To appear in Knowledge Engineering Review, 15(1), 2000

Bridging the Gap Between Planning and Scheduling

David E. Smith, Jeremy Frank[1], and Ari K. Jónsson[2]

NASA Ames Research Center
Mail Stop 269–2
Moffett Field, CA 94035
{de2smith, frank, jonsson} @ptolemy.arc.nasa.gov

Abstract

Planning research in Artificial Intelligence (AI) has often focused on problems where there are cascading levels of action choice and complex interactions between actions. In contrast, Scheduling research has focused on much larger problems where there is little action choice, but the resulting ordering problem is hard. In this paper, we give an overview of AI planning and scheduling techniques, focusing on their similarities, differences, and limitations. We also argue that many difficult practical problems lie somewhere between planning and scheduling, and that neither area has the right set of tools for solving these vexing problems.

1 The Ambitious Spacecraft

Imagine a hypothetical spacecraft enroute to a distant planet. Between propulsion cycles, there are time windows when the craft can be turned for communication and scientific observations. At any given time, the spacecraft has a large set of possible scientific observations that it can perform, each having some value or priority. For each observation, the spacecraft will need to be turned towards the target and the required measurement or exposure taken. Unfortunately, turning to a target is a slow operation that may take up to 30 minutes, depending on the magnitude of the turn. As a result, the choice of experiments and the order in which they are performed has a significant impact on the duration of turns and, therefore, on how much can be accomplished. All this is further complicated by several things:

- There is overlap among the capabilities of instruments, so there may be a choice to make for a given observation. Naturally, the different instruments point in different directions, so the choice of instrument influences the direction and duration of the turn.
- Instruments must be calibrated before use, which requires turning to one of a number of possible calibration targets. Recalibration is not required if successive observations are made with the same instrument.
- Turning uses up limited fuel and observations use power. Power is limited but renewable at a rate that depends on which direction the solar panels are facing.

Given all of this, the objective is to maximize scientific return for the mission or at least to use the available time wisely.

Of course, this problem is not hypothetical at all. It occurs for space probes like Deep Space One, planetary rovers like Mars Sojourner, space-based observatories like the Hubble Space Telescope, airborne observatories like KAO and SOFIA, and even automated terrestrial observatories. It is also quite similar to maintenance planning problems, where there may be a cascading set of choices for facilities, tools, and personnel, all of which affect the duration and possible ordering of various repair operations.

What makes these problems particularly hard is that they are optimization problems that involve continuous time, resources, metric quantities, and a complex mixture of action choices and ordering decisions. In AI, problems involving choice of actions are often regarded as *planning* problems. Unfortunately, few AI planning systems could even represent the constraints in the above problem, much less perform the desired reasoning and optimization. While *scheduling* systems would have an easier time representing the time constraints and resources, most could not deal with the action choices in this problem. In a sense, these problems lie squarely in between planning and scheduling, and it is our contention that many important practical problems fall into this area.

1. Caelum Research Corporation
2. Research Institute for Advanced Computer Science

In this paper, we provide an introduction to prominent AI planning and scheduling techniques, but from a rather critical perspective – namely, we examine the prospects of these techniques for helping to solve the kind of problem introduced above. We start with traditional planning techniques in Section 2 and move to scheduling in Section 3. We have not attempted to cover the vast array of scheduling techniques developed within the OR community. Instead we limit our discussion to techniques developed and used within the AI scheduling community. In Section 4, we return to planning again and examine recent work on extending planning techniques to provide the capabilities often found in scheduling systems (and desperately needed for our spacecraft problem).

2 Planning

Open any recent AI textbook [119, 63, 118, 58] and you find a chapter on planning. Yet, it is difficult to find a succinct definition of planning, independent of the particular formalisms and algorithms being used. Fundamentally, planning is a synthesis task. It involves formulating a course of action to achieve some desired objective or objectives. In the most general sense, a course of action could be any program of actions and might include such things as conditional branches, loops, and parallel actions. In practice, the form is often restricted to simple sequences or partial orderings of actions. The objective in a planning problem can encompass many things, including achieving a set of *goals*, instantiating and performing an abstract task, or optimizing some objective function.

This definition of planning is very general and encompasses several specialized types of problems, including motion or path planning, assembly sequence planning, production planning, and scheduling. Yet, work in planning seems to have little connection with work that typically goes on in these more specialized areas. The reason is that the focus of work in AI planning has been very different. To a large extent, AI planning work has concentrated on problems that involve cascading levels of action selection with complicated logical interactions between actions. In addition, most work in planning has made strong assumptions about time, resources, the nature of actions and events, and the nature of the objective. We will talk more about these assumptions later in this section

Much of the planning work in AI has fallen into one of three camps, Classical Planning, Hierarchical Task Network (HTN) planning, and Decision-theoretic Planning. (Two other categories, case-based planning and reactive planning, will not be considered here, because they are less relevant to the solution of combinatorial optimization problems like the spacecraft problem.) For each of these approaches, there are survey articles that describe the history and techniques in greater detail [133, 134, 34, 21, 24]. We do not attempt to duplicate all this material here. In the following sections, we will give a brief introduction to these three approaches, concentrating on the relationships and shortcomings of the approaches with respect to the ambitious spacecraft.

