Planning with Goal Preferences and Constraints[*]

Ronen I. Brafman[†]
NASA Ames Research Center[‡] and
Computer Science Dept.
Stanford University
brafman@email.arc.nasa.gov

Yuri Chernyavsky
Computer Science Department
Ben-Gurion University
Beer-Sheva 84105, Israel
yurac@cs.bgu.ac.il

Abstract

In classical planning, the planner is given a concrete goal; it returns a plan for it *or* a failure message. In the latter case, the user can either quit or modify the goal. For many applications, it is more convenient to let the user provide a more elaborate specification consisting of constraints and preferences over possible goal states. Then, let the system discover a plan for the most desirable among the feasible goal states. To materialize such an approach we require a formalism for specifying preferences and constraints over goals and an algorithm for solving the resulting constrained optimization problem. In this work we motivate the need for planning with preferences and constraints, suggest a rich, yet intuitive formalism for representing goal preferences in the context of a deterministic action model, discuss some of its properties, propose an efficient algorithm for planning with preferences and constraints based on this formalism, and provide extensive experimental analysis in an interesting new domain of configuration planning.

Introduction

Much recent work in the area of AI planning is concerned with extensions of the classical planning framework that contain explicit notions of time, non-deterministic actions, more involved goal conditions, and explicit resource constraints. As the complexity of the specification of a planning problem increases, it is much harder to determine before hand whether a particular goal is achievable. As an example, consider the well-known logistics domain. As long as there is some transportation path between the initial and final location of each package, we know that a plan can be found. However, if constraints on time, maximal number of plane rides, etc., are added, plan existence becomes nontrivial. This is also true when we introduce uncertainty about the effect of actions, and more complex goal conditions. Indeed, in many planning domains, one of the main problems

is to identify a feasible subset of a large set of potential goals (e.g., packages to ship) for which to plan for. This problem has been discussed recently under the terms of over-subscription and/or partial satisfaction planning (Briel *et al.* 2004; Smith 2004).

When the feasibility of achieving a user's goal becomes less likely or when the set of goals is a-priori too large, we must choose between two modes of operation. In one mode, we simply announce our failure, and ask the user to specify a different goal. In the other, more appealing approach, we elicit the user's preferences over the set of acceptable goal states, and attempt to generate a plan for the optimal feasible goal state. Here are a few motivating examples:

Planetary Exploration: There are a number of sites from which we would like to collect samples. Obtaining a sample from some sites is more important than from other sites. Within each site, there are different tests that can be done, with some being more important than the others. We are constrained by battery life and carrying capacity to a certain number of samples and a certain number of tests. This domain motivated the work of (Smith 2004) on *over-subscription planning*. Smith notes the fact that due to its resource constraints, the Mars Rover is limited in the number of goals it can achieve, and examines various techniques for selecting the best site visit schedule for the Rover.

Logistics Domain: Consider a variant of the logistics domain, in which we are constrained by time/resources in the amount of travel that we can do. We have a number of packages to deliver, but it is unlikely that we can deliver all of them. Some packages have priority over other packages. Some packages require the content of other packages to be useful. For example, some packages contain legal material that cannot be understood without the content of some other package. Or, the content of some packages (e.g. a battery) is required to operate the content of another package.

Travel Planning: We'd like to plan a vacation. There are numerous options and various activities we're interested in, as well as time and cost constraints. Some activities are preferred to others, and we recognize we can only do so much within the allotted time.

These examples call for an approach for planning with goal preferences and constraints. The approach should be built upon a convenient formalism for specifying user preferences and constraints, together with efficient algorithm for

generating an optimal plan given the user's preferences and constraints. Work on over-subscription planning assumed real-valued rewards are associated with each goal proposition, and used specialized methods to optimize goal selection and then planned for them. In this paper, we suggest a different approach with distinct properties that are more suitable for certain domains. First, our approach is based on a qualitative preference specification language. This will be useful when dealing with lay users who are not willing to quantify their preferences using numbers. Moreover, our language allows for conditional preferences, so that the desirability of some goal proposition may depend on whether or not other goal propositions are satisfied, such dependencies cannot be represented with fixed goal weights. In addition, our algorithm combines the search for the preferred goals and the actual plan, and can prune one using the other, as well as prune goal choices without exploring the complete set of goal options.

Another benefit of our specification methodology is that it decouples the specification of the planning domain from the specification of the user's preferences. This is particularly desirable for consumer applications, and more specifically, in what we call *configuration planning*, of which vacation planning is a particular instance. In these domains, domain knowledge rests with the producer, e.g., travel agent, and is common to many consumers, whereas preferences are specific to each consumer. Moreover, the natural-language like preference statements used make preference elicitation from lay users possible.

Our algorithm is a synthesis between an algorithm that converts planning problem to a constraint satisfaction problem (CSP) and an algorithm for constrained optimization over CSPs in the context of TCP-nets – the preference specification formalism we use (Brafman & Domshlak 2002). We use a STRIPS-like deterministic action model, but in principle, our approach can handle any domain model that can be encoded as a CSP. We currently support only *goal* preferences and constraints (e.g. preferences and constraints over possible final system states). *Plan trajectory* preferences and constraints (e.g. preferences and constraints over possible actions and intermediate system states) cannot be expressed explicitly. We note, however, that it is possible to encode trajectory parameters within the system state and, thus, specify trajectory preferences and constraints implicitly as goal preferences and constraints.

The rest of this paper is organized as follows: In Section 2 we explain our preference specification language and briefly discuss GP-CSP((Do & Kambhampati 2001)) – a Graphplan-based planner that we utilized. Section 3 describes our algorithm, and demonstrates its operation on an detailed example. In Section 4 we describe the experimental framework we constructed for testing the efficiency of our algorithm, and provide some results. We conclude with related work in Section 5. For more details, see (Chernyavsky 2004).

