ITERATIVE REPAIR PLANNING FOR SPACECRAFT OPERATIONS USING THE ASPEN SYSTEM

Gregg Rabideau, Russell Knight, Steve Chien, Alex Fukunaga, Anita Govindjee

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 126-347, Pasadena, CA 91109-8099
phone: +1 818 393-5364, fax: +1 818 393-5244, email: {firstname.lastname}@jpl.nasa.gov

ABSTRACT

This paper describes the Automated Scheduling and Planning Environment (ASPEN). ASPEN encodes complex spacecraft knowledge of operability constraints, flight rules, spacecraft hardware, science experiments and operations procedures to allow for automated generation of low level spacecraft sequences. Using a technique called *iterative repair*, ASPEN classifies constraint violations (i.e., *conflicts*) and attempts to repair each by performing a planning or scheduling operation. It must reason about which conflict to resolve first and what repair method to try for the given conflict. ASPEN is currently being utilized in the development of automated planner/scheduler systems for several spacecraft, including the UFO-1 naval communications satellite and the Citizen Explorer (CX1) satellite, as well as for planetary rover operations and antenna ground systems automation. This paper focuses on the algorithm and search strategies employed by ASPEN to resolve spacecraft operations constraints, as well as the data structures for representing these constraints.

1. INTRODUCTION

Planning and scheduling technology offers considerable promise in automating spacecraft operations. Planning and scheduling spacecraft operations involves generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals (see [Chien et al., 1998b] for an overview). We discuss ASPEN and its use of an *iterative repair* algorithm for planning and scheduling as well as for replanning and rescheduling.

ASPEN is a reconfigurable planning and scheduling software framework [Fukunaga et al., 1997]. Spacecraft knowledge is encoded in ASPEN under seven core model classes: activities, parameters, parameter dependencies, temporal constraints, reservations, resources and state variables. An activity is an occurrence over a time interval that in some way affects the spacecraft. It can represent anything from a high-level goal or request to a low-level event or command. Activities are the central structures in ASPEN, and also the most complicated. A more detailed definition is given in a later section. Together, these constructs can be used to define spacecraft components, procedures, rules and constraints in order to allow manual or automatic generation of valid sequences of activities, also called *plans* or *schedules*.

Once the types of activities are defined, specific instances can be created from the types. Multiple activity instances created from the same type might have different parameter values, including the start time. Many camera imaging activities, for example, can be created from the same type but with different image targets and at different start times. The sequence of activity instances is what defines the plan.

The job of a planner/scheduler, whether manual or automated, is to accept high-level goals and generate a set of low-level activities that satisfy the goals and do not violate any of the spacecraft flight rules or constraints. ASPEN provides a Graphical User Interface (GUI) for manual generation and/or manipulation of activity sequences. However, the automated planner/scheduler will be the focus of the remainder of this paper.

In ASPEN, the main algorithm for automated planning and scheduling is based on a technique called *iterative repair* [Zweben et al., 1994]. During iterative 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 reservation, parameter dependency or temporal constraint. Conflicts can be repaired by means of several predefined methods. The repair methods are: moving an activity, adding a new instance of an activity, deleting an activity, detailing an activity, abstracting an activity, making a reservation of an activity, canceling a reservation, connecting a temporal constraint, disconnecting a constraint, and changing a parameter value. The repair algorithm may use any of these methods in an attempt to resolve a conflict. How the algorithm works is largely dependent on the type of conflict being resolved.