2.1 Classical Planning

2.1.1 Representation

Over the last 30 years, much of the work done on planning falls loosely into what could be called the classical planning paradigm. In a classical planning problem, the objective is to achieve a given set of goals, usually expressed as a set of positive and negative literals in the propositional calculus. The initial state of the world, referred to as the *initial conditions*, is also expressed as a set of literals. The possible actions are characterized using what are known as STRIPS operators. A STRIPS operator is a parameterized template for a set of possible actions. It contains a set of *preconditions* that must be true before the action can be executed and a set of changes or *effects* that the action will have on the world. Both the preconditions and effects can be positive or negative literals.

Consider our spacecraft example from the introduction. Using STRIPS operators, we might model a simplified version of the action of turning the spacecraft to a target as:

```
Turn (?target):
    Preconditions:   Pointing(?direction), ?direction ≠ ?target
    Effects:         ¬Pointing(?direction), Pointing(?target)
```

This operator has one parameter, the target orientation for the turn. The preconditions specify that before a turn can be performed the spacecraft must be pointing in some direction other than the target direction. If this is the case, following the turn operation, the spacecraft will no longer be pointing in the original direction and will instead be pointing in the target direction. Note that the effects of the operator only specify those things that change as a result of performing the operator. The status of the spacecraft cameras is not changed by the operation, nor is the status of any other proposition not explicitly mentioned in the effects of the operator.

Similarly, we might model the two operations for calibrating an instrument and taking an image of a target as:

```
Calibrate (?instrument):
    Preconditions:   Status(?instrument, On), Calibration-Target(?target), Pointing(?target)
    Effects:         ¬Status(?instrument, On), Status(?instrument, Calibrated)

TakeImage (?target, ?instrument):
    Preconditions:   Status(?instrument, Calibrated), Pointing(?target)
    Effects:         Image(?target)
```

As originally defined, both the preconditions and effects for STRIPS operators were limited to being a conjunctive list of literals. However, a number of more recent planning systems have allowed an extended version of the STRIPS language known as ADL [105]. ADL allows disjunction in the preconditions, conditionals in the effects, and limited universal quantification in both the preconditions and effects. Quantification and conditionals in the effects of an action turn out to be particularly useful for expressing things like "all the packages in a truck change location when the truck changes location".

There are, however, a number of more serious limitations with the STRIPS and ADL representations:

Atomic time. There is no explicit model of time in the representation. One cannot specify the duration of an action or specify time constraints on either goals or actions. In effect, actions are modeled as if they were instantaneous and uninterruptable, so there is no provision for allowing simultaneous action or representing external or exogenous events.

Resources. There is no provision for specifying resource requirements or consumption. We cannot say that turning the spacecraft uses up a certain amount of fuel, or that taking an image uses a certain amount of power or data storage.

Uncertainty. There is no ability to model uncertainty. The initial state of the world must be known with certainty, and the outcomes of actions are assumed to be known with certainty.

Goals. The only types of objectives that can be specified are goals of attainment. It is not possible to specify a goal involving the maintenance of a condition or achievement of a condition by a deadline. In part, this is due to the fact that there is no explicit model of time. There is also no ability to specify a more general objective that involves optimization. In classical planning, optimization is generally assumed to be unimportant – simply finding a plan is good enough.

At one time or another, various researchers have extended the STRIPS and/or ADL representations to address some of these limitations. In particular, [130, 108, 125] have incorporated richer models of time into the STRIPS representation, [108, 85, 83] allow various types of resources, and [110, 47, 109, 87, 41, 116, 68, 124, 135, 67, 66] introduce forms of uncertainty into the representation. [130, 70, 138, 137] introduce more interesting types of goals, including maintenance goals and goals that involve deadlines. All of these extensions require extending the techniques for solving classical planning problems, and these extensions typically have significant, detrimental impact on performance.

2.1.2 Techniques

A number of different techniques have been devised for solving classical planning problems. It would be impossible to cover them in much detail here. Instead, we give a brief sketch of the most significant approaches, concentrating on the central ideas, advantages, and disadvantages of the approaches.

Forward State Space Search

The most obvious and straightforward approach to planning is *forward state space search (FSS)*. The planner starts with the world state consisting of the initial conditions, chooses an operator whose preconditions are satisfied in that state, and constructs a new state by adding the effects of the operator and removing any proposition that is the negation of an effect. Search continues until a state is found where all the goals are attained. Figure 1 gives a sketch of this algorithm.