Background

We start with a brief description of GP-CSP (Do & Kambhampati 2001) – the algorithm we use for converting plan-

(a) Planning graph

Variables: $G_1, G_2, P_1, \ldots, P_4$

Domains: $G_1 : \{A_1, \bot\}, G_2 : \{A_2, \bot\},$
$P_1 : \{A_3, \bot\}, P_2 : \{A_4, A_7, \bot\},$
$P_3 : \{A_5, \bot\}, P_4 : \{A_6, A_7, \bot\},$

Constraints: $P_1 = A_3 \Leftrightarrow P_2 \neq A_7$
(action) $\quad\quad P_1 = A_3 \Leftrightarrow P_4 \neq A_7$
$\quad\quad P_2 = A_4 \Leftrightarrow P_4 \neq A_6$

Constraints: $G_1 = A_1 \Rightarrow P_1 \neq \bot \wedge P_2 \neq \bot \wedge P_3 \neq \bot$
(activation) $\quad G_2 = A_2 \Rightarrow P_1 \neq \bot \wedge P_4 \neq \bot$

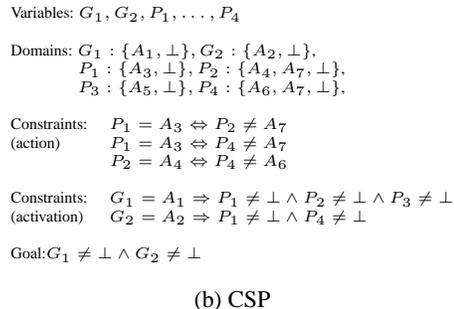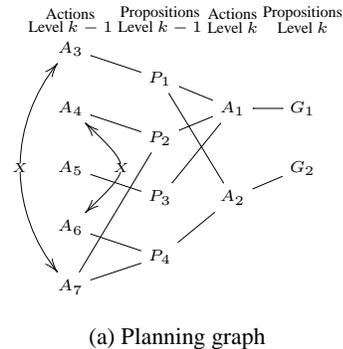Goal: $G_1 \neq \bot \wedge G_2 \neq \bot$

(b) CSP

Figure 1: A planning graph and the corresponding CSP

ning problem to an equivalent CSP problem. Then we describe the language and semantics of TCP-nets which are used to specify preferences.

GP-CSP

GP-CSP(Do & Kambhampati 2001) is an algorithm that automatically converts a planning problem into an equivalent CSP problem using a Graphplan encoding. The resulting CSP encoding is solved using standard CSP techniques.

The CSP is constructed as follows (we assume familiarity with Graphplan (Blum & Furst 1997)): a planning graph is constructed to a pre-specified length n. Each node in each propositional level of this graph is associated with a variable. The possible values of this variable correspond to the possible actions that have this proposition as an effect, as well as a null ($\bot$) value which encodes the fact that the proposition does not hold at this level. Thus, the variables in the CSP encode which propositions hold at different points in time, and what actions have caused each proposition to hold. The constraints now follow naturally. For instance, if p is made true by action a at level k, i.e., $p(k)$'s value is a, then the preconditions of a must have a value different than $\bot$ at time $k - 1$. Goal proposition g on level n of the planning graph must take non-$\bot$ value. Figure 1 illustrates this encoding on an example problem.

Preference Specification

An algorithm for planning with preferences and constraints requires a model for representing user preferences. A simple

solution would be to ask the user to rank the set of possible goal states, e.g., list all PCs in decreasing order of preference. This approach is tedious and does not scale up. Instead, we suggest a more structured compositional approach that is favored in work on preference elicitation. In particular, we utilize the preference statements used in TCP-nets, see (Brafman & Domshlak 2002). This choice seems appropriate because: (1) These statements are natural and intuitive and can be provided by lay users. (2) They support an efficient optimization algorithm which we have been able to adopt for our particular application.

Basically, the user describes her goal preferences using two types of propositional preference statements: (1) Conditional and unconditional preferences for the value of a variable. For example, "I prefer L1 to L2 as the final location of Package A," and "I prefer L1 to L2 as the final location of Package A, if the final location of Package B is L1." (2) Conditional and unconditional relative importance statements. For example, "Getting package A is more important than getting package B." Or "Getting A is more important than getting B if we got package C."

We demonstrate the power of these statements through a planning scenario:

Example 1 *Consider a logistics domain in which packages have priorities and dependent content. In particular, suppose we have five packages, $p1, p2, p3, p4$ and $p5$. $p1$, $p2$, and $p3$ contain legal documents for the same destination. These documents must be processed in the order $p1, p2, p3$. We prefer that $p1$ will arrive rather than not arrive. If $p1$ arrives, we prefer that $p2$ arrive, but if $p1$ does not arrive, we do not really care. Similarly, we prefer that $p3$ arrive if $p1$ and $p2$ have arrived. A similar dependency exists between $p4$ and $p5$. Finally, it is more important to deliver $p1$ than to deliver $p4$. (Formally, each p-i stands for some proposition of the form* at(package-i,destination-i)*.)*

Goal Preference Semantics TCP-nets organize the two classes of preference statements discussed above in the form of a graph. The user need not be aware of this underlying representation of her preference statements, but the graphical representation is very useful in consistency testing and plays an important role in the optimization algorithms introduced later. We refer the reader to (Brafman & Domshlak 2002) for a more formal description of TCP-nets. Here, we introduce them by means of an example.

A TCP-net is a graph whose nodes correspond the variables of interest (i.e., the possible goal conditions in our domain). The edges are used to provide information about preferential dependencies and the relative importance of variables. In Figure 2 we see a TCP-net over five binary variables A, B, C, D, and E. The graph shows that the preferences over the values of B depend on A's value, and those of D and E depend on B's value. These dependencies follow from the presence of conditional-preference edges (the regular edges) from A to B and from B to C and to D. The nature of this dependence is specified in the conditional preference table (CPT) associated with the appropriate node. For example, when B is *true*, we prefer that D will be *true*, too.