Specifically, we haven taken a most-committed, local, heuristic, iterative repair approach. This approach has many desirable properties for spacecraft operations planning. First, using a repair algorithm allows automated planning to be utilized at any time and on any given initial plan. The initial plan may be as incomplete as a set of goals, or it may be a previously produced plan with only a few flaws. Repairing an existing plan enables fast replanning when necessary from manual plan modifications or from unexpected differences detected during execution. Second, heuristics allow the search to be pruned, ruling out less promising planning choices. In addition, heuristics may also suggest particular choices that may lead to a solution in less time, or to a higher quality solution. Third, a local iterative algorithm does not incur the overhead of maintaining intermediate plans or past repair attempts. This allows the planner to quickly try many plan modifications for repairing the conflicts. However, unlike systematic search algorithms, it cannot be guaranteed that our iterative repair algorithm will explore all possible combinations of plan modifications or that it will not retry unhelpful modifications. In our experience, these guarantees are not worth the required overhead. Finally, by committing to values for parameters, such as activity start times and resource usages, the effects of a resource usage and the corresponding resource profiles can be efficiently computed. Least-commitment techniques retain plan flexibility, but can be computationally expensive for large applications. Further discussions on this topic can be found in [Chien et al., 1998b].

The full paper will describe the ASPEN search structure in greater detail. We will describe the constraints that can be modeled in ASPEN, as well as the conflicts for each type of constraint violation. We will also describe how the search can be influenced using heuristics. Finally, we will discuss current, future and related work.

2. MODEL COMPONENTS AND CONSTRAINTS

Spacecraft models are developed in the ASPEN Modeling Language (AML) [B. Smith et al., 1998; Sherwood et al., 1998]. These models are parsed into data structures that provide efficient reasoning capabilities for planning and scheduling. There are seven basic components to an ASPEN model: activities, parameters, parameter dependencies, temporal constraints, resources, state variables, and reservations. Together, they describe what the spacecraft can and cannot do during operations.

A *parameter* is simply a variable with a restricted domain. One parameter, for example, can be the range of integers between ten and twenty. Other parameter types include floating point numbers, booleans and strings. A *parameter dependency* is a functional relationship between two parameters. An activity end time, for example, is a function (the sum) of the start time and the duration. A more complicated dependency might compute the duration of a spacecraft slew from the initial and final orientation.

In the model, relative ordering constraints can be specified for pairs of activities. A *temporal constraint* is a relationship between the start or end time of one activity with the start or end time of another activity (see Figure 1). One might specify, for example, that an instrument warming activity must end before the start of an activity that uses the instrument. Minimum and maximum separation distances can be specified in a temporal constraint. The warming activity for example, might be required to end at least one second but at most five minutes before using the instrument. Temporal constraints can be combined with conjunctive or disjunctive operators to form more complicated expressions.

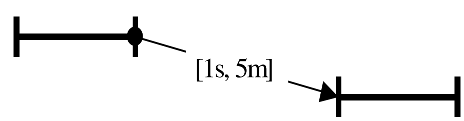

Figure 1: A temporal constraint with a required separation of at least 1 second and at most 5 minutes.

A *resource* represents the profile of a physical resource or system variable over time (see Figure 2), as well as the upper and lower bounds of the profile. In ASPEN, a resource can either be depletable or non-depletable. A depletable resource is used by a reservation and remains

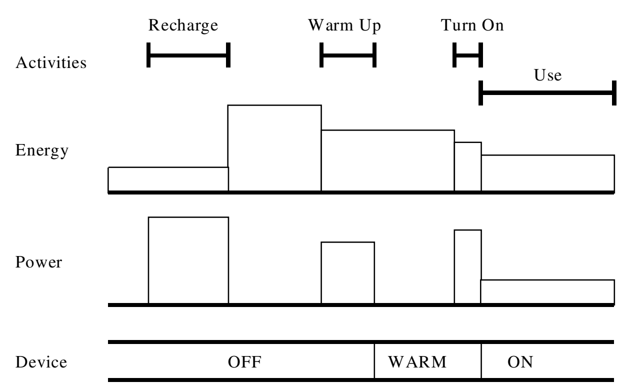

Figure 2: Timelines for activities, a depletable resource (energy), a non-depletable resource (power), and a state variable (device).

used even after the end of the activity making the reservation. Examples of depletable resources on spacecraft include memory, fuel and energy. A non-depletable resource is used only for the duration of the activity making the reservation. Power is an example of a non-depletable resource. A resources can be assigned a capacity, restricting its value at any given time. A *state variable* represents the value of a discrete system variable over time. The set of possible states and the set of allowable transitions between states are both defined with the state variable. An example of a state variable is an instrument switch that may be either ON, WARMING, or OFF. This state variable may be restricted to transitions from OFF to WARMING and not directly to ON. *Reservations* are requirements of activities on resources or state variables. For example, an activity can have a reservation for ten watts of power. Some reservations are modeled as instantaneous effects (e.g., reservations that change the state on a state variable). The user can specify whether this effect occurs at the start or end of the activity.