The trouble with forward state space search is that the number of permissible actions in any state is often very large, resulting in an explosion in the size of the search space. In our spacecraft example, there are many instruments, switch-

FSS(current-state, plan)

1. If goals $\subseteq$ current-state then **return**(plan)
2. **Choose** an action A (an instance of an operator) with preconditions satisfied in current-state
3. If no such actions exist, then **fail**.
4. Construct a new state s' by:
   removing propositions in current-state inconsistent with effects of A
   adding the effects of A into current-state
5. FSS(s', plan | A)

Figure 1: A non-deterministic forward state space search algorithm. The algorithm is initially called with the initial state and an empty plan. The statement beginning with **choose** is a backtrack point.

es, and valves that can be activated at any given moment. In addition, there are an infinite number of different directions that the spacecraft could turn towards. In order for FSS search to be successful, strong heuristic guidance must be provided so that only a small fraction of the search space is explored. For this reason, most planning systems have used a much different search strategy. However, there are two recent and notable exceptions: in [8, 7], Bacchus uses formulas in temporal logic to provide domain specific guidance for a forward state space planning system, and Geffner [23] automatically derives strong heuristic guidance by first searching a simplified version of the space. Both planning systems have exhibited speed that is competitive with the most successful techniques described below, although the resulting plans are often considerably longer than the plans generated by those other techniques.

Goal-directed Planning

Until recently, most classical planning work has focussed on constructing plans by searching backwards from the goals. The basic idea is to choose an action that can accomplish one of the goals and add that action to the nascent plan. That goal is removed from the set and replaced by *subgoals* corresponding to the preconditions of the action. The whole process is then repeated until the subgoals that remain are a subset of the initial conditions. Like the forward state space approach, this approach is relatively simple as long as the nascent plan is kept totally ordered. However, if the actions are left only partially ordered, some additional bookkeeping is necessary to make sure that the unordered actions do not interfere with each other. This algorithm is sketched in Figure 2.

Subgoal(goals, constraints, plan)

1. If constraints is inconsistent, **fail**
1. If goals $\subseteq$ initial-conditions then **return**(plan)
2. Select a goal $g \in$ goals
3. **Choose** an action A (an instance of an operator) with g as an effect
4. **Choose**:
   Subgoal(goals-g+preconditions(A), new-constraints, plan+A)
   If A is already in the plan, Subgoal(goals-g, new-constraints, plan)

Figure 2: A non-deterministic goal-directed planning algorithm. The algorithm is initially called with the goals and an empty plan. Statements beginning with **Choose** are backtrack points. Note that in step 4, even if a suitable action exists in the plan, adding another copy must still be considered as an alternative in case sharing produces conflicts.

Various strategies have been developed for doing the bookkeeping mentioned above [127, 29, 91, 80]; the most widely used being the *causal-link* approach popularized by McAllester [91]. Along with a plan, a set of causal links is maintained indicating propositions that must be preserved in between certain actions in the plan. Thus, when an action is added to the plan to establish a particular subgoal, a causal link is also added to make sure that the subgoal is preserved between the establishing action and the action that needed the subgoal. In our spacecraft example, suppose that our goal is to have an image of a particular asteroid, Image(Asteroid), and the plan already contains the action, TakeImage(Asteroid, Camera), for taking the image. We therefore have a subgoal, Pointing(Asteroid), of having the spacecraft pointed in the direction of the asteroid. To accomplish this subgoal, the planner adds the action, Turn(Asteroid), of turning the spacecraft to point at the asteroid. Along with this turn action, the planner would add the causal link to preserve the proposition pointing(asteroid) in between the Turn action and the action of taking the image.

Causal links must be checked periodically during the planning process to make sure that no other action can *threaten* them. In that case, additional ordering constraints are imposed among the actions in order to eliminate the threat. For the spacecraft, a second turn action (to a different heading) would threaten the causal link for the first heading. This turn action would, therefore, have to come either before the first turn action or after the TakeImage action.

Planners that make use of this approach are often called *partial order causal link (POCL)* planners. Good introductions to POCL planning can be found in [133, 119, 34].

Despite all the effort invested in goal-directed classical planning (particularly POCL planning), these planners have not met with much practical success. Although the branching factor is typically lower when searching backwards from goals (rather than forwards from initial conditions), it is still big enough, and there is considerable bookkeeping involved. As a consequence, the success of goal-directed planning depends just as heavily on strong heuristic guidance as FSS planning. Interestingly enough, guidance often seems to be more difficult to express in the goal-directed search paradigm. Without such guidance, these planners have generally been limited to solving problems that involve on the order of a dozen actions.

POCL planning has been extended beyond the classical planning paradigm. The most well known and widely distributed POCL planner, UCPOP [107], handles operators with quantified conditional effects, along with other features of the ADL language. POCL planners have also been constructed that can handle time [130, 108], metric quantities [108], and uncertainty [110, 47, 116, 68, 87, 41, 67, 66]. Unfortunately, these planners have generally been unable to solve problems involving more than a handful of actions.

Graphplan

In 1995, Blum and Furst introduced a planning system, called Graphplan [18, 19], that employs a very different approach to searching for plans. The basic idea is to perform a kind of reachability analysis to rule out many of the combinations and sequences of actions that are not compatible. Starting with the initial conditions, Graphplan figures out the set of propositions that are possible after one step, two steps, three steps, and so forth. For the first step, this set will contain the union of the propositions that hold in states reachable in one step from the initial conditions. For our spacecraft example, after one step, an image could be taken in the starting direction, or the spacecraft could be pointed in a new direction. So, all of these propositions are in the reachable set after one step. After two steps, in addition to all of the propositions possible after one step, an image could be taken in any of the new directions, or a data link could be established with Earth (if the spacecraft were pointed at Earth in step 1). After three steps, data could be transmitted back to Earth.

