P) (Giunchiglia and Maratea 2007), on the other hand, is an extension of the award-winning SATPLAN planner (Kautz and Selman 1999) that is able to plan for final-state preferences by calling an external SAT solver. The approach is similar to PREFPLAN in the sense that a variable ordering is imposed on propositional variables corresponding to final-state preferences in such a way that most-

preferred plans will be explored first by the SAT solver. Preferences in SATPLAN(P) can be defined either in a qualitative or a quantitative language. In the qualitative language, the \preceq relation is induced from a partial order between final-state properties on the final state. In the quantitative language, on the other hand, each preference on the final state has an associated weight; two plans are compared based on the sum of the weights of the preferences satisfied in the final state.

Open Research Directions

There are a number of open research directions for PBP. We discuss two of them: improvements to current planning algorithms, and interaction with the user.

Although there has been progress towards developing advanced PBP algorithms, in our opinion, these have not yet reached a level of maturity comparable to current state-of-the-art planners in at least two respects. First, in terms of efficiency, although current PBP planners are able to utilize heuristics, most of these heuristics are based on relaxed planning graph techniques. Such heuristics are known to be effective in general but can be quite uninformative in very simple scenarios. Something similar applies to bounding functions used for pruning that are key to the planner's performance. Most of these functions utilize a relaxed planning graph too (BBOP-LP being a good counterexample).

Second, PBP development has focused mainly on deterministic planning (with work by Shapara, Pistore, and Traverso [2006] as one of the few exceptions). Consequently, preference languages are specially designed for a deterministic view of the world. We expect to see development of languages and planners for more general settings, such as planning in the context of incomplete knowledge.

On a different topic, current approaches to PBP assume that the problem of preference elicitation is solved. However, elicitation of users' preferences may not be easy to achieve. Arguably, as in other decision-making processes, users are usually unable to provide a full account of their preferences until they are shown different alternatives. However, in contrast to other decision-making problems, in PBP showing different alternatives might be hard to achieve, since coming up with just one plan could be computationally very hard. Preference elicitation, as it relates specifically to PBP, and mixed-initiative PBP, where the user and planner interact to develop a most-preferred plan, are topics for future research.

Notes

1. Makespan is typically defined as the minimum time in

which we can perform the plan. We use this definition here as we restrict our presentation to planning problems that, for the most part, do not mention time explicitly.

2. zeus.ing.unibs.it/ipc-5/.