We also see an importance edge connecting B and E. This indicates that the value of B is more important to us than that of E, i.e., if we have to compromise one, we'd rather compromise on the value of E. Finally, there is an undirected edge between C and D. This indicates a conditional importance relation between these variables. Thus, sometimes C is more important than D, and sometimes D is more important than C. The relative importance of C and D is conditioned on the assignment to B and E, and this information is annotated on the edge from C to D. The precise dependence is shown in the associated conditional-importance table. For instance, we see that when B and E are assigned $b\bar{e}$ or $\bar{b}e$, then D is more important than C. When B and E are assigned $\bar{b}\bar{e}$, C is more important than D. Note that binary variables are used for simplicity, but there is no such restriction in the theory.
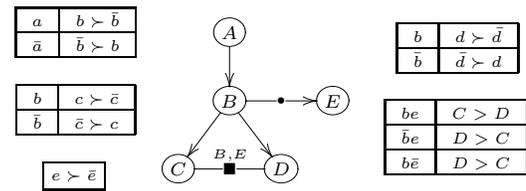

Figure 2: Illustrations for Example TCP-Net.

Figure 3 provides a TCP-net organizing the preference information in Example 1.

A TCP-net defines a *partial* order over the set of possible assignments to its variables. Each conditional preference table, importance relation, and conditional importance relation, adds some information to this partial order. This defines an initial partial order whose transitive-closure constitutes the semantics of the TCP-net. Note that a partial order may support a number of (Pareto) optimal elements, i.e., elements for which no other element is better.

The preference information in a TCP-net is interpreted under the *ceteris paribus* semantics as follows: The conditional preference table of variable X specifies the relation between any two complete assignments, o and o', that differ *only* in the value of X. To compare o and o' we simply look at X's table and check which one of them assigns X a more preferred value. This depends on the value of $Pa(X)$, which must be identical in both o and o'.

For example, according to Figure 2, $\bar{a}\bar{b}\bar{c}de$ is preferred to $\bar{a}bcde$ because $\bar{c}$ is preferred to c given $\bar{b}$, and the other attributes have identical values in both outcomes.
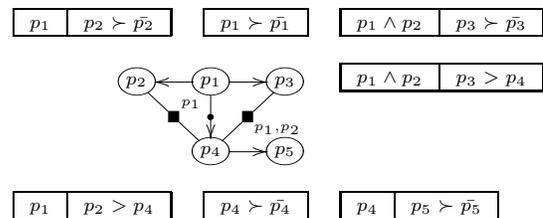

Figure 3: TCP-net for logistics domain.

Importance relation provide similar information. When X is more important than Y, we can compare any two outcomes o and o' that differ in the value of X and Y *only*. o is better than o' if o assigns X, the more important variable, a better value than o' assigns to Y. Conditional importance provides similar information but in a more restricted context, i.e., when the select set has the appropriate value.

For example, according to Figure 2, $abcd\bar{e}$ is better than $a\bar{b}cde$ because B is more important than E. Thus, it is better to get a less preferred value of E, as in $abcd\bar{e}$ than a less preferred value of B, as in $a\bar{b}cde$, all else being equal. Similarly, $abc\bar{d}\bar{e}$ is better than $ab\bar{c}d\bar{e}$. This follows from the fact that given $b\bar{e}$, C is more important than D. Thus, it is more important to get the preferred value for C than for D, all else being equal. However, we can't compare $a\bar{b}cd\bar{e}$ with $ab\bar{c}d\bar{e}$ directly, since we don't have an explicit importance relation between C and D when B and E are assigned $\bar{b}\bar{e}$.

We note that not all TCP-nets specify a consistent partial order, and that not all partial orders are specifiable by a TCP-net, see (Brafman & Domshlak 2002) for a more formal description of TCP-nets. In this paper we restrict our attention to the rich class of *conditionally acyclic* TCP-nets that are always consistent. This property can be verified automatically by an appropriate user interface.

Problem Formulation and Solution

We now provide a formal formulation of the planning with preferences problem, show how it can be solved efficiently and illustrate these ideas with an example.

Problem formulation

The constrained planning with preferences problem is defined as follows: Given a planning problem $\mathcal{P}$, user preferences $\mathcal{N}$, constraints $\mathcal{C}$ and plan length bound n, find a plan for $\mathcal{P}$ that satisfies all constraints in $\mathcal{C}$ and is an optimal such plan with respect to $\mathcal{N}$ among all plans satisfying $\mathcal{C}$. Plan-length bound can be viewed as a simple form of resource constraint. In this particular paper, we consider the following concrete elements:

$\mathcal{P} = \langle D, I \rangle$ - A STRIPS planning problem, where D is a deterministic, propositional domain theory, and I is an initial state. For example, D = logistics domain; $I = \{$`at_truck1_PORT`, `empty_truck1`, `at_pack1_AIRPORT`$\}$. Note that this specification contains no goal formula.

$\mathcal{N}$ - a conditionally acyclic binary TCP-net, where each variable represents a possible goal proposition for $\mathcal{P}$. The value assigned to each variable corresponds to the achievement of the equivalent proposition by a plan. For example, the preference order $t \succ f$ for variable `at_pack1_PORT` implies that we prefer plans that achieve goal G such that `at_pack1_PORT` $\in G$. Hence, each outcome represents a goal. In the following, we use the terms *outcome* and *goal* interchangeably.

$\mathcal{C}$ - constraints over goal state propositions. We provided several convenient constraint schemata. For example, the constraint `exclusive([at_pack1_PORT,at_pack1_AIRPORT])`

ensures that exactly one variable of the specified list of goal variables holds true.