However, not all of these propositions are compatible; even if we permitted concurrent action, not all of the reachable propositions could be achieved at the same time. For the spacecraft, we cannot take an image in the original direction and turn simultaneously, because the action of taking the image requires that the spacecraft remain pointed in the original direction. Likewise, the spacecraft cannot turn towards the asteroid and turn to face Earth at the same time. Graphplan captures this notion of incompatibility by inferring binary mutual exclusion (*mutex*) relationships between incompatible actions and between incompatible propositions. The rules for mutual exclusion are remarkably simple:

- Two actions are mutex at a given step if either:
  - they have opposite effects
  - an effect of one is opposite a precondition of the other
  - they have mutex preconditions at that step
- Two propositions are mutex at a given step if either:
  - they are opposite literals
  - if all actions that give rise to them are mutex at the previous step

Using these rules, Graphplan can infer the following:

1. After one step it is not possible to be oriented towards Earth and be oriented towards the asteroid.
2. After two steps it is not possible to have the asteroid image and be oriented towards Earth, nor is it possible to have the asteroid image and have a communications link established.

3. After three steps it is still not possible to have both the asteroid image and have a communications link established.

Using this simple mutex information, the goal of having the asteroid image transferred back to Earth cannot be attained until after five steps. As a result, Graphplan would not bother to actually search for a plan until proposition level six of the plan graph had been constructed. The key features of the first two levels of the plan graph are illustrated in Figure 3 .

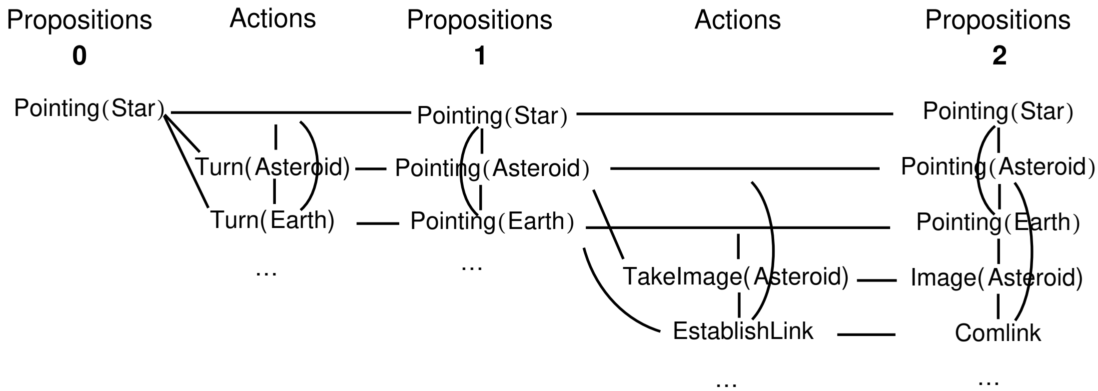

Figure 3: Key features of the first two levels of the plan graph for a spacecraft example. Vertical arcs between pairs of propositions and between pairs of actions indicate mutex relationships. Many additional actions, propositions, and mutual exclusion relationships have been omitted at each level for clarity.

Graphplan has been shown to significantly outperform the previously discussed POCL planning systems on a wide range of problems. In the recent planning systems competition held at AIPS-98 [92], all but one of the competing planning systems were based on Graphplan techniques. Roughly speaking, the best planners using this technology have been limited to problems involving less than about 50 actions.

As with POCL planning, the Graphplan technique has been extended to handle operators with quantified conditional effects and other features of ADL [57, 86, 5]. Attempts have been made to extend Graphplan to allow reasoning under uncertainty [124, 135, 20], but these efforts have, so far, not proven to be very practical. There are more recent efforts to allow certain limited consideration of time [125] and metric quantities in Graphplan [85]. It remains to be seen how far these ideas will go and how well the techniques will scale.

A more detailed introduction to Graphplan and extensions of Graphplan can be found in [134].

Planning as Satisfiability

The basic idea behind planning as satisfiability is to guess a plan length, translate the planning problem into a set of propositional formulas, and try to solve the resulting satisfiability (SAT) problem. If the formulas are unsatisfiable, the length is increased, and the process is repeated. A number of different encoding schemes have been studied to date [82, 43, 81], but the basic idea in most of these schemes is to have a propositional variable for

- each possible action at each step
- each possible proposition at each step

Each action variable indicates the presence or absence of the action at that step in the plan. Each proposition variable indicates whether or not that proposition is true at that step in the plan. SAT clauses are generated for each of the following constraints:

Initial Conditions	The propositions in the initial conditions are true at step 0.
Goals	The goals are true at the last step.
Actions	Each action occurring at step k implies that all of its preconditions are true at step k, and all of its effects are true at step k+1.
Causality	If a proposition is false (true) at step k and true (false) at step k+1, then at least one of the actions that can cause the proposition to become true (false) must have occurred at step k.
Exclusion	Two incompatible actions cannot occur at the same step.