3. For brevity, we have omitted the closed list of nodes for storing nodes that have already been expanded.

4. Based on a personal communication with C.-W. Hsu.

References

- Baier, J. A., and McIlraith, S. A. 2007. On Domain-Independent Heuristics for Planning with Qualitative Preferences. Paper presented at the 7th IJCAI Workshop on Nonmonotonic Reasoning, Action and Change (NRAC-07). Hyderabad, India, January 7–8.
- Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2007. A Heuristic Search Approach to Planning with Temporally Extended Preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1808–1815. Menlo Park, CA: AAAI Press.
- Benton, J.; Kambhampati, S.; and Do, M. B. 2006. *YochanPS*: PDDL3 Simple Preferences and Partial Satisfaction Planning. Paper presented at the Fifth International Planning Competition (IPC-2006), part of the International Conference on Automated Planning and Scheduling. The English Lake District, Cumbria, UK, June 6–10.
- Benton, J.; van den Briel, M.; and Kambhampati, S. 2007. A Hybrid Linear Programming and Relaxed Plan Heuristic for Partial Satisfaction Problems. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, 34–41. Menlo Park, CA: AAAI Press.
- Bienvenu, M.; Fritz, C.; and McIlraith, S. A. 2006. Planning with Qualitative Temporal Preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR-06)*, 134–144. Menlo Park, CA: AAAI Press.
- Bonet, B., and Geffner, H. 2006. Heuristics for Planning with Penalties and Rewards Using Compiled Knowledge. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR-06)*, 452–462. Menlo Park, CA: AAAI Press.
- Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. *Journal of Artificial Intelligence Research* 21: 135–191.
- Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11: 1–94.
- Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 355–362. Menlo Park, CA: AAAI Press.
- Brafman, R., and Chernyavsky, Y. 2005. Planning with Goal Preferences and Constraints. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, 182–191. Menlo Park, CA: AAAI Press.
- Brafman, R. I.; Domshlak, C.; and Shimony, S. E. 2006. On Graphical Modeling of Preference and Importance. *Journal of Artificial Intelligence Research* 25: 389–424.
- Braziunas, D., and Boutilier, C. 2008. Elicitation of Factored Utilities. *AI Magazine* 29(4): 79–92.
- Brewka, G. 1989. Preferred Subtheories: An Extended Logical Framework for Default Reasoning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1043–1048. San Francisco: Morgan Kaufmann Publishers.
- Brewka, G. 2004. A Rank Based Description Language for Qualitative Preferences. Paper presented at the 16th European Conference on Artificial Intelligence (ECAI-04), 303–307. Bruxelles, Belgium: European Coordinating Committee for Artificial Intelligence (ECCAI).
- Brewka, G.; Niemelä, I.; and Truszczyński, M. 2008. Preferences and Nonmonotonic Reasoning. *AI Magazine* 29(4): 60–78.
- Bylander, T. 1994. The Computational Complexity of Propositional STRIPS planning. *Artificial Intelligence* 69(1–2): 165–204.
- Delgrande, J. P.; Schaub, T.; and Tompits, H. 2007. A General Framework for Expressing Preferences in Causal Reasoning and Planning. *Journal of Logic and Computation* 17(5): 871–907.
- Edelkamp, S. 2006. Optimal Symbolic PDDL3 Planning with MIPS-BDD. Paper presented at the Fifth International Planning Competition (IPC-2006), part of the International Conference on Automated Planning and Scheduling. The English Lake District, Cumbria, UK, June 6–10.
- Edelkamp, S.; Jabbar, S.; and Naizih, M. 2006. Large-Scale Optimal PDDL3 Planning with MIPS-XXL. Paper presented at the Fifth International Planning Competition (IPC-2006), part of the International Conference on Automated Planning and Scheduling. The English Lake District, Cumbria, UK, June 6–10.
- Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. HTN Planning: Complexity and Expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, 1123–1128. Menlo Park, CA: AAAI Press.
- Feldmann, R.; Brewka, G.; and Wenzel, S. 2006. Planning with Prioritized Goals. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR-06)*, 503–514. Menlo Park, CA: AAAI Press.
- Fikes, R. and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3/4): 189–208.
- Fritz, C., and McIlraith, S. A. 2006. Decision-Theoretic Golog with Qualitative Preferences. In *Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR-06)*, 153–163. Menlo Park, CA: AAAI Press.
- Gerevini, A., and Long, D. 2005. Plan Constraints and Preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy.
- Giunchiglia, E., and Maratea, M. 2007. Planning as Satisfiability with Preferences. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI-07)*, 987–992. Menlo Park, CA: AAAI Press.
- Hoffmann, J. 2003. The Metric-FF Planning System: Translating “Ignoring Delete Lists” to Numeric State Vari-

The AAAI Conference Turns 25 in San Francisco!

AAAI is pleased to announce that the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11) will be held at the Hyatt Regency in San Francisco, California, August 7–11, 2011. You won't want to miss this milestone event, held for the very first time in the City by the bay. Details about the AAAI-11 program will be posted as they become available at www.aaai.org/aaai11. We hope to see you in San Francisco!

- ables. *Journal of Artificial Intelligence Research* 20: 291–341.
- Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation through Heuristic Search. *Journal of Artificial Intelligence Research* 14: 253–302.
- Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2007. Constraint Partitioning for Solving Planning Problems with Trajectory Constraints and Goal Preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1924–1929. Menlo Park, CA: AAAI Press.
- Kautz, H. A., and Selman, B. 1999. Unifying SAT-Based and Graph-Based Planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 318–325. San Francisco: Morgan Kaufmann Publishers.
- Lin, N.; Kuter, U.; and Sirin, E. 2008. Web Service Composition with User Preferences. In *Proceedings of the 5th European Semantic Web Conference (ESWC2008)*. Lecture Notes in Computer Science, Volume 5021. Berlin: Springer.
- McDermott, D. V. 1998. PDDL—The Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.
- Niemelä, I. 1999. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3–4): 241–273.
- Pearl, J. 1990. System Z: A Natural Ordering of Defaults with Tractable Applications to Nonmonotonic Reasoning. In *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge (TARK-90)*, 121–135. San Francisco: Morgan Kaufmann Publishers.
- Pednault, E. P. D. 1989. ADL: Exploring the Middle Ground between Strips and the Situation Calculus. In *Proceedings of the 1st International Conference of Knowledge Representation and Reasoning (KR-89)*, 324–332. San Francisco: Morgan Kaufmann Publishers.
- Pnueli, A. 1977. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS-77)*, 46–57. Piscataway, NJ: Institute of Electrical and Electronics Engineers.
- Rabideau, G.; Engelhardt, B.; and Chien, S. 2000. Using Generic Preferences to Incrementally Improve Plan Quality. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 236–245. Menlo Park, CA: AAAI Press.
- Shaparau, D.; Pistore, M.; and Traverso, P. 2006. Contingent Planning with Goal Preferences. In *Proceedings of the*

21st National Conference on Artificial Intelligence (AAAI-06). Menlo Park, CA: AAAI Press.

Smith, D. E. 2004. Choosing Objectives in Over-Subscription Planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, 393–401. Menlo Park, CA: AAAI Press.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2008. HTN Planning with Quantitative Preferences via Heuristic Search. Paper presented at the ICAPS 2008 Workshop on Oversubscribed Planning and Scheduling, Sydney, Australia, September, 2008.

Sohrabi, S. and McIlraith, S., 2008. On Planning with Preferences in HTN. In *Multidisciplinary Workshop on Advances in Preference Handling*, Ed. J. Chomicki, V. Conitzer, U. Junker, and P. Perny, 103–109. AAAI Technical Report WS-08-09. Menlo Park, CA: AAAI Press.

Sohrabi, S.; Prokoshyna, N.; and McIlraith, S., 2006. Web Service Composition via Generic Procedures and Customizing User Preferences. In *Proceedings of the 5th International Semantic Web Conference (ESWC2008)*. Lecture Notes in Computer Science, Volume 4273, 597–611. Berlin: Springer.

Son, T. C., and Pontelli, E. 2006. Planning with Preferences Using Logic Programming. *Theory and Practice of Logic Programming* 6(5): 559–607.

Tu, P. H.; Son, T. C.; and Pontelli, E. 2007. CPP: A Constraint Logic Programming Based Planner with Preferences. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-07)*, volume 4483 of LNCS, 290–296. Berlin: Springer.

van den Briel, M.; Nigenda, R. S.; Do, M. B.; and Kambhampati, S. 2004. Effective Approaches for Partial Satisfaction (Oversubscription) Planning. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)*, 562–569. Menlo Park, CA: AAAI Press.

Jorge A. Baier received his bachelor and MSc degree for the Pontificia Universidad Católica in 1999 and 2000, respectively. In 2002, he joined the Computer Science Department at the Pontificia Universidad Católica de Chile, where he is currently holds an instructor position. He is currently a Ph.D. student at the University of Toronto. Baier and colleagues received a distinguished performance award at the International Planning Competition in 2006, qualitative preference track, for their planner HPLAN-P.

Sheila McIlraith is an associate professor in the Department of Computer Science at the University of Toronto. Prior to joining the University of Toronto, McIlraith spent six years as a research scientist at Stanford University, and one year at Xerox PARC. McIlraith's research is in the area of knowledge representation and automated reasoning. She has 10 years of industrial R&D experience developing artificial intelligence applications. McIlraith has authored more than 20 scholarly publications, is an associate editor of the journal *Artificial Intelligence* and past program cochair of the International Semantic Web conference. McIlraith's early work on semantic web services has had notable impact. Her research has also made practical contributions to the development of next-generation NASA space systems and to emerging web standards.