Activity hierarchies can be specified in the model using decompositions (see Figure 3). A decomposition is a set of sub-activities along with temporal constraints between them. In this way, one can define a high-level activity that decomposes into a set of lower-level activities that may be required to occur in some relative order. These activities in turn may have their own decompositions. In addition, an activity may have multiple decompositions to choose from. Thus, allowing an activity to be expanded in different ways.

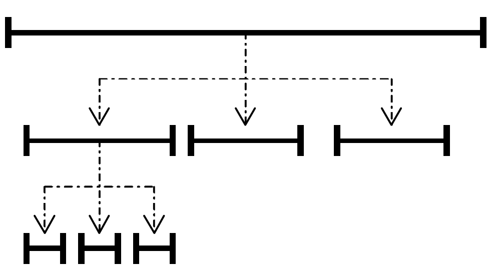

Figure 3: An activity hierarchy

An *activity* has a set of parameters, parameter dependencies, temporal constraints, reservations and decompositions. All activities have at least three parameters: a start time, an end time and a duration. There is also at least one parameter dependency, relating these three parameters. In addition, all activities have at least one temporal constraint that prevents the activity from occurring outside of the planning horizon. Any additional components are optional.

3. CONFLICTS

A complete plan may not always be consistent with the constraints in the model. A conflict is a violation of one of the model constraints. We define ten basic types of conflicts in ASPEN:

- Abstract activity conflicts
- Unassigned parameter conflicts
- Violated parameter dependency conflicts
- Unassigned temporal constraint conflicts
- Violated temporal constraint conflicts
- Unassigned reservation conflicts
- Depletable resource conflicts
- Non-depletable resource conflicts
- State requirement conflicts
- State transition conflicts.

Each conflict provides information about what objects are involved and how to repair the conflict.

An *abstract activity conflict* is simply an activity that has not yet been decomposed into its sub-activities. All activities must be expanded to their most detailed level. If an activity has more than one decomposition, the planning algorithm must decide which decomposition to use when detailing the activity. Detailing an activity involves creating instances of the activities specified in the decomposition. In addition, all temporal constraints and parameter dependencies must be connected among the new sub-activities and the parent activity.

An *unassigned parameter conflict* is a parameter that does not have a unit value. A parameter can be a range or a set of values. However, this is a conflict in the plan until a value, chosen from the range or set, has been assigned to the parameter.

A *violated parameter dependency conflict* is a violation of a functional relationship between two parameters. In other words, the value of a parameter is not equal to the result of a function that constrains that parameter value. For example, a parameter `p` may be required to be the square of another parameter `q`. If `q` is assigned to 5 and `p` is assigned any value other than 25, this will be a parameter dependency conflict. This conflict can be resolved by assigning a different value to either `p` or `q`.

An *unassigned temporal constraint conflict* occurs when a temporal constraint exists for an activity, but an activity instance has not been selected to satisfy the constraint (see Figure 4). A temporal constraint is defined in one activity type `A` and specifies the requirement for another activity `B` within some temporal relationship. When an instance of `A` is created, the temporal constraint is created and is not initially assigned an instance of `B`. The conflict

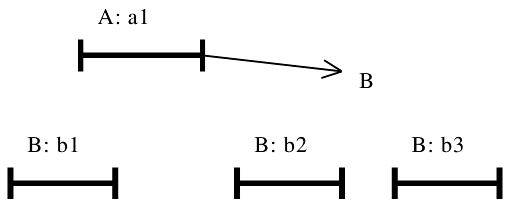

Figure 4: An unassigned temporal constraint conflict requiring an activity of type B. Any of b1, b2 or b3 can be use, or a new instance of type B can be added.

computes all activity instances that can repair this conflict (basically, all instances of type `B`).