A number of additional tricks can be employed to reduce the number of variables and clauses, particularly in cases where actions have multiple arguments [43, 81, 134].

After translation is performed, fast simplification algorithms, such as unit propagation and pure literal elimination, are used to shrink the formulas. Systematic or stochastic methods can then be used to search for solutions.

The best SAT planners and Graphplan-based planners have very similar performance. Both significantly outperform POCL planners on most problems. Like POCL and Graphplan planners, SAT planners can be extended to allow operators with quantification and conditional effects. In fact, this extension only impacts the translation process, not the solution process. Metric quantities like fuel present more serious difficulties. Wolfman [139] handles metric quantities by using linear programming in concert with SAT planning techniques. We will discuss this further in Section 4.2.

The most serious disadvantages of the SAT planning approach are:

Encoding size. The number of variables and clauses can be very large because all possible actions and propositions are represented explicitly for each discrete time point. As a result, SAT planners often require huge amounts of memory (gigabytes) for even modestly sized problems.

Continuous Time. The encoding described above is limited to discrete time and, therefore, cannot deal with actions that have varying durations or involve temporal constraints. An alternative *causal encoding* [81] could be used instead, but so far this encoding has not proven very practical or efficient.

A good introduction to SAT planning can be found in [134].

2.2 HTN Planning

Virtually all planning systems that have been developed for practical applications make use of Hierarchical Task Network (HTN) planning techniques [136, 128, 103]. The basic difference between HTN planning and classical planning is that HTN planning is about reducing *high-level tasks* down to *primitive tasks*, while classical planning is about assembling actions to attain goals. In HTN planning, the objective is usually specified as a high-level task to be performed, rather than as a conjunction of literals to be attained. For example, the spacecraft objective of having an image of a particular asteroid might be specified as the high-level task, ObtainImage(Asteroid). Planning proceeds by recursively expanding high-level tasks into networks of lower level tasks that accomplish the high-level task. The allowed expansions are described by transformation rules called *methods*. Basically, a method is a mapping from a task into a partially ordered *network* of tasks, together with a set of constraints. In the spacecraft example, a possible method for instantiating ObtainImage tasks is shown in Figure 4. According to this method, ObtainImage(?target, ?instrument) can

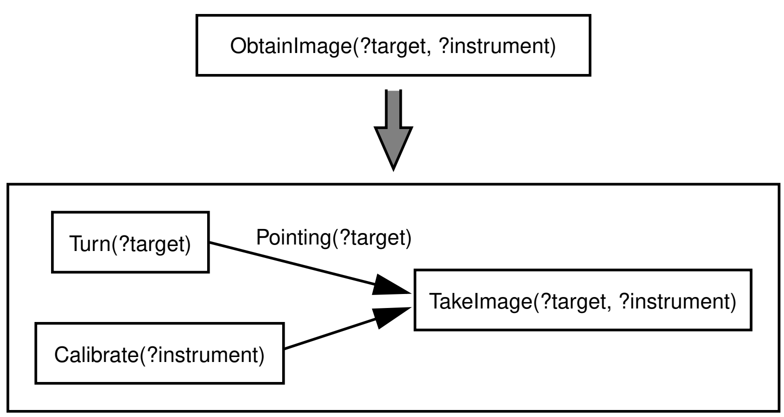

Figure 4: Simple decomposition method for obtaining an image.

be replaced with a partially ordered network of three tasks, Turn(?target), Calibrate(?instrument), and TakeImage(?target, ?instrument), with an additional constraint that the proposition Pointing(?target) must be maintained in between turning and taking the image. After each expansion, an HTN planner looks for conflicts among tasks in the network. Conflicts are resolved using *critics* that typically impose additional ordering constraints and combine or eliminate overlapping actions. Planning is complete when the resulting network contains only primitive tasks and the set of constraints is consistent. Figure 5 shows a simple pseudo-code algorithm for HTN planning.

```
HTN-Plan (N)
  1. If N contains conflicts,
  2.       If there is no way to resolve the conflicts then fail
  3.       Else choose a way of resolving the conflicts and apply it
  4. If N contains only primitive tasks, return(N)
  5. Select a non-primitive task t in N
  6. Choose a method t → E for task t
  7. N' ← Replace t with E in N
  8. HTN-Plan(N')
```

Figure 5: The basic HTN decomposition procedure for a task network N. **Choose** indicates a backtrack point.

Time and metric quantities do not present as much difficulty for HTN planning systems. These constraints can be specified within the methods and can be checked for consistency along with ordering and protection constraints. In fact, several existing HTN planning systems provide the capability for time and metric constraints [136, 128]. It is also relatively easy to combine an HTN planning system with an underlying scheduling system – once a task network has been reduced to primitive tasks, a scheduling system can be used to optimize the order of the resulting network.