Finally, the notion of optimality we use is Pareto-optimality with respect to the preference order specified by $\mathcal{N}$, i.e., a goal is (Pareto) optimal if no other goal is strictly better than it (with respect to $\mathcal{N}$).

Solution algorithm

PREFPLAN, the solution algorithm we propose, performs a TCP-net based optimization over CSP encoding of the problem. PREFPLAN builds upon the plan-graph based CSP encoding described in (Do & Kambhampati 2001). We extend this encoding and adapt the original CSP solver for interaction with the TCP-net by imposing a variable instantiation order that is consistent with the TCP-net. We believe that this modularity of our approach is an important advantage, as other researchers using a constraint-based planner can add a preference module on top of it, with minor modifications to the actual planner – much as we did.

The PREFPLAN routine (Pseudocode 1) first constructs a CSP encoding of the problem (lines 2-3). Unlike GP-CSP, which forces n'th level goal propositions to be active (i.e. supported by actions), we allow all n'th level propositions to be inactive (i.e. assigned $\perp$ value). This way, different goal combinations could be considered. In line 3, user specified goal state constraints are appended to the CSP problem in a straightforward fashion. In line 4, the extended CSP encoding is submitted to the modified CSP solver. The rest of the code demonstrates portions of the CSP solving algorithm that have been modified in order to interface the TCP-net structure. They are discussed below:

- Variable ordering (lines 7-12). Variable are assigned in the following order:

  1. Variables that appear both in the TCP-net and the n'th level of the planning graph (we designate this group of variables by A) are instantiated first, according to a preference order consistent with the TCP-net. This is achieved by always selecting a TCP variable with no parents (Boutilier *et al.* 2004).

  2. Other variables that appear in the planning graph (group B) are instantiated in an arbitrary order (any CSP heuristic may be used).

- Domain ordering (lines 13-17). For a variable v selected for instantiation by nextVar routine, the domain ordering is as follows:

  - for $v \in A$ such that $t \succ f$ for v (note that this is well defined since any conditioning variable of v must have already been assigned) we attempt to support v by an action, leaving $\perp$ as the least desired alternative.

  - otherwise, $\perp$ is the preferred value.

- Assignment/unassignment (lines 18-27). Upon assigning/unassigning a variable v such that $v \in A$, we eliminate this variable from the TCP network by removing the connected edges and reducing dependent CPT and CIT tables (Brafman & Domshlak 2002). We keep record of these modifications in order to be able to restore them upon backtracking.

Pseudocode 1 PREFPLAN

function PREFPLAN /*the entry point*/
Input: $\langle \mathcal{P}, \mathcal{N}, \mathcal{C}, n \rangle$
Output: A plan that achieves a Pareto-optimal (non-dominated) goal with respect to $\mathcal{N}$.

1: expand planning graph pg for $\mathcal{P}$ up to level n.
2: Build the GP-CSP encoding csp for pg; skip goal constraints.
3: $csp \leftarrow csp \cup \mathcal{C}$

4: $solution \leftarrow$ modified-CSP-solver(csp) /*perform search*/
5: $plan \leftarrow$ process($csp, solution$) /*extract plan from CSP problem solution*/
6: return $plan$

function nextVar /*choose next variable for assignment*/
Output: next variable
7: $TCPvars \leftarrow \{v | v \in vars(\mathcal{N}),$
    v has no parents and no ci-edges connected$\}$
8: **if** $TCPvars \neq \emptyset$ /*TCP-net is not fully assigned*/
    then
9:   $v \leftarrow$ select($TCPvars$)
10: **else**
11:   $v \leftarrow$ select($\{$all unassigned variables$\}$)
12: **end if**

function orderDomain /*order values of the variable to be assigned (the variable returned by nextVar)*/
Input: var
13: **if** $var \in vars(\mathcal{N})$ and TCPpreference($\mathcal{N}, var$) = "$true \succ false$" **then**
14:   order $\perp$ after actions
15: **else**
16:   order $\perp$ before actions
17: **end if**

function uponAssign /*upon CSP variable assignment*/
Input: var, val
18: **if** $var \in vars(\mathcal{N})$ **then**
19:   **if** $val = \perp$ **then**
20:     TCPreduce($\mathcal{N}, var, false$)
21:   **else**
22:     TCPreduce($\mathcal{N}, var, true$)
23:   **end if**
24: **end if**

function unassign /*upon CSP variable unassignment*/
Input: var
25: **if** $var \in vars(\mathcal{N})$ **then**
26:   TCPrestore($\mathcal{N}, var$)
27: **end if**

Example

Consider a problem, where the domain model, initial state and plan length bound yield the planning graph shown in Figure 4(a). Preferences and constraints description appears in Figures 4(c), 4(d). Variables G_X, G_Y, G_Z on the last level of the planning graph, correspond to TCP variables X, Y, Z respectively. The CSP encoding of the planning graph together with user constraints is depicted in Figure 4(b) . The algorithm proceeds as follows:

1. Variable X is selected by the TCP-net as the only variable with no parents (Figure 5(a)).

   (a) CSP domain of G_X is ordered as follows: $\{A_1, \perp\}$.
   (b) Assignment $G_X = A_1$ fails during Forward Checking against G_Y (the only possible value for G_Y is inconsistent with this assignment).
   (c) Assignment $G_X = \perp$ is consistent.
   (d) The TCP-net is reduced (Figure 5(b)).

2. Variable Y is selected by the TCP-net (Figure 5(b)).

   (a) A_3 is the only possible value for G_Y.
   (b) Assignment $G_Y = A_3$ is consistent.
   (c) The TCP-net is reduced (Figure 5(c)).

3. Variable Z is selected by the TCP-net (Figure 5(c)).

   (a) CSP domain of G_Z is ordered as follows: $\{\perp, A_2, A_4\}$.
   (b) Assignment $G_Z = \perp$ is consistent.
   (c) The TCP-net is fully assigned now .

4. At this point all group A CSP variables (participating in preference statements) are assigned . Now group B CSP variables (G_W, P_1, P_2, P_3) are to be assigned. A satisfying assignment exists for these variables, given the assignments done so far. Hence, the TCP-net will not be consulted again and the solver will eventually return a plan that achieves the goal $\{G_Y\}$ that is pareto-optimal with respect to the TCP-net . Note, that in principle, a backtracking to a group A variable could have occurred. In this case, we would have unassigned all intermediate variables and continued in same fashion as described above.

In the longer version of this paper (Chernyavsky 2004), we prove that PREFPLAN is sound, complete, and moreover, optimal, i.e., the plan returned is Pareto optimal with respect to the user's preferences.

An important property of PREFPLAN is that both goal-space and plan-space search are encoded as a single CSP problem. As a result, goal search tree branches are being pruned during search, due to conflicts of goal variables with other plan variables. Moreover, certain sub-goal combinations are recognized as no-goods early on, and prevent us from investing time in any planning for goal states that contain them. In the example we discussed, the goal-space branch corresponding to the assignment $X = x$ is pruned (Figure 6). In contrast, the *Generate and Test* approach (i.e., generate the set of all possible goals in a non-increasing order until an achievable goal is found) would attempt to find a plan for each one of the four goals in this redundant branch. As the goal space grows, entering redundant
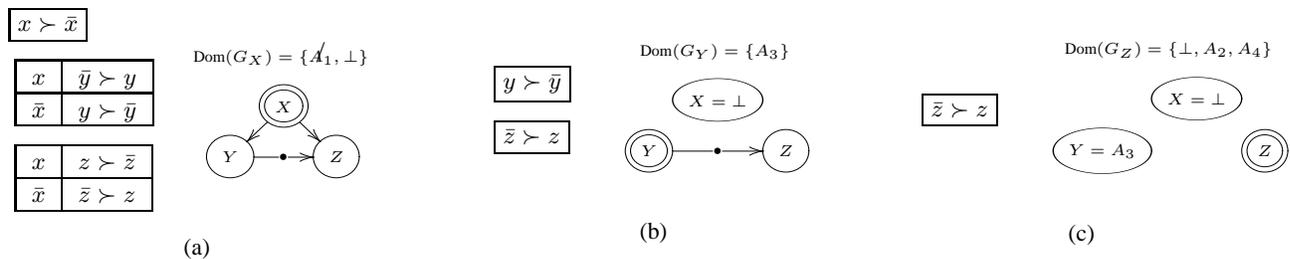
Figure 5: Algorithm execution.

search tree branches may become computationally expensive. Hence, we expect our algorithm to be more efficient than algorithms that separate goal selection from planning.

Experiments and Results

We implemented PREFPLAN on top of the GP-CSP planner, defined an interesting new planning-configuration domain to test it on, and run a large number of experiments to evaluate its performance. Our implementation is described in more detail in (Chernyavsky 2004). Our domains and experimental results are discussed below. The implementation and testing suit are available at: http://www.cs.bgu.ac.il/~yurac.

Configuration Planning Domains

We constructed a set of planning domains and problems, based on the idea of configuration planning discussed earlier. The configuration problem is to select the sub-components of a final product given certain constraints on component compatibility, and possibly preferences between alternative sub-component choices. We introduce to this problem the actual planning required for the assembly process. We note that the introduction of the planning element into configuration problem is an interesting contribution to the configuration literature which is mainly concerned with the problem of component selection for the final product, but does not explicitly model the "assembly" process.

We developed a "configuration-domain" template. This template comprises of the following elements:

- item types: each item can be either atomic, or may require other items for assembly. The relationships between items can be presented as an acyclic, directed graph with the atomic item types as leaves.
- machine types.
- assembly action for each non-atomic item type. Assembly may require a machine, or a number of machines of different types.

Items can be moved between locations. Machines are static. Machine may need to be warmed up before use. Machine can participate in at most one assembly task at a time. To define a concrete problem we must declare all items, machines and locations in the system and specify initial state of each actor.

For a given configuration problem, our task is to construct a final product, using available items as building blocks. Final product configuration is determined by the item types that appear in the product. Among the item types we distinguish between those that may appear in the final product and those that are used to assemble items of the former type. The set of possible final product configurations is viewed as the set of possible planning goals. For example, for some configuration domain in which the item types are A,B,C,D, and the item types that may appear in the final product are B,D, the possible goals are:

$\{\{B,D\},\{\neg B,\neg D\},\{\neg B,D\},\{B,\neg D\}\}$,

where each goal corresponds to a specific final product configuration.

We constructed an automated tool for creating random product configuration domains and problems and defined several input parameters that influence problem complexity. The parameters include:

- maximal depth of item assembly process.
- number of shared subitems and sharing degrees (how many item types need this subitem type for assembly).
- percent of items that require a machine for assembly.

Experiments

We run experiments using our domain generator. The results reported in this paper were obtained on a Intel PIII 500 MHz machine with 384MB Ram running Linux. To assess the performance of our algorithm we compared it to two other algorithms: The first is the oracle-based direct planning algorithm. This algorithm obtains an optimal achievable goal state from an oracle and plans for it directly. It is useful as it allows us to assess the approximate cost of the plan-space search. The second algorithm is a *Generate-and-test* algorithm that generates the set of all goals in a non-increasing preference order (the theory of TCP-nets provides an algorithm for doing so (Boutilier *et al.* 2004)). The *Generate-and-test* algorithm, which uses GP-CSP as the underlying planner, re-uses the same planning graph, so that reachability information is not recomputed. However, we did not reuse no-goods from one search to the other, since (Do & Kambhampati 2001) show that this modification would lead to performance degradation due to increased complexity of no-good bookkeeping.
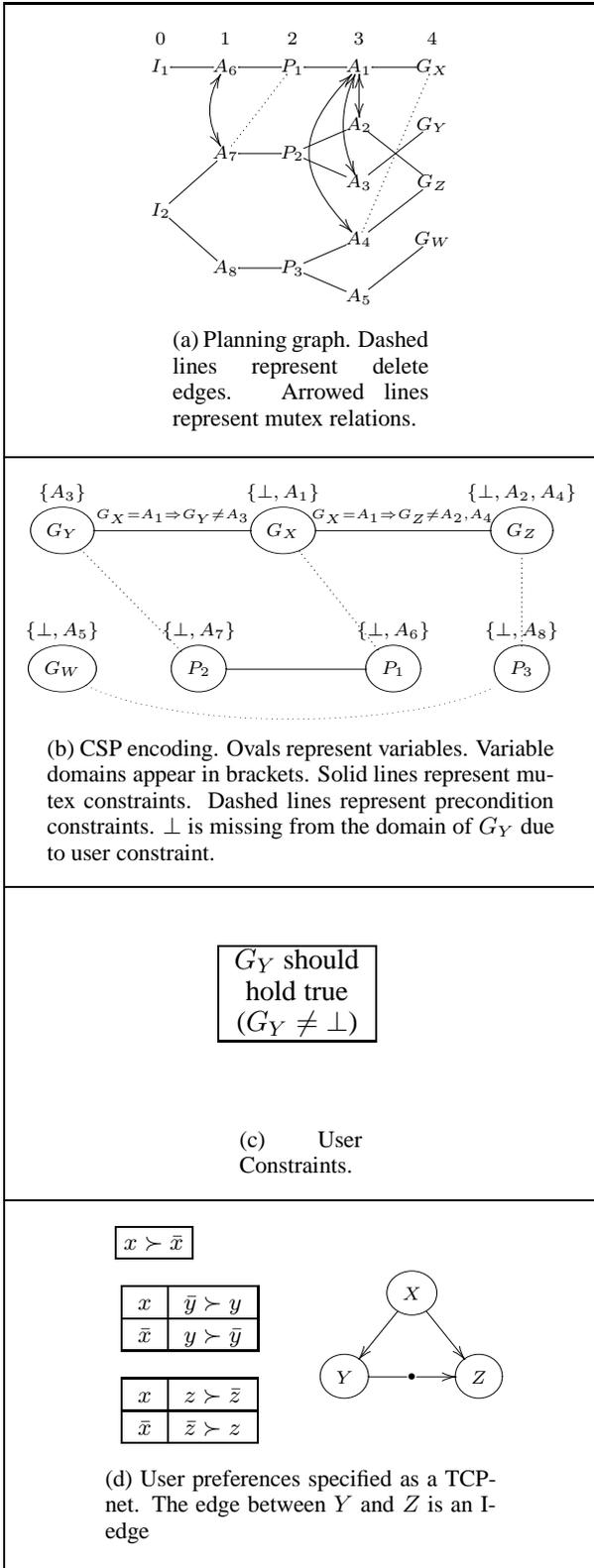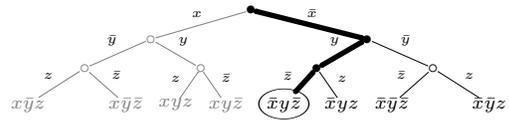
Figure 6: Goal search tree

We estimate and compare performance, in terms of the following factors:

- Plan search time. We compare only time spent during search. All three approaches construct equivalent planning graphs, and thus, spend the same time in planning graph construction. Note that CPU time is machine specific and depends on processor load.

- Total number of constraint checks performed by the CSP solver during search. Unlike plan search time, this measure is machine independent and reproducible.

We solve each problem by all three algorithms and measure running times ($time$ in tables) and number of constraint checks ($checks$) performed by each algorithm. Moreover, for each problem we count the number of goal sets tried by *Generate-and-test* algorithm before an achievable goal was found ($numgoals$). Note, that we regard only goal sets for which actual planning was performed. We don't count goal sets that aren't reachable, or violate Graphplan's mutex relations, as such goals are discarded immediately without planning.

Results

We defined 15 different sets of parameters. For each scenario, we generated 20 random domain-problem pairs. For each scenario, we used a single set of user preferences, and a single reasonable plan-length bound. Note, that a single set of user preferences has different meaning when applied to different domains.

Tables 1, 2 summarize our experimental results. Table 1 shows average times and numbers of constraint checks. Table 2 shows the relationship between number of nontrivial goals tried (by the *Generate-and-test* algorithm) and the performance of the algorithms in terms of number of constraint checks.

Observations:

- Our experiments show that for many, even trivial problems, identifying a goal as unachievable during planning, may be very expensive computationally. For most of our examples, finding a plan for an achievable goal took significantly less time than an attempt to find a plan for an unachievable goal (see Table 1 columns 3,6). We conclude, that it is important to eliminate unachievable goals whenever possible, without planning.

- On all instances, our algorithm exhibits significantly better performance than the *Generate-and-test* solution method. We attribute this to the fact that our algorithm utilizes information gained during the search for a plan for one goal, while

(a) Planning graph. Dashed lines represent delete edges. Arrowed lines represent mutex relations.

(b) CSP encoding. Ovals represent variables. Variable domains appear in brackets. Solid lines represent mutex constraints. Dashed lines represent precondition constraints. $\perp$ is missing from the domain of G_Y due to user constraint.

G_Y should hold true
$(G_Y \neq \perp)$

(c) User Constraints.

$x \succ \bar{x}$

| x | $\bar{y} \succ y$ |
| $\bar{x}$ | $y \succ \bar{y}$ |

| x | $z \succ \bar{z}$ |
| $\bar{x}$ | $\bar{z} \succ z$ |

(d) User preferences specified as a TCP-net. The edge between Y and Z is an I-edge

Figure 4: Example problem

# scenario	time[sec.] (avg.)			checks (avg.)			numgoals (avg.)	$checks3/(numgoals-1)$ (avg.)
	PREFPLAN ($time1$)	Oracle ($time2$)	gnt ($time3$)	PREFPLAN ($checks1$)	Oracle ($checks2$)	gnt ($checks3$)		
1	0.42	0.01	1.21	233809	3741	1124941	10	114207
2	2.60	0.10	7.65	1086670	70493	2973995	5	653625
3	10.37	0.32	215.61	4619469	337787	69228710	16	4554520
4	2.83	0.20	12.71	1814943	225664	7125316	5	1436555
5	34.88	0.20	133.65	8878274	214215	45395780	19	2487440
6	0.13	0.03	1.02	81264	16670	999487	8	128303
7	0.49	0.08	2.26	409697	81307	2620939	14	187880
8	3.44	0.08	69.61	1959888	90313	19178545	22	901671
9	2.92	0.19	3.88	1670667	172040	2887338	6	515596
10	2.11	0.26	3.78	999126	154251	1918761	4	581442
11	19.31	0.43	46.56	7515027	316297	31237698	13	2541716
12	36.85	0.42	72.01	7738893	277661	26696087	16	1744842
13	15.37	0.72	53.61	28377081	1718833	92856675	8	13078404
14	25.82	1.58	151.51	27758063	2282725	113798948	9	13080338
15	11.37	0.91	59.43	11532309	1078066	61450697	11	5689879

Table 1: Number of constraint checks and number of nontrivial goals tried

$numgoals$[range] (avg.)	checks (avg.)			$checks3/checks1$ (avg.)
	PREFPLAN ($checks1$)	Oracle ($checks2$)	gnt ($checks3$)	
0-11	5941572	482780	20413877	3.44
12-23	9620704	543408	47333659	4.92
24-35	4493323	150987	42155946	9.38
36-47	11425309	166285	121766702	10.66

Table 2: Number of constraint checks for different values of $numgoals$

searching for a plan for subsequent goals. Moreover, it employs effective CSP techniques, such as forward checking, arc-consistency checking and conflict-directed backjumping not only in search for a plan for a given goal, but also in the search in goal space. Therefore, our algorithm is able to prune branches of the goal search tree, consisting only of unachievable goals.

The *Generate-and-test* approach, on the other hand, doesn't keep information related to failure to achieve previous goals, and thus, is prone to entering redundant branches of its goal search tree.

- As we might intuitively expect, Table 2 shows that both the CSP-based algorithm and the *Generate-and-test* algorithm usually spend more effort searching for a plan on problems where more nontrivial goals must be considered. Our algorithm demonstrates better performance relative to the *Generate-and-test* algorithm, as the number of goals grows (see column 5).

Discussion

We motivated the need for a richer model for reasoning about user preferences within the classical planning framework. We suggested one particular formalism for specifying user preferences, that is simple, yet relatively expressive. This formalism has the advantage that it is accessible and intuitive to a large class of users. For instance, it is easy to envision scientists working with the Mars space rover specifying preferences over target sites using our preference language, as well, as consumers seeking some customized product. We developed a model for planning with preferences and constraints that relies on this formalism, and constructed an effective CSP-based algorithm for solving the corresponding constrained optimization problem. Our approach can be used with any other reduction of planning to CSP, and although we concentrated on goal constraints, it can incorporate any plan constraints that the underlying constraint solver is able to handle. In future work, we hope to incorporate resource constraints and preferences over plan (rather than goal) properties into PREFPLAN.

Related Work

Our work is closely related to recent work on over-subscription planning (Briel *et al.* 2004; Smith 2004), i.e., planning with a large set of goals that cannot all be achieved. Our work differs from both papers in its preference-representation and its algorithmic approach. We use the language of TCP-nets to express preference, whereas they associate real values with sub-goals. Real-valued weights allow for quantitative preferences but cannot express conditional preferences. Conditional preferences are very important because they let us capture context-dependent value for goals, such as "taking an image of this rock is important only if some analysis of its content was performed". Qualitative preferences also appear more natural for lay users to specify.

Algorithmically, Smith takes a two tiered approach – solve an abstracted problem to determine the sub-goals, and then plan for them, while Briel *et al.* propagate cost information in the planning graph construction phase, and use that to guide the selection of sub-goals. In essence, both techniques heuristically select the goal set for which to plan, and then attempt to plan for it. Thus, they do not guarantee optimality. This approach bears certain similarity to the (qualitative) *Generate and Test* algorithm we examined which greedily selects the most promising combination of sub-goals that are not ruled out by the planning-graph. Our approach, based on a constrained optimization algorithm, is guaranteed to generate an optimal solution. It searches over a space that combines both goals and plans.

Utility - directed planning. Decision theoretic planning provides a rich notion of plan quality and allows to construct expressive quantitative utility models. In decision-theoretic planning goals are not defined explicitly, and the planner attempts to find a plan that maximizes expected utility. The TCP-net based preference model we use does not specify goal formulas explicitly either. However, it is qualitative, and hence less rich, although easier to specify and elicit from users.

The MO-GRT planning system (Refanidis & Vlahavas 2001), extends the GRT heuristic planner to handle plan cost functions with multiple criteria. The planner performs a state-space goal-satisfying search, guided by A*-like heuristic, based on plan cost estimate. Costs are represented by vectors, where each dimension represents the numeric value of one criterion. For each action, a description of its effects on cost criteria, are added to action definition. Cost estimates for each state are computed once during problem preprocessing, in a single backward pass. The estimates are then used to guide the state-space forward search. At present, we do not handle multiple resource constraints. On the other hand, our preference model can handle more complex conditional and importance relations, and decouples the preference specification from the domain model.

(Eiter *et al.* 2002) describe an approach for planning based on answer set programming. Planning problems are translated to answer set programs and solved by existing answer set solvers. (Son & Pontelli 2004) provides a declarative, high level language, $\mathcal{PP}$, for expressing planning preferences that can be integrated with this framework and supports nontrivial plan trajectory preferences. It can also express importance orders between different preference criteria combinations. To the best of our knowledge, this approach has not been empirically tested, yet. We cannot handle complex preferences over plan trajectory, whereas $\mathcal{PP}$ does not handle conditional preferences and importance relations.

The plan utility model of (Haddawy & Hanks 1993) uses an extended goal condition for planning problems. Goal specification consists of a goal formula and an a-temporal goal component which describes time restrictions for satisfying the goal formula. Plan quality is determined by the degree of satisfaction of both goal components and the amount of system resources consumed. PYRRHUS (Williamson & Hanks 1994) implements this model by extending a classical planning algorithm to solve the resulting optimization problem using branch-and-bound search.

Our planner can deal with complex, conditional preference statements that the above model cannot handle. In this work we don't address temporal deadlines. We note however, that our planner can be adapted to handle temporal conditions if the underlying CSP solver supports them.

(Rabideau, Engelhardt, & Chien 2000) describe an optimization framework for the ASPEN planning model. Preference is defined as a mapping from a plan variable (e.g. resource level, goal count, etc.) to a quality metric. The optimization algorithm repeatedly selects a preference with a low score and performs a plan modification that could potentially improve the score. Such modifications may introduce plan conflicts, in which case ASPEN plan repair step is invoked. Different preferences are treated independently. Therefore, each improvement step for one preference may result in score decrease for other preference, or lower overall plan score. (Estlin & Gaines 2002) adapt this algorithm to handle problems with goal interdependencies. For this purpose, numeric rewards are defined for individual goals and goal combinations, as a part of problem specification. This approach does not guarantee optimality, whereas our algorithm returns an optimal plan.

Constraint-Based Planning Several methods have been proposed for encoding planning problem as a constraint satisfaction problem and solving it using existing algorithms. These include BLACKBOX (Kautz & Selman 1999), GP-CSP (Do & Kambhampati 2001), and the planner of (Lopez & Bacchus 2003). Preferences can be handled once we move from hard CSPs to soft-CSPs. (Bistarelli *et al.* 2001) present a formalism that generalizes most soft-CSP approaches. User preferences form a semiring $\langle A, +, \times, 0, 1 \rangle$, where A is the set of the possible preference values, $\times$ is a combination operator, $+$ is a projection operator that is used to compare preference values, 0, 1, are minimal and maximal preference levels respectively. For example, the semiring $\langle \{true, false\}, max, min, false, true \rangle$ can be used to represent satisfaction degrees of hard constraints. (Bistarelli *et al.* 2001) adapt many CSP solution techniques, including arc-consistency enforcement and forward checking, to semiring-based soft CSPs. The application of these results to planning with preferences was not discussed. We consider it as an important future research topic.

(Miguel, Jarvis, & Shen 2001) use soft constraint satisfaction in the context of optimal planning. The planning problem is encoded as a variant of soft CSP, Flexible CSP. FGP supports a rich domain model using fuzzy propositions. A degree of satisfaction is associated with different goals and different operators. The degree of satisfaction of a plan is the minimal satisfaction level of the operators used and the goal achieved. Our preference specification language is able to capture complex conditional relations that this approach cannot handle. In addition, unlike FGP, our preference specification model is decoupled from the domain model. This is advantageous when we want to integrate a single, fixed domain model with different objectives.

References

Bistarelli, S.; Frhwirth, T.; Marte, M.; and Rossi, F. 2001. Soft constraint propagation and solving in constraint handling rules.

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

Boutilier, C.; Brafman, R.; Geib, C.; and Poole, D. 1997. A constraint-based approach to preference elicitation and decision making. In *Proc. AAAI'97*, 19–28.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H.; and Pool, D. 2004. Preference-Based Constrained Optimization with CP-Nets. *Computational Intelligence, Special Issue on Preferences*. to appear.

Brafman, R., and Domshlak, C. 2002. Introducing Variable Importance Tradeoffs into CP-Nets. In *Proc. UAI'02*.

Briel, M.; Sanchez, R.; Do, M.; and Kambhampati, S. 2004. Effective Approaches for Partial Satisfaction (Over-Subscription) Planning. In *Proc. of AAAI'04*.

Chernyavsky, Y. 2004. Planning with goal preferences and constraints. Master's thesis, CS Dept., Ben-Gurion University.

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

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2002. Answer set planning under action costs.

Estlin, T., and Gaines, D. 2002. An Optimization Framework for Interdependent Planning Goals. In *Proc. AIPS'02*, 21–28.

Haddawy, and Hanks, S. 1993. Utility models for goal-directed decision theoretic planners. Technical Report TR-93-06-04, University of Washington, Department of Computer Science and Engineering.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. IJCAI'99*.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a CSP. In *Proc. IJCAI'03*.

Miguel, I.; Jarvis, P.; and Shen, Q. 2001. Efficient flexible planning via dynamic flexible constraint satisfaction. *Engineering Applications of Artificial Intelligence* 14(3):301–327.

Rabideau, G.; Engelhardt, B.; and Chien, S. 2000. Using Generic Preferences to Incrementally Improve Plan Quality. In *Proc. AIPS'00*.

Refanidis, I., and Vlahavas, I. 2001. Multiobjective heuristic state-space planning.

Smith, D. 2004. Choosing Objectives in Over-Subscription Planning. In *Proc. of ICAPS'04*.

Son, T. C., and Pontelli, E. 2004. Planning with Preferences using Logic Programming. In *Proc. LPNMR'04*, 247–260.

Tsang, C. 1993. *Foundations of Constraint Satisfaction*. Academic Press, New York.

Williamson, M., and Hanks, S. 1994. Utility-directed planning. In *Proc. AIPS'94*.