A *violated temporal constraint conflict* occurs when a temporal constraint has been assigned, but the relationship (specified in the model) does not hold for the two participating activities (see Figure 5). For example, consider an activity instance `A` that must end before the start of activity instance `B` by at least 10 seconds but at most 1 minute. If `A` ends at time `t`, then there is a conflict if `B` does not start between time `t`+10 and `t`+60. The conflict keeps track of the contributing activities, which in this example includes activities `A` and `B`. In addition, the conflict computes the start time intervals for moving an activity that would repair the conflict. Continuing with the example, the repair interval for `B` would be from `t`+10 to `t`+60. Activity `A` could also be moved to a different repair interval.

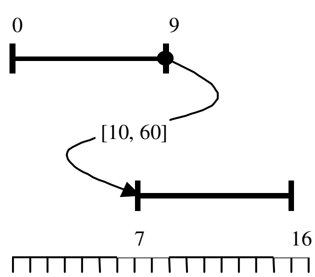

Figure 5: A violated temporal constraint conflict.

An *unassigned reservation conflict* is a reservation in an activity that has not been assigned to a resource or state variable of the required type. Resource and state variable types are defined in the model, and the plan can have multiple instances of the same type (e.g., multiple power sources). The plan keeps a timeline for each instance representing the value of the resource or state variable value. An unassigned reservation conflict is repaired by selecting a resource or state variable instance and making the reservation (i.e., propagating the effects of the reservation on the timeline).

The most complicated types of conflicts are *violated timeline conflicts*. A conflict can occur on a depletable resource, a non-depletable resource, or a state variable. For state variables, there are two types of conflicts: state usage and state transition conflicts.

When a resource value at a particular time exceeds the minimum or maximum bounds of the resource, a conflict is generated. The contributing activities are the activities with reservations that use the resource during the time of the conflict (see Figures 6 and 7). For non-depletables, these are the reservations that overlap, exceeding the resource bounds. For depletables, these are all reservations on the timeline that occur at or before the conflict. If the value is above the resource maximum (i.e., *overuse*), then contributors are only those activities with reservations that reserve a positive value. Those with negative values are contributors when the resource value is below the minimum (i.e., *underuse*). The conflict also knows which activity types would repair the conflict if a

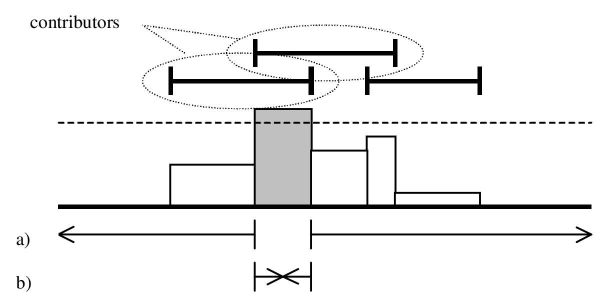

Figure 6: Time intervals that resolve a non-depletable resource conflict by a) moving a positive contributor or b) adding a negative contributor.

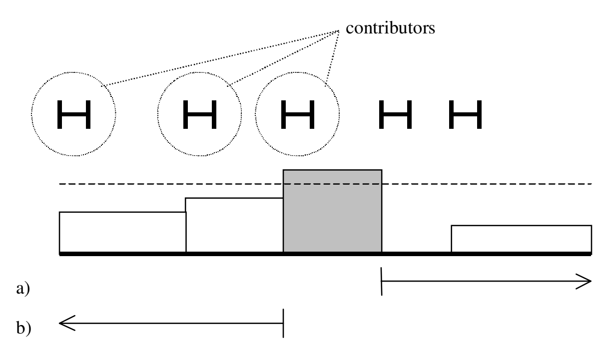

Figure 7: Time intervals that resolve a depletable resource conflict by a) moving a positive contributor or b) adding a negative contributor.

new instance were created. This includes activity types with negative usage for overuse conflicts and types with positive usage for underuse conflicts. The conflict also computes the start times indicating where to move or add activities in order to repair the conflict (see Figures 6 and 7). For moving existing activities, repair start times are all times except during the conflict. For adding new activities, repair start times are just the opposite—times during the conflict.

A state variable can have a conflict in two ways: when a reservation requires a state that is not available for the duration of the reservation (i.e., state requirement conflict), or when a reservation makes a transition that is not allowed by the state variable (i.e., state transition conflict). The contributors of a state usage conflict include the activity that changes the state (called a *changer*) and all activities that use a state (called *users*) that is different from the state during the time of the conflict (see Figure 8). In order to fix this conflict, the users might be moved anywhere but over the state in conflict. Otherwise, if we decide to move the changer, it must be moved to a time later than the state in conflict or earlier than the previous state so that this changer no longer affects the state required by the conflicting users. For state transition conflicts, the contributor is only the activity that changes the state (i.e., makes the illegal transition). Again, the changer must be moved to a time later than the state in conflict or earlier than the previous state. As with resource conflicts, new activities can be created to repair state variable conflicts. For a state usage conflict, we can add activities that can change to the desired state. These activities must be added at a time before the conflicting user, but after the conflicting changer. For state transition conflicts, we can add

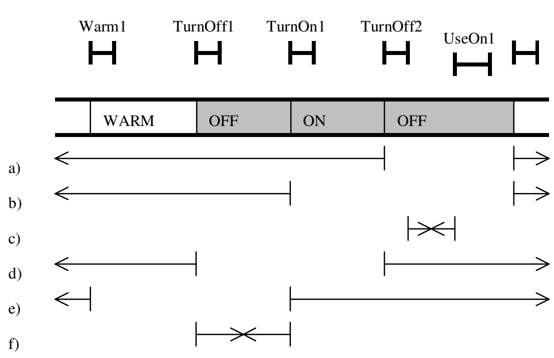

Figure 8: Time intervals that resolve a state variable usage conflict by a) moving UseOn1 b) moving TurnOff2 or c) adding TurnOn; and time intervals that resolve a state variable transition conflict by d) moving

activities that can change to a state that makes a legal transition. These activities must be added between the two conflicting changers.

4. ITERATIVE REPAIR SEARCH

ASPEN organizes its search around several types of constraints that must hold over valid plans. ASPEN then has organized around each constraint type, a classification of the ways in which the constraint may be violated. These violations are called conflicts. Organized around each conflict type, there is a set of repair methods. The search space consists of all possible repair methods applied to all possible conflicts in all possible orders. We describe one tractable approach to searching this space.

The iterative repair algorithm searches the space of possible schedules in ASPEN by making decisions at certain choice points, and modifying the schedule based on these decisions. The choice points are:

- Selecting a conflict
- Selecting a repair method
- Selecting an activity for the chosen repair method
- Selecting a start time for the chosen activity
- Selecting a duration for the chosen activity
- Selecting timelines for reservations
- Selecting a decomposition for detailing
- Selecting parameters to change
- Selecting values for parameters

Given a schedule with a set of conflicts of all types, the first step in the iterative repair algorithm is to select one of the conflicts to be attacked. Next, a method is selected for repairing the conflict. We define the possible repair methods as:

- Moving an existing activity to a new location
- Creating a new activity and insert at a location
- Deleting an existing activity
- Connecting a temporal constraint between two activities
- Disconnecting a temporal constraint between two activities
- Detailing an activity
- Abstracting an activity
- Making reservations of an activity
- Canceling reservations of an activity
- Grounding a parameter in an activity
- Applying a dependency function between two parameters

As described in the previous section, the type of conflict will determine the set of possible repair methods for any given conflict. If it was decided to try to move or delete an activity, the algorithm must decide which activity to move or delete. The type of conflict and the location of the conflict will determine the set of possible activities that, if moved or deleted, may resolve the conflict. In addition, a new start time and duration must be assigned to the activity. If it was decided to try to add a new activity, the activity type must be chosen from the list of possible types determined by the conflict. For abstract activity conflicts, the repair algorithm will most likely choose to detail the activity. If it has multiple decompositions, one of them must be chosen. Deciding to abstract an activity requires choosing which activity to abstract. When making a reservation in an attempt to resolve a conflict, a resource or state variable must be chosen for the set of possible resources or state variables. Also, if the reservation has an unspecified value, one must be chosen for it. Canceling reservations only requires choosing which reservation to cancel. If the repair algorithm has decided to connect a temporal constraint, the specific activity for the constraint must be selected. When disconnecting, only the constraint to be disconnected must be chosen. Finally, changing a parameter value requires choosing a new value for the parameter. After all decisions are made and the repair method is performed, the effects are propagated and the new conflicts are computed. This process repeats until no conflicts exist or a time limit has been exceeded.

5. SEARCH HEURISTICS

All throughout the iterative repair algorithm, many decisions must be made. In other words, there are many ways in which a conflict may be resolved. Some ways ultimately work better than others do. For example, deleting an activity may resolve a resource conflict caused by that activity. However, that activity may have been required by other activities. Or, if the activity was a high-level goal, the user might prefer to have as many goals satisfied as possible. Another typical example involves choosing a location to move an activity. Many locations may resolve the conflict being addressed, but many locations may also create addition conflicts. In order to guide the search toward more fruitful decisions, the user can define a set of search heuristics.

In ASPEN, a heuristic is a function that orders and prunes a list of choices for a particular decision in the search. Heuristics can be defined at each of the choice points in the algorithm. For example, one heuristic might sort the list of conflicts, indicating which conflicts to address first. In addition, each heuristic can use the knowledge of all previous decisions made. For example, the heuristic for deciding which method to use to resolve the conflict can (and should) be dependant on which conflict was chosen. Each heuristic can be assigned a confidence level that indicates how often the heuristic should be used. When the heuristic is not used, other heuristics can be specified, otherwise the decision will be made randomly.

ASPEN currently has some built-in domain-independent heuristics that can be used for repairing conflicts. First, a heuristic exists for sorting conflicts by their type. This heuristic prefers conflicts that require new activities (i.e., planning type conflicts) and then considers conflicts on timelines (i.e., scheduling type conflicts). This heuristic seems to work well and therefor has a high level of confidence for most of our models.

There is also a heuristic for selecting the repair method for a given conflict type. This heuristic prefers moving activities for repairing most types of conflicts. If move is not selected, the next preferred method is adding new activities. Finally, a small percentage of the time, it will choose to delete an activity. Obviously, these methods are only chosen for those conflicts for which they make sense (e.g., timeline conflicts). Some conflicts have only two possible repair methods, one of which is to delete, therefor making the decision much easier (e.g., undetailed activity conflicts can only be resolved by detailing or deleting the activity).

Another significant heuristic available in ASPEN is a heuristic for selecting start time intervals for activities being moved or created. This heuristic first tries selecting start time intervals that not only resolve the current conflict but also do not create any new conflicts[1]. If there are no such start times, the heuristic may try selecting times that create only a few conflicts. If this list is also empty, then it may select start times that simply resolve the current conflict. Sometimes, however, it may decide to return an empty list, indicating that this particular activity should not be moved or added.

A few other heuristics are currently being used in some of the domains modeled in ASPEN. All of them, however, are relatively simple and work well for the wide range of ASPEN models.

6. RELATED WORK

This work builds on considerable previous work in iterative repair problem solving. The high-speed local

[1] In general, ASPEN provides functions for querying the current plan about operations that can be performed or values that can be assigned without creating new violations. These algorithms are interesting in their own right, and will be discussed in future work.

search techniques used in ASPEN are an evolution of those developed for the DCAPS system [Chien et al., 1998a] that has proven robust in actual applications. In terms of related work, iterative algorithms have been applied to a wide range of computer science problems such as traveling salesman [Lin & Kernighan, 1973] as well as Artificial Intelligence Planning [Chien & DeJong, 1994; Hammond, 1989; Simmons, 1988; Sussman, 1973]. Iterative repair algorithms have also been used for a number of scheduling systems. The GERRY/GPSS system [Zweben et al., 1994; Deale et al., 1994] uses iterative repair with a global evaluation function and simulated annealing to schedule space shuttle ground processing activities. The Operations Mission Planner (OMP) [Biefeld & Cooper, 1991] system used iterative repair in combination with a historical model of the scheduler actions (called chronologies) to avoid cycling and getting caught in local minima. Work by [Johnston & Minton, 1994] shows how the min-conflicts heuristic can be used not only for scheduling but also for a wide range of constraint satisfaction problems. The OPIS system [S. Smith, 1994] can also be viewed as performing iterative repair. However, OPIS is more informed in the application of its repair methods in that it applies a set of analysis measures to classify the bottleneck before selecting a repair method. With iterative repair and local search techniques, we are exploring approaches complementary to backtracking refinement search approach used in the New Millennium Deep Space One Remote Agent Experiment Planner [Muscettola et al., 1997].

7. CONCLUSIONS AND FUTURE WORK

Planning and scheduling technology offers considerable promise in automating spacecraft operations. Planning and scheduling spacecraft operations involves generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals. We have extended and implemented a technique called *iterative repair* for automatically resolving conflicts in a plan/schedule. In addition, we have isolated a set of conflict types that identify plan violations as well as suggest ways in which to repair the violation.

Current and future work includes integrating repair planning with execution [Chien et al., 1999]. Here, the idea is to continuously replan around updated information coming from execution monitoring. As an embedded system, ASPEN would enable fast response to unforeseen events (e.g., faults or science opportunities) with little or no human interaction. In addition, we are also working on a framework for plan optimization. In this case, the objective is to find plans with high quality in addition to being conflict-free. We take an approach that parallels iterative repair called *iterative optimization*. Here, we classify a set of user preferences for certain plan characteristics. These preferences are used to calculate a score for the plan. The iterative optimization algorithm makes plan modifications suggested by the preferences in order to increase the overall score.

ACKNOWLEDGEMENTS

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

Biefeld, E., and Cooper, L., "Bottleneck Identification Using Process Chronologies," *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

Chien, S., and DeJong, G., "Constructing Simplified Plans via Truth Criteria Approximation," *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, June 1994, pp. 19-24.

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

Chien, S., Rabideau, G., Willis, J., and Mann, T., "Automating Planning and Scheduling of Shuttle Payload Operations," *Artificial Intelligence Journal Special Issue on Applications*, 1998.

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

Deale, M., Yvanovich, M., Schnitzius, D., Kautz, D., Carpenter, M., Zweben, M., Davis, G., and Daun, B., "The Space Shuttle Ground Processing System," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

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

Hammond, K., "Case-based Planning: Viewing Planning as a Memory Task," Academic Press, San Diego, 1989.

Johnston, M., and Minton, S., "Analyzing a Heuristic Strategy for Constraint Satisfaction and Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

Lin, S., and Kernighan, B., "An Effective Heuristic for the Traveling Salesman Problem," *Operations Research* Vol. 21, 1973.

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

Muscettola, N., Smith, B., Chien, S., Fry, C., Rajan, K., Mohan, S., Rabideau, G., Yan, D., "On-board Planning for the New Millennium Deep Space One Spacecraft," *Proceedings of the 1997 IEEE Aerospace Conference*, Aspen, CO, February, 1997, v. 1, pp. 303-318.

Sherwood, R., Govindjee, A., Yan, D., Rabideau, G., Chien, S., Fukunaga, A., "Using ASPEN to Automate EO-1 Activity Planning," *Proceedings of the 1998 IEEE Aerospace Conference*, Aspen, CO, April, 1998.

Simmons, R., "Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems," Technical Report, MIT Artificial Intelligence Laboratory, 1988.

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

Smith, S., "OPIS: An Architecture and Methodology for Reactive Scheduling," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

Sussman, G., "A Computational Model of Skill Acquisition," Technical Report, MIT Artificial Intelligence Laboratory, 1973.

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