The greatest strength of HTN planning is that the search can be tightly controlled by careful design of the methods. In classical planning, the preconditions and effects of an action specify when the action *could* be used and what it *could* be used to achieve. In HTN planning, methods specify precisely what combinations of actions *should* be used for particular purposes. In a sense, an HTN planner is told *how* to use actions, while a classical planner must figure this out from the action description.

There are three principle criticisms that have been levied against HTN planning:

Semantics. Historically, many HTN planning systems have not had clearly defined semantics for the decomposition methods or for the system behavior. As a result, it has been difficult to judge or evaluate such things as consistency and completeness. Although some systems still suffer from a lack of rigor, there have been recent efforts by Erol [45, 46] and others [140, 79, 13] to provide a clear theoretical framework for HTN planning.

Engineering. In general, it is difficult to develop a comprehensive set of methods for an application. One must anticipate all the different kinds of tasks that the system will be expected to address and all useful ways that those tasks could be accomplished. Methods must then be developed to cover all of those possibilities. If there are many different kinds of tasks, and/or many different ways in which tasks can be achieved, this becomes a daunting engineering task. Changes to the domain can also be problematic; e.g. if new instrument capabilities are added to our spacecraft, a whole host of new methods may be required to use those capabilities, even if those capabilities overlap those provided by existing instruments.

Brittleness. HTN planners are often seen as brittle, because they are unable to handle tasks that were not explicitly anticipated by the designer, even if the available primitive actions are sufficient for constructing a suitable plan.

There is no denying that most practical planning systems have used HTN techniques [136, 128, 103], and anyone currently considering a serious planning application would be well advised to consider this approach. Nevertheless, many researchers are dissatisfied with HTN planning because it is closer to “programming” a particular application, rather than providing a declarative description of the available actions and using general techniques to do the planning.

There is no comprehensive overview article that describes different HTN planning systems and techniques, but a clear introduction to the basic mechanisms of HTN planning can be found in [45].

2.3 MDP Techniques

For many years, researchers in Operations Research and Decision Sciences have modeled sequential decision problems using Markov Decision Processes (MDPs). In the last ten years, there has been growing interest within the AI community in using this technology for solving planning problems that involve uncertainty.

Basically, an MDP is a state space in which transitions between states are probabilistic in nature. For example, suppose that the imager in the spacecraft has a sticky shutter that sometimes fails to open. Thus, when the spacecraft attempts to take an image in a particular direction, it may or may not get one. Part of the MDP for this scenario is illustrated in Figure 6.

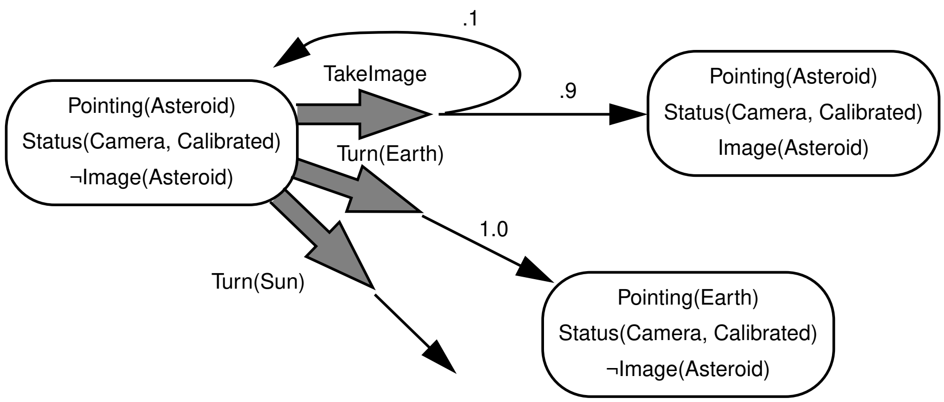

Figure 6: A fragment of the spacecraft MDP.

MDPs are traditionally solved using powerful techniques called *value-iteration* and *policy-iteration* [117]. These techniques find optimal *policies* for an MDP, which amount to conditional plans that specify which action should be taken in every possible state of the MDP.

The principal difficulty with using MDPs for planning has always been the size of the state space. If the spacecraft contains 50 switches, each of which can be either on or off, then there are 2^{50} different possible states just for the switches alone. As a result, much of the AI work in this area has concentrated on limiting the size of the state space by:

- using more compact representations that exploit the fact that, typically, many propositions and actions are independent or nearly so.
- using approximation techniques that expand only the most probable and seemingly useful portions of the state space.

These techniques have been successfully used to solve planning problems in certain carefully circumscribed domains. In particular, they have proven useful in robot navigation tasks, where there is uncertainty in the robot's location and orientation after moving [33]. The size of the state space is still a significant obstacle in broader application of these techniques. To be fair, alternative approaches (extensions of POCL, Graphplan, and SAT techniques) for planning under uncertainty have not proven very practical either.

Apart from the issue of state space size, there are some other significant limitations of the MDP framework:

Complete Observability. The MDP framework assumes that, after performing an action with an uncertain outcome, the agent (our spacecraft) can observe the resulting state. This is not a viable assumption in a world where machines have limited sensors and sensing is costly. Such problems can be represented in Partially Observable Markov Decision Processes (POMDPs), but the representation and solution techniques have generally not proven tractable for anything beyond very tiny problems.

Atomic Time. There is no explicit model of time in the representation. Actions are modeled as if they were discrete, instantaneous, and uninterruptable. Allowing concurrent action or exogenous events results in a further explosion in the size of the state space.

Goals. It is somewhat difficult to express goal attainment problems in the MDP framework. In general, they must be modeled as infinite horizon problems or as a succession of longer and longer finite horizon problems.

Another issue is that optimal policies are often large and difficult to understand. If humans must examine, understand, and carry out plans, then it is better to have a simpler, more compact plan that covers only the most critical or likely contingencies. Limited contingency planning has been explored within the classical planning framework [42], but it is not yet clear how to do it within the MDP framework.

Since our spacecraft example did not include uncertainty, the primary strength of the MDP approach is not needed for this problem. In general, optimization is more easily handled in the MDP approach than in the classical planning or HTN approaches; however, in this case, the inability to model time is more serious. The order in which observations are performed influences the duration of the turns, which in turn influences the completion time for observations. It is not obvious how to do this in the MDP framework without dramatically increasing the size of the search space.

Introductions to the use of MDP techniques in planning can be found in [21, 24]. Many of the issues and limitations listed above are discussed, along with recent work on overcoming those limitations.

3 Scheduling

Scheduling did not receive serious attention in AI until the early 1980s when Fox et al. began work on the ISIS constraint-directed scheduling system [50]. Since that time, a growing number of AI researchers have been working in the area. The common conception of scheduling in AI is that it is a special case of planning in which the actions are already chosen, leaving only the problem of determining a feasible order. This is an unfortunate trivialization of scheduling. Two well known OR textbooks on the subject [11, 112] define scheduling as the problem of assigning limited resources to tasks over time to optimize one or more objectives. There are three important things to note about this definition:

- Reasoning about time and resources is at the very core of scheduling problems. As we noted earlier, these issues have received only limited attention within the AI planning community.
- Scheduling problems are almost always optimization problems. Often it is easy to find a legal schedule by just stretching out the tasks over a long period. Finding a good schedule is much tougher.
- Scheduling problems also involve *choices*. Often this is not just confined to choices of task ordering but includes choices about which resources to use for each given task. For a given task, several alternative resources may be available that have differing costs and/or durations.

The types of choices in a scheduling problem can also extend beyond ordering and resource choices. Alternative processes may be available for some steps in a scheduling problem. For example, it might be possible to either drill or punch a hole, and these two possibilities require different machines and have different costs. Tasks can also have setup steps – optional steps that may need to be performed before the task. In our spacecraft example, instrument calibration could be regarded as a setup step, because we only need to do it once for any sequence of observations using the same instrument.

Given that scheduling problems can involve choices among resources and perhaps even process alternatives, what distinguishes scheduling from the general definition of planning we gave in the beginning of Section 2? The difference is a subtle one: scheduling problems only involve a small, fixed set of choices, while planning problems often involve cascading sets of choices that interact in complex ways. In a scheduling problem, the set of tasks is given, although some tasks may be optional and some may allow simple process alternatives. In a planning problem, it is usually unknown how many tasks or actions are even required to achieve the objective.

There is a vast literature on scheduling in OR (see [10, 11, 26, 112, 113] for introductions). What distinguishes most AI work on scheduling from the OR work is that AI work tends to focus on general representations and techniques that cover a range of different types of scheduling problems. In contrast, OR work is often focused on developing optimized techniques for specific classes of scheduling problems like flow-shop, job-shop, and sports team scheduling problems. As an example, consider a recently published text on scheduling [26]. In this text, scheduling problems are taxonomized according to roughly 10 features; each feature having between 2 and 10 values. Specific solution approaches are then presented for many of these problem classes, often based on specialized representations and algorithms. In addition, some of the algorithms and the corresponding computational complexity results depend directly on problem details, such as the number of available resources.

The purpose of this section is not to examine the multitude of different methods for solving particular classes of scheduling problems. Instead, we concentrate on approaches to *general* scheduling problems and discuss how these general approaches differ from both AI planning techniques and the more traditional OR scheduling techniques. To better ground the overview of general AI scheduling techniques, we start out by focusing on a fairly general class of scheduling problems known as resource-constrained project scheduling (RCPS) problems. Later, we show how the presented approaches can be extended to more complex choices, such as resources with different cost/duration functions and set-up steps. An instance of an RCPS problem consists of:

- A finite set of tasks, each of a certain duration.
- A finite set of resources, each having a specified capacity.
- A specification of how much of each resource each task requires (may be none at all).
- A set of ordering constraints on the tasks.

3.1 Representation

In AI, the most common approach to solving a scheduling problem is to represent it as a *constraint satisfaction problem* (CSP) and to use general constraint satisfaction techniques [16, 32, 126].[3] A constraint satisfaction problem formally specifies a set of decisions to be made and a set of constraints that limit which combinations of decisions are valid. The decisions are described in terms of variables, each of which can be assigned a value from its domain of values. The constraints are described in terms of relations that specify which combinations of value assignments are valid for the participating variables.

Two main approaches have been explored in modeling scheduling problems as constraint satisfaction problems. The distinguishing factor is which decisions are being made when constructing a schedule. The possibilities are:

- Assign start times to each task so that all given time and resource constraints are satisfied.
- Impose ordering constraints among tasks so that all the given time and resource constraints are satisfied.

In this section, we first give a brief overview of these two core approaches and their strengths and weaknesses. We then go on to discuss how these two approaches map into different constraint representation techniques, including constraint optimization problems.

3.1.1 Selecting Start Times

A very natural way of representing RCPS problems as constraint satisfaction problems is to:

- Define a variable that represents the start time of each task. The variable's domain consists of all possible starting times in a discretized interval defining the scheduling horizon.
- Specify constraints enforcing the given task orderings; for example, if task A must come before task B, then the start of B must be no earlier than the start of A plus the duration of A.
- For each time point and each resource, specify the constraint that the total usage of all tasks active at that point does not exceed the capacity of any resource.

Decisions then take the form of assigning individual start times to tasks.

Much of the initial work on scheduling as constraint satisfaction was done using this approach and, for certain applications, it continues to be the favored representation. For complex resource problems, such as when capacity and usage change over time, this approach makes it possible to accurately determine the remaining resource availability for each point in time. However, there are also limitations that arise from fixing the exact time for each task and having the set of choices depend on the number of timesteps:

Problem Size. The set of possible choices is unnecessarily large, since the real number of choices is much smaller than the set of all possible assignments of tasks to time points. This makes it difficult to search for solutions, both due to the sheer size of the search space and due to the number of operations needed to change the solution candidate significantly.

3. The development of constraint representation and reasoning techniques has resulted in two closely related formalisms – constraint satisfaction and constraint logic programming. Although there are some differences between the two, the core scheduling approaches and techniques are essentially the same. Therefore, we will present AI scheduling techniques in terms of the core concept of constraint satisfaction problems and disregard this distinction.

Discretized Time. The approach depends on discretized time, making it necessary to define atomic timesteps before the problem can be solved. To make matters worse, the size of the representation depends on how time is discretized – using hours for measuring time results is a very different representation from using seconds.

3.1.2 Ordering Tasks

The other common representation is based on the idea that two tasks that are ordered will not compete for the same resource. Defining ordering variables for pairs of tasks, we get the following constraint representation:

- A Boolean variable for each ordered pair of tasks, representing that the first comes before the second. (Note that this gives rise to two ordering variables for each pair of tasks. If both of these variables are assigned "false" then the two tasks may overlap; assigning "true" to both variables is disallowed.)
- Constraints among ordering variables, which both encode pre-existing ordering constraints and also enforce the proper order of tasks based on assignments to these variables.
- Constraints for determining the possible start times for each task, based on the ordering.
- Constraints enforcing that, if each task starts as early as possible, then no resource will be overused.

This representation permits the use of constraint propagation (described in section 3.2.1) to keep track of possible start times, since the only decisions made are those that order (or fail to order) operations. Keeping track of the possible start times for each task serves two purposes. First, any ordering that violates a given time bound can be identified when the set of possible start times for a task becomes empty. Second, keeping track of this derived information makes it possible to effectively determine whether resources are in danger of being over-subscribed or not and, thus, guarantee that the final schedule is valid. Keeping track of the possible start times is relatively straightforward. In fact, the most commonly used approaches are based on the same principles as Ford's algorithm [49] and other techniques from OR that determine the set of possible execution times for each task given a partial ordering of the tasks.

The task-ordering approach goes a long way towards addressing the limitations outlined above. For almost any realistic scheduling problem, the resulting search space is significantly smaller. There are more decisions to be made (n^2 rather than n, where n is the number of tasks), but the number of options for each decision is much smaller (2 rather than T, where T is the number of time steps). The task-ordering representation is also independent of time discretization, as there are algorithms that can keep track of the range of start times without using discretized time points (see for example [37]). This is because mapping a partial ordering into a set of possible start times is a question of arithmetic calculations, not a question of selecting specific times. Consequently, minimal schedule length can be determined directly from a given partial ordering, without searching through alternative start time assignments. Finally, this representation can be very useful in situations where durations are uncertain, as a partial ordering provides much more flexibility than a fixed time-stamped schedule.

When using an ordering-based encoding, ordering decisions need only be made until the system can guarantee that no resources are over-subscribed. However, as resource demands become more complex, verifying that resources are not over-subscribed becomes more challenging. Thus, despite its advantages, the task-ordering approach still has at least one significant weakness:

Complex Resources. In cases where the resource requirements depend on when a task is scheduled, it is difficult to determine correctly whether or not the resource capacity is exceeded if only a bound on the task execution time is known.

Some progress has been made towards addressing this weakness for certain aspects of ordering-based scheduling. Examples include an approach to handling changing resource capacities in an ordering-based framework [31] and techniques for effectively evaluating resource contention without the use of absolute time assignments [15]. The technique presented in [12] includes necessary conditions for the existence of a solