Using Generic Preferences to Incrementally Improve Plan Quality

Gregg Rabideau, Barbara Engelhardt, Steve Chien

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 126-347
Pasadena, California 91109-8099
firstname.lastname@jpl.nasa.gov

Abstract

We describe a methodology for representing and optimizing user preferences on plans. Our approach differs from previous work on plan optimization in that we employ a generalization of commonly occurring plan quality metrics, providing an expressive preference language. We introduce a domain independent algorithm for incrementally improving the quality of feasible plans with respect to preferences described in this language. Finally, we experimentally show that plan quality can be significantly increased with little additional modeling effort for each domain.

Introduction

Traditionally, AI planning has focused on generating feasible plans that satisfy a set of goals. However, in many domains it is insufficient to simply model the hard constraints of the system. Numerous undesirable, yet executable plans may exist which satisfy the goals. For example, unnecessary movements may be arbitrarily inserted into a robot's plan, as long as it ends in the goal position. In addition, strict feasibility constraints may be too weak for most problems, but necessary for completeness. For example, while it may be physically possible to completely drain a robot's battery, reasons of risk and longevity will make it preferable to maintain a certain level of charge. However, this preferred charge level, if encoded as a hard constraint, would preclude solutions where a full battery drain was necessary. Feasible plans may have a continuous measure of quality and only a subset may be considered acceptable. In the robot example above, quality continuously improves for plans with fewer movements and less battery drain. In an over-constrained system, quality may improve as more goals are satisfied. We need to be able to evaluate plan variables at a finer granularity than simply as consistent or violated. To achieve this, we build on the traditional representation of discrete *hard constraints* and mandatory goals to include continuous *soft constraints* (i.e.,

preferences) and optional goals. In other words, we extend the notion of what the plan *must* accomplish (and how) to what we *prefer* the plan to accomplish (and how). In this way, the user can specify which feasible solutions are more desirable, establishing a basis for automatically generating high quality plans.

In many NASA domains, the user can have a complicated definition of plan quality. For example, scientists typically would like to complete as many experiments as possible within given windows of opportunity. Other users, such as engineers, might have a preference for fewer power cycles of a spacecraft instrument in hopes of extending the life of the instrument. Certain system states may be more desirable than other states. For example, extending an arm of a rover might make the rover less stable, suggesting a preference to keep the arm stowed when not in use. Some timing constraints may be flexible but also have a preferred time. For example, a calibration may be most useful immediately before an experiment, but still have some utility up to five minutes earlier. As another example, an experiment may have several different ways of collecting data, each resulting with different levels of data quality. We present a general representation of plan quality that is capable of encoding a wide range of preferences including the ones just described.

We implement our representation of plan quality in the ASPEN planning and scheduling system (Fukunaga et al. 1997). In addition, we demonstrate our approach to plan optimization using a generalization of a technique called *iterative repair* (Minton and Johnston 1988; Zweben et al. 1994). In ASPEN, the main algorithm for generating feasible plans is a local, early-commitment approach to iterative repair (Rabideau et al. 1999). During repair, the conflicts in the schedule are detected and addressed one at a time until no conflicts exist, or a user-defined time limit has been exceeded. A conflict is a violation of a plan constraint, and can be resolved by making certain modifications to the plan. The most common plan modifications include moving an activity, adding a new instance of an activity, and deleting an activity. For each conflict, a domain-independent repair expert automatically generates modifications that could potentially repair the conflict.

We adopt a similar local, early-commitment, iterative approach to optimization. During *iterative optimization*, low scoring preferences are detected and addressed individually until the maximum score is attained, or a user-defined time limit has been exceeded. A preference is a quality metric for a plan variable, and can be improved by making modifications to the plan similar to repair. For each preference, a domain-independent improvement expert automatically generates modifications that could potentially improve the preference score. For example, minimizing tardiness is a preference on the end time variables of activities and can be improved by moving activities to earlier times.

The iterative optimization algorithm has many of the same desirable properties as iterative repair. First, both algorithms take advantage of previous planning by starting with the current plan. Solutions may be disrupted by manual modifications or by automatic updates from unexpected differences detected during execution. Repairing or improving the existing plan enables a fast turn-around time when small changes create conflicts or degrade plan quality. Second, local search algorithms do not incur the overhead of maintaining intermediate plans or past attempts[1]. This allows the planner to quickly try many plan modifications for repairing conflicts or improving quality. Indeed, local stochastic search methods have been shown to be very effective for hard planning problems (Kautz and Selman 1996). However, unlike systematic search algorithms, local search cannot guarantee that all possible combinations of plan modifications will be explored or that unhelpful modifications will not be retried. Finally, reasoning with uninstantiated variables, such as activity start times and resource usage, can increase the complexity of planning systems (Wilkins 1988). The temporal relationships and resource profiles of activities with instantiated variables can be computed more efficiently. Least-commitment techniques retain plan flexibility and can reduce the number of search nodes, but the cost per search node can be high. Further discussions with applications to spacecraft commanding can be found in (Chien et al. 1998).

In the next section, we describe the ASPEN planning model. Then, we describe an extension to this model for representing plan quality. Next, we present one possible algorithm for optimization that uses this representation. Finally, we introduce four realistic NASA domains with complicated quality metrics for which we have easily encoded preferences and quickly improved plan quality.

The ASPEN Planning Model

There are many variables of a plan that must be considered during the planning process. Some of these variables play a role in defining the feasibility of the plan. The set of plan constraints identifies these variables and the values required for successful execution.

In ASPEN, we have adopted a planning model with an explicit representation of constraints for time, resources and states (Smith et al. 1998). Plan operators, called activities, have a set of local variables including a start time and duration. Activities may have a set of temporal constraint variables, each specifying a minimum and maximum separation between two activities in the plan. Activities also share a number of global resources or state variables. Local constraint variables may be defined in an activity specifying the required value of a resource or state variable for the activity. The combined effects of the activities define the time-varying profiles (i.e., timelines) for the values of the resources and state variables. Global constraints can be defined for each timeline, limiting its set of legal values. For resources, these are capacity constraints. For state variables, the set of legal state transitions can be specified. The ASPEN planning model also includes a representation for activity hierarchies. Activities can have a disjunctive set of decompositions, each of which expands the activity into different set of sub-activities. Each sub-activity may also have its own decompositions. A local variable represents the currently selected decomposition. Arbitrary functional relationships can be expressed between any of the variables in the activities. This allows ASPEN to make external calls to special reasoning modules for calculating plan values, if necessary.

Finally, ASPEN has an explicit representation of mandatory and optional goals. Goals are simply activity specifications that do not immediately appear in the plan. A mandatory goal is a conflict until the activity has been inserted into the plan (i.e., the goal is satisfied). Optional goals are not considered conflicts but instead degrade plan quality when not satisfied.

Representing Plan Quality

We define preferences as quality metrics for variables in complete plans. Preferences provide a mechanism for specifying which plan variables are relevant to plan quality. Certain values of these variables are preferred over others, without regard for legality. Variables for preferences may be selected from local variables of activities or from global variables representing features of the plan as a whole. We define a set of preference classes that directly corresponds to the set of plan variable classes.

```
Prefer linearly less order wait_time
Prefer linearly more observation
  occurrences between 1 and 30 weighted 10
Prefer linearly more battery min value
Prefer linearly more earth_point duration
```

Figure 1: Example preferences in ASPEN.

---

[1] Some success has been shown in storing a limited history, such as with tabu lists (Mazure, Sais and Gregoire 1997).

Preference Variables

To better understand what types of preferences are included in our semantics, we must describe the types of plan variables that can contribute to plan quality. We define five basic types:

- local activity variable
- activity/goal count
- resource/state variable
- resource/state change count
- state duration

An activity variable preference indicates a ranking for the values of a local variable in an activity instance in the plan. Local activity variables include domain-specific variables as well as internal variables for start time, end time, duration, resource usage, temporal distance from other activities, and selected decomposition. Typically, a preference is made for variables with a particular name defined in a particular type of activity. For example, minimizing tardiness in (Williamson and Hanks 1994; Miyashita and Sycara 1995) is a preference on the end times of activities that fulfill factory orders. Minimizing work in process (WIP) is a preference on the distance between the order request and order fulfillment activities (see Figure 1). Other preferences can score the plan based on the number of existing activities of specific types (i.e., activity count). Or, one can make a general preference for satisfying more of the optional goals. In a typical spacecraft domain, scientists prefer to include as many observation activities as possible in a limited window of opportunity.

A preference can also be made for certain values of a global resource or state variable. A resource/state variable preference ranks the set of resource/state values that exist within the planning horizon. For example, a preference can be made for maximizing the minimum value of a battery over time. Other preferences can score the plan based on the number changes occurring on a resource/state variable (i.e., resource/state change count). This type of preference could be used to limit the number of power spikes on the battery. Finally, a preference can be made on the duration of a particular state on a state variable. Pointing a spacecraft antenna towards earth, for example, is preferred when the spacecraft is not constrained to any other state.

Mapping to Quality Metrics

A preference is a mapping from a plan variable to a quality metric (i.e., score) in the interval [0,1] (see Figure 2). Specifically, a preference indicates whether the score is monotonically increasing or decreasing with respect to the plan variable within certain bounds[1]. The user can also specify that the score increases as the difference with a given fixed value decreases. In other words, the high
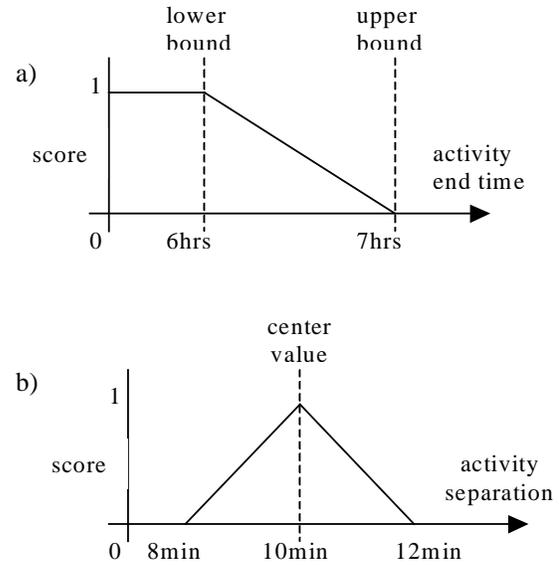

Figure 2: a) Mapping the end time of an activity to a score. This implements a preference for minimizing tardiness of an activity. The deadline is at the sixth hour and the score decreases to zero one hour after the deadline. b) Mapping the distance between two activities to a score centered on a given value. This implements a preference for maintaining a 10 minute separation with a ±2 minute tolerance.

score is centered on a value selected from the domain of the variable. From this high-level specification, mapping functions are generated that take preference variables as arguments and return real-valued scores.

Each preference includes an upper and lower bound to indicate the range of the variable for which the score increases or decreases. Any values outside this range produce a score of either zero or one. For example, anything over 90% battery charge may be indistinguishable in terms of quality. Therefore, a preference can be defined as increasing with minimum charge and reaching a maximum score at 90% charge. Each preference also includes a weight for specifying the relative importance of the preference to overall plan quality. The score of a plan is computed as the weighted average of scores for plan variables with preferences.

Aggregate Preferences

An aggregate preference is defined for many plan variables, and can either score each variable independently, or score the result of applying a function to the variables. If the preference scores each variable, then the scores are weighted equally and averaged. The built-in functions that can be used in aggregate preferences include average[2], sum, minimum, and

---

[1] The mappings are internally defined as simple linear functions in order to guarantee monotonicity.

[2] The score of the average can be quite different from the average score of a set of variables. In particular, this difference is clear when the preference is centered on a value.

maximum. These functions constitute the set of functions most commonly observed in preferences from various domains. For example, minimizing makespan is a preference on the *maximum* end time of all activities in the schedule. The specified function is computed for the current set of plan variables, and the result is mapped to a score for the preference.

Improving Plan Quality

Preferences allow us to define quality metrics for evaluating feasible plans and making quantitative distinctions between different plans. The next step is to use these preferences to produce high quality plans. Preferences can be used as heuristics when generating a feasible plan or to directly improve the quality of an existing plan. We interleave repair-based planning with preference-driven, incremental optimization.

Local Improvement Experts

In addition to establishing quality metrics, preferences can provide insight into how to improve plan quality. We define domain-independent *improvement experts* to aid in optimization (see Figure 3). Improvement experts are based solely on the class of preference (and variable) for which it is constructed. An instance of an expert uses the preference specification to calculate plan modifications that will improve the score for the given preference and current plan. In other words, an expert is a link between changes in the plan and the change in quality. For example, if less resource usage were preferred, expert improvements would include deleting an activity that is currently using the resource. It is a local expert, however, and does not guarantee an increase in overall plan quality. Improvement experts provide a framework for optimization algorithms, defining the search space of possible improvements. We define a separate class of improvement expert for each class of preference.

Local activity variable expert. One class of expert is used for improving preferences on local activity variables. The most obvious modification for improving this preference is to change the value of the local variable. The expert only considers variables that are currently contributing to the low score. For example, only the end time of activity a2 in Figure 3 can be changed to improve the score for this preference. If score is a decreasing function of the variable, then making an improvement requires assigning a value less than its current value. Likewise, we must assign a value greater than its current value to improve an increasing preference. In cases where the variable is the start or end time of the activity, assigning a value implies moving the activity to earlier or later times. Expert modifications also include creating activities with high scoring values or deleting activities with low scoring values for the specified variable.

Activity/goal count expert. A different class of improvement expert is used for preferences on the number
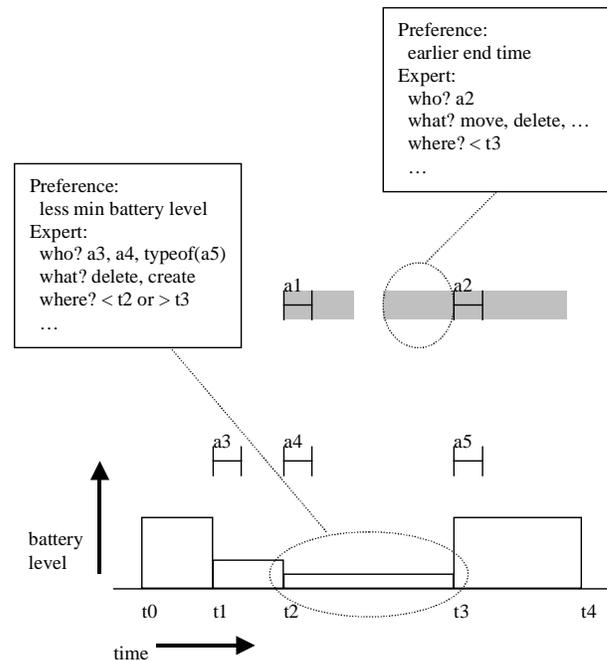

Figure 3: Local improvement experts.

of activities/goals. For a given preference of this class, there is only one expert modification. When the preference is for more occurrences of a goal/activity, creating new activities is the only beneficial modification. When the preference is for fewer occurrences, deleting existing activities is the only improvement.

Resource/state variable expert. Another class of expert improves preference scores for the values of resources or state variables. Only activities that use the resource or state variable are considered. For a high resource preference, the expert selects activities that increase the resource when adding and activities that decrease the resource when deleting. Just the opposite is true for low resource preferences. When moving, if the preference is for a higher minimum resource value, activities that decrease the resource during this time can be moved away from the minimum value. In Figure 3, activities a3 and a4 both contribute to the low minimum battery level. If the preference is for a lower maximum resource value, activities that increase the resource during this time can be moved away from the maximum value. Similar (but probably less useful) cases exist for higher maximum and lower minimum resource values. Moving an activity does not significantly change the average resource value and therefore is not considered for preferences on averages.

Resource/state change count expert. A simpler class of expert is used for improving scores of preferences on the number of times a resource or state variable changes over time. Adding activities that use the resource or state variable will increase the number of changes. Deleting will decreases the number of changes. Because each activity makes a constant number of changes on a

resource/state variable, moving has no impact on the change count.

State duration expert. The last class of improvement expert works on state duration preferences. Activities that change the state variable can be created, deleted, or moved in order to change the amount of time planned in a particular state. When the preference is for a longer duration, activities that change to the specified state can be created at times when the variable is in a different state. Conversely, when the preference is for a shorter duration, activities that change to any other state can be created at times when the variable is in the specified state. For example, an activity that switches an instrument off can shorten times where the instrument has been left on. Similar reasoning is used when deleting or moving activities.

Monotonic Preference Assumption

In order to make improvement calculations tractable, we make a *monotonic preference assumption*, requiring each mapping from plan variable to quality metric to either be consistently increasing or decreasing within a given range of the variable. For preferences centered on a value, the score must increase for values less than the specified center value, and decrease for values greater than the center value. In this way, the problem can be restated as simply identifying modifications that increase or decrease the current values of plan variables participating in preferences. For example, if a variable with integer domain [1,10] and current value 4 has a decreasing preference, then only values in the range [1,3] will increase the score for this preference.

Iterative Optimization

The full set of potential plan improvements can be quite large. Once the expert has calculated this set, we search for more optimal plans by iteratively selecting and making improvements (see Figure 4). We call this technique *iterative optimization* because of its similarity to iterative repair techniques used for repairing plan conflicts (i.e., constraint violations). More specifically, the iterative optimization algorithm first selects a preference from the list of low-scoring preferences. Typical heuristics for this decision include selecting a preference with one of the lowest scores or one with the most potential gain (weight * (1 − score)). Next, the algorithm must decide which type of modification to perform for the selected preference.

We allow several types of plan modifications in ASPEN. New activities can be instantiated from types defined in the domain, scheduled activities can be moved to different time or simply deleted, and local variables in activities can be changed. For each type of modification, there are additional decisions that must be made before applying the modification. When creating a new activity, the activity type must be selected and values must be assigned to variables in the new activity instance. When

```
Iterative Optimize (T)
   Let P = P_best  = current plan
   Let S = S_best  = score(P)
   While (S <1 and time < T)
      If conflicts exist, Then repair(T-time)
      Else
         Let Q = set of preferences with score < 1
         q = choose(Q)
         M = E_q(P) //get the set of modifications
         m = choose(M)
         P = m(P) // apply the chosen modification
         S = score(P)
         If (S > S_best) // save if best-so-far
            S_best = S
            P_best = P
   Return P_best
```

Figure 4: The ASPEN optimization algorithm. $E_q(P)$ returns the set of modifications for plan P calculated by the expert E for improving preference q.

moving, an existing activity and a new start time must be selected such that the resulting preference score is increased.

As an example, consider a preference for a high minimum resource availability. The preference expert would find activity types that provide the resource and existing activity instances that consume the resource at the time of minimum availability. The expert would also suggest adding the provider at the time of minimum availability or moving the consumer to any other time. In short, the improvement experts provide information as to which alternatives for each decision are useful for optimization.

After making a local improvement, the resulting plan may not be optimal. The iterative optimization algorithm continues by selecting another preference, and repeating the improvement process. After each improvement, the resulting overall score is compared with the best score achieved so far. If the current score exceeds the best score, the current plan is saved. The algorithm halts when the maximum score is attained, or when a specified time limit is reached. If an optimal plan was not found, the saved plan with the best score is returned.

Maintaining Feasibility

When making modifications during iterative optimization, a few simple, domain-independent heuristics are used to avoid violating hard constraints. However, some improvements may require creating new conflicts. Adhering to the plan constraints may be too restrictive, precluding modifications necessary for improving quality. Therefore, the iterative optimization algorithm may create infeasible intermediate plans while searching for an optimal plan. However, because it is unknown how the plan will change to achieve feasibility, we do not attempt to define quality for inconsistent plans. Plans with violations are assigned the minimum possible score and

the iterative repair algorithm is invoked to restore feasibility before continuing with optimization.

Competing Objectives

The iterative optimization algorithm does not perform strict hill-climbing. Since modifications are applied to increase the score for a single preference, scores for other preferences may have suffered and the overall score for the plan may have decreased after a single iteration. This suggests that a subset of the preferences represent competing objectives. Although we focus on a single preference at each step in optimization, we do not necessarily maximize the preference score. We only attempt to increase the score by stochastically choosing one of the potential improvements. Therefore, we would expect competing preferences with a large disparity to eventually reach a compromise rather than thrash between a high score for one and a low score for the other.

Continuous Improvements

During execution we may notice differences between actual and expected values for activities or resources. These differences may violate hard constraints or degrade plan quality. The CASPER system (Chien et al. 1999) was developed to continuously initiate and monitor the execution of an ASPEN plan, updating the plan when necessary. As the result of a plan update, CASPER uses the iterative algorithms to fix new conflicts and improve preference scores. In this way, CASPER provides continuous planning and optimization during the course of execution.

Experimental Results

We now describe our initial experiments with incremental optimization on several NASA domains. An ASPEN planning model was developed for each domain, including user preferences on various plan features as well as the typical activity definitions and hard constraints. Each model required less than ten lines of text describing the preferences in the ASPEN preference language. Then, iterative optimization and repair were run on randomly generated problem instances of three different levels of difficulty. The problem difficulty has many factors including the number of goals, the complexity of the goals, and the length of goal opportunity windows. Initially, there are no satisfied goals, and the algorithm continues to do one of three things: 1) satisfy a goal by adding the requested activity to the plan, 2) improve the score for any of the other preferences, or 3) repair conflicts created by 1 or 2.

Each problem was run on a Sun Sparc Ultra-60. After five minutes of planning and optimization, the saved plan with the maximum overall score is reloaded and the relevant data is recorded. Results from each problem size were averaged over 100 runs and are shown in the left-hand column under each problem size in the tables.

Approximate "optimal" values were manually estimated for comparison and are shown in the right-hand column under each problem size in the tables[1]. It is important to note that the "optimal" values were estimated for each preference in isolation. In other words, each represents the best value that we can hope for even if all other preferences were ignored. Given this, most values approach the "optimal" value within reason.

New Millennium ST-4

ST-4 is a proposed spacecraft designed to land on a comet, mine core samples of the surface, and return a sample to earth. The model has 6 shared resources, 6 state variables, and 22 activity types. Resources and states include a battery, bus power, communications, drill location, drill state, 2 oven states for the primary and backup ovens, camera state, and RAM. There are two activity groups that correspond to different types of experiments: mining and analyzing a sample, and taking a picture. There is a downlink activity type that replenishes the RAM buffer by transmitting data to earth. Each ST-4 problem instance includes fixed profile that represents the comet-landed phase of the mission with randomly generated oven failures. Each problem also includes requests for mining and picture experiments at random start times.

The ST-4 model includes preferences for: more science goals, using the primary oven, higher minimum battery level, fewer downlinks, later downlinks (so they might transmit more data), fewer drill operations, and fewer oven operations. Interactions come from science goals lowering the battery level and requiring a downlink when the RAM buffer is full.

Table 1 gives the results for the average values for each of the preference variables in the best ST-4 plan. The "goal count" refers to the average number of satisfied goals versus the average number of requested goals. The second row gives the percent of goals that use the backup oven. The backup oven is required when the primary oven fails or is oversubscribed. While the numbers show that plans that use the primary oven are preferred, it is not clear what minimum percent of goals must use the backup oven. Therefore, these cells contain a dash (—) in the table. The average number of downlinks planned is shown in the third row while the average downlink amount is in the fourth row. The downlink buffer holds 30MB and each experiment uses 5MB, forcing a downlink after 6 experiments. In the best case, each downlink would replenish all 30MB. The fifth row gives the number of operations planned for the primary oven. Half of the experiments require an oven, each of which may require 3 oven operations (preheat, heat, cool). Therefore, the "optimal" value is calculated by multiplying 3 times the average number of goals that use the primary oven (any more would be unnecessary operations).

---

[1] The complexity of the problems makes it difficult to define true optimal plan values.
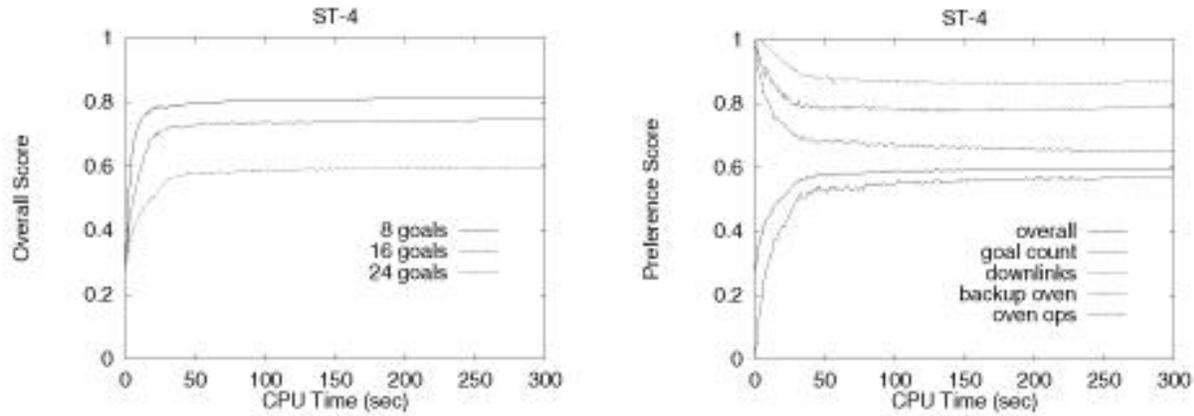
Figure 5: a) The change in the score of the "best-so-far" plan for 8, 16 and 24 requested ST-4 goals. The final score is lower for more difficult problems because a smaller percentage of the requested goals are satisfied. Also, the score increases more slowly for the more difficult problems. b) The change in the individual preference scores for the 24 goal ST-4 problem. The preference for satisfying goals is weighted higher, allowing optimization to somewhat lower scores for other preferences in order to increase the goal preference. The scores quickly reach a stable state where the utility of adding a goal does not justify the decrease in other quality metrics that is necessary to support the new goal.

Figure 5 shows the performance of optimization in the ST-4 domain. Only the change in score is important, as the absolute score values are somewhat arbitrary. The graphs for the other domains are similar and therefore omitted.

Problem Size	8 goals		16 goals		24 goals	
goal count	6.62	8	12.6	16	14.2	24
backup oven	0.05	—	0.13	—	0.22	—
downlinks	0.62	0.39	2.13	1.40	3.36	1.69
downlink amt	15.6	30	19.1	30	16.4	30
oven ops	10.3	9.43	17.5	16.4	17.1	16.6

Table 1: ST-4 results.

New Millennium EO-1

New Millennium Earth Observer 1 (EO-1) is an earth imaging satellite featuring an advanced multi-spectral imaging device. EO-1 mission operations consists of managing spacecraft operability constraints (power, thermal, pointing, buffers, consumables, telecomm, etc.) and science goals (imaging surface targets within specific observation parameters). One of the interesting constraints involves the Solar Array Drive (SAD) which keeps the solar arrays facing the sun. For a few minutes before and during each data-take, the SAD must be locked to avoid spacecraft jitter, which can corrupt data. The EO-1 model consists of 14 resources, 10 state variables and total of 38 different activity types. Each EO-1 problem instance includes a randomly generated, fixed profile that represents typical sun and cloud patterns. Each problem also includes randomly placed science requests for observations and calibrations. The size of the problem varies from 2 to 6 days, and for each day, four additional observation and calibration goals are added.

The EO-1 model includes preferences for: more science goals, more time with the SAD tracking the sun, fewer changes of the SAD state, and less deviation from the preferred separation of data-take and SAD locking activities. The last preference has a high score centered on a value because if the settling time is too small there will be too much jitter, but if the separation is too large the solar array power output will suffer.

Table 2 gives the results for the average values for each of the preference variables in the best EO-1 plan. The second row gives the number of hours planned for the SAD in the "tracking" state while the third row gives the number of SAD operations in the plan. In the best case, the SAD would simply track the sun 24 hours a day. However, the observations require a small amount of time with the SAD locked, which requires one operation to lock the SAD and one to return it to tracking mode. The last row contains the average number of minutes between each SAD lock activity and the subsequent data-take. The desired separation is five minutes.

Problem Size	1 day		2 days		3 days	
goal count	5.56	6	11.8	12	11.7	18
SAD tracking	23.8	24	47.8	48	71.7	72
SAD ops	4.23	4	8.27	8	8.15	8
SAD/data-take separation	5.00	5	4.76	5	4.87	5

Table 2: EO-1 results.

Data-Chaser

Data-Chaser is a shuttle science payload that flew onboard Space Shuttle Flight STS-85 in August 1997. The model consists of 19 shared resources, 24 state variables, and 72 activity types. Resources and states include shuttle orientation, contamination state, 3 scientific instruments (including apertures, relays, heaters, etc.), several RAM buffers, tape storage, power (for all instruments/devices), and downlink availability. Each Data-Chaser problem

instance includes a randomly generated, fixed profile that represents shuttle orientation and contamination states. We generate plans for 1 to 3 days of mission operations. The number of randomly generated science requests is based on the fixed profile and the number of days for the given problem instance.

The Data-Chaser model includes preferences for: more science goals (i.e., experiment data-takes), earlier data-take start times, fewer relay changes, and less duration of the relay "on" state. These preferences are interesting for several reasons. First, data-takes cannot overlap and require the relay to be in the "on" state. Therefore, more data-takes will require a longer "on" state duration. Second, shortening the duration of the "on" state might require more changes of the state of the relay.

Table 3 gives the results for the average values for each of the preference variables in the Data-Chaser plan with the maximum overall score. The second row contains the average number of changes of the relay state variable in the plan. Experiments in the plan tend to be grouped during windows of opportunity (approximately 1 per day). Therefore, the approximate "optimal" number of relay switches is twice the number of days (one turning the relay on and one turning it off). The third row gives the average amount of time (in hours) planned for the relay to remain in the "on" state. The relay should stay on during each window of opportunity and each window is approximately 30 minutes long. Therefore, the "optimal" duration is about 0.5 hours for every day of operations. The values for goal and relay counts are somewhat reasonable. It is unclear why resulting plans were so poor with respect to the "on" duration preference (the relay was left in the "on" state much longer than necessary).

Problem Size	1 day		2 days		3 days	
goal count	5.88	6.66	9.14	13.0	13.1	23.1
relay count	3.26	2	7.21	4	8.98	6
"on" duration	18.5	0.5	12.3	1	16.3	1.5

Table 3: Data-Chaser results.

Rocky-7 Mars Rover

Rocky-7 is a prototype Mars rover for long-range planetary science gathering. The model consists of 18 shared resources, 12 state variables and 32 activity types. Resources and states include 3 digital cameras (at the front, rear, and on a mast), a deployable mast, a shovel, a spectrometer, solar arrays, batteries, and a RAM buffer. There are five activity groups that correspond to different types of science experiments: one for collecting spectrometer readings, and four for taking images at a location (a front image, a rear image, a panorama using the mast, and a closeup image using the mast). Rover problems are sized by the number of hours of daylight (all operations require illuminated solar arrays). A series of science goals are generated corresponding to the number of hours of daylight, and the parameters for the goals are randomly generated (such as target locations). For each additional hour of daylight, ten additional goals are added

spread over two new locations. Repair heuristics include traveling salesman heuristics, which attempt to order the rover traversals such that the total distance traveled is minimized.

The Rocky-7 model includes preferences for: more science goals, less traversing, less time with the mast deployed, fewer mast deploy and stow operations, and a higher minimum battery level. This set also contains interacting preferences. For example, adding science activities will drain the battery and may require mast operations and a rover traversal.

Table 4 gives the results for the average values for each of the preference variables in the best Rocky-7 plan. Again, the first row contains the number of satisfied goals versus the number of requested goals. While the number of satisfied goals does not significantly increase when more goals are requested, the goals are spread across more locations making the problem more difficult. The second row contains the average number of traverses made by the rover and the third row gives the average number of minutes spent traversing. Each hour contains goals at two different locations, which requires two rover traverses. The rover is traveling one meter per minute over a 20 square meter area. The minimum required traversal time, however, is difficult to estimate and has been omitted. The fourth row gives the number of minutes planned for the mast in the "deployed" state while the sixth row gives the number of mast operations in the plan. The "optimal" number of mast operations is estimated as two per location, one to deploy it for the goals and one to stow it before the next traverse. At a minimum, the mast would need to be deployed for six minutes at each location. The last row gives the minimum battery level for the plan. The maximum minimum level is estimated assuming the battery is recharged after every operation (which is quite optimistic considering the time required to recharge).

Problem Size	1 hour		1.5 hours		2 hours	
goal count	8.03	10	8.91	15	8.31	20
traverses	2.77	2	5.21	3	8.31	4
traversing	24.5	—	42.8	—	62.8	—
deployed	41.8	12	53.6	18	67.8	24
mast ops	5.78	4	12.3	6	22.8	8
battery level	20.9	51	20.5	51	22.0	51

Table 4: Rocky-7 results.

Related Work

Much of the recent work in plan optimization has been looking at ways to integrate linear programming (LP) techniques with symbolic AI and constraint propagation (Baptiste, Le Pape, and Nuijten 1995; Hooker et al. 1999; Kautz and Walser 1999; Vossen et al. 1999). Typically, LP equations are used to represent numeric constraints and objectives in conjunction with STRIPS-style planning operators. While LP formulations have the advantage of taking a global view of plan quality, they can be difficult

to develop and computationally expensive to solve when including representations for state, resource, and temporal constraints. Moreover, there is no sense of incremental improvements. Slight changes in the problem specification require a restart of optimization and may result in a drastically different solution.

PYRRHUS (Williamson and Hanks 1994) is a partial-order planner that has been extended to handle metric resources, time and a utility model. Utility is defined as achieving goals early and using a minimal amount of resources. While we do not attempt to define utility on incomplete plans, partial-order planners *must* evaluate the utility of partial-plans in order to address optimization. The PYRRHUS algorithm uses a branch-and-bound search, discarding partial plans whose upper bound on utility is less than the utility of the current best complete plan. In order to compute the upper bound on utility of partial plans, they require the representation to have an overall utility function that is monotonically non-increasing as refinements are made. This is significantly more restrictive than our assumption, which only requires that the individual components of utility be monotonic. They define utility as a decreasing function of missed deadlines and resource consumption. This is a specialization of our utility model and is less applicable in some domains. Spacecraft commanding problems, for example, typically are more concerned with packing science into limited time windows, rather than meeting deadlines.

(Myers and Lee 1999) view the optimization problem as providing a *set* of qualitatively different plans, which can then be further refined by human planners. Plans are generated with the SIPE-2 planner modified to produce a good sampling along the various dimensions of quality. This appears to be very useful for domains with a high degree of interaction between quality metrics, or with little understanding of the overall quality. Preferences and their weights could be identified using a mixed-initiative approach. Once quality is well-defined, however, it becomes more desirable to find a single, high-quality plan.

The CABINS (Miyashita and Sycara 1995) system uses a similar iterative optimization algorithm to improve complete but sub-optimal schedules. Here, case-based reasoning (CBR) is used to learn control rules for optimization problems. While CABINS focuses on an individual activity when attempting modifications for improvement, we focus on an individual quality metric. But the main difference from our approach is that preferences are stated for scheduling decisions rather than for values of variables. In addition, preferences are learned in a case acquisition phase, where the user implicitly makes preferences by evaluating the problem solving results. While the user is not required to supply the set of objectives, the user must provide an explanation of their evaluation, which includes weighting the impact of the modification on each objective.

Our approach is a specialization of black-box optimization techniques. Black-box algorithms proceed with no knowledge of the quality function and iteratively sample the quality surface using various search methods. While these techniques are generic and can optimize arbitrary quality functions, the large search space makes both finding and applying the appropriate technique prohibitively expensive. In contrast, we can pinpoint moves that have potential for improvement, and vastly prune the search space by simply assuming monotonicity along individual dimensions of quality. While this somewhat limits the expressiveness of quality, we do not believe it to be a burden in many planning domains.

Discussion and Future Work

In general, our technique is more likely to perform well on problems with local, non-interacting preferences. On each iteration, our optimization algorithm reasons with only a single preference (and a single parameter in an aggregate preference). Improving preferences that require simultaneous evaluation will be more difficult (i.e., require more search). For example, during makespan minimization, moving one activity earlier in the sequence may require another activity to finish later and increase makespan. Although our preferences are general enough to represent global problems such as makespan minimization, iterative optimization is not likely to perform as well as algorithms designed specifically for this task.

In future work, we would like to compare the performance of our approach to constraint relaxation techniques. One could start over-constrained to the preferred values and then relax the constraints, or start under-constrained and then incrementally optimize toward the preferred values. Initially, constraint relaxation may be closer to optimal plans, but further from feasible ones. Incremental optimization quickly finds a feasible plan, but one that may be far from optimal. At this point it is unclear which would perform better. We would also like to make use of the preference experts in plan construction. While constructing or repairing a plan, preference experts might be able to give heuristic information about which decisions are more likely to result in high quality plans. Although the overall quality of an incomplete plan is not well-defined, the relative quality of some decisions can be computed.

Finally, while this paper focuses on iterative optimization, much of our effort was in developing a representation of plan quality and a framework for optimization. This will facilitate the implementation of more sophisticated search algorithms in future work. For example, the "Squeaky Wheel" optimization algorithm (Joslin and Clements 1999) could be implemented in our framework. In this case, we might consider low-scoring preferences as the "squeaky wheels" and move them closer to the front of the queue of preferences waiting to be optimized.

Conclusions

We have described an approach for representing and optimizing user quality metrics (i.e., soft constraints) using generic preferences for values of variables that occur in a plan. In our approach, the set of local improvements can be efficiently computed for each preference in a domain-independent fashion. To accomplish this, the representation is restricted to monotonic functions for mapping plan values to quality metrics. We have demonstrated the feasibility of our approach by implementing and empirically evaluating a local search algorithm for iterative optimization on four different domains.

Acknowledgements

References

Baptiste, P.; Le Pape, C.; and Nuijten, W. 1995. Incorporating Efficient Operations Research Algorithms in Constraint-Based Scheduling, In *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999. Integrated Planning and Execution for Autonomous Spacecraft, In *Proceedings of the 1999 IEEE Aerospace Conference*.

Chien, S.; Smith, B.; Rabideau, G.; Muscettola, N.; and Rajan, K. 1998. Automated Planning and Scheduling for Goal-Based Autonomous Spacecraft, *IEEE Intelligent Systems*, September/October, 50-55.

Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D.; 1997. Toward an Application Framework for Automated Planning and Scheduling, In *Proceedings of the 1997 International Symposium of Artificial Intelligence, Robotics and Automation for Space* (iSAIRAS-97), Tokyo, Japan.

Hooker, J. N.; Ottosson, G.; Thorsteinsson, E. S.; and Kim, H. 1999. On Integrating Constraint Propagation and Linear Programming for Combinatorial Optimization, In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (AAAI-99), 136-142. AAAI Press.

Joslin, D.; Clements, D. 1999. "Squeaky Wheel" Optimization, *Journal of Artificial Intelligence Research* 10:353-373.

Kautz, H.; and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search, In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI-96), 1194-1201. AAAI Press.

Kautz, H.; and Walser, J. 1999. State-space Planning by Integer Optimization, In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (AAAI-99), 526-533.

Mazure, B.; Sais, L.; and Gregoire, E. 1997. Tabu Search for SAT, In *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (AAAI-97), 281-285. AAAI Press.

Minton, S.; and Johnston, M. 1988. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence* 58:161-205.

Miyashita, K.; and Sycara, K. 1995. CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair, *Artificial Intelligence*, 76(1-2):377-426.

Myers, K. L.; and Lee, T. J. 1999. Generating Qualitatively Difference Plans through Metatheoretic Biases, In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (AAAI-99), 570-576. AAAI Press.

Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations Using the ASPEN System, In *Proceedings of the 1999 International Symposium of Artificial Intelligence, Robotics and Automation for Space* (iSAIRAS-99).

Smith, B.; Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Representing Spacecraft Mission Planning Knowledge in ASPEN, *Artificial Intelligence Planning Systems Workshop on Knowledge Acquisition*, Pittsburgh, PA.

Sycara, K.; Dajun Zeng; and Miyashita, K. 1995. Using Case-Based Reasoning to Acquire User Scheduling Preferences that Change Over Time, In *Proceeding of the Eleventh Conference on Artificial Intelligence for Applications*, 240 –246.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. 1999. On the Use of Integer Programming Models in AI Planning, In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (IJCAI-99).

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann.

Williamson, M.; and Hanks, S. 1994. Optimal Planning with a Goal-Directed Utility Model, In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems* (AIPS-94), 176-181. AAAI Press.

Williamson, M; and Hanks, S. 1996. Flaw Selection Strategies for Value-Directed Planning, In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems* (AIPS-96), 237-244. AAAI Press.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. Scheduling and Rescheduling with Iterative Repair, In *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, CA. 241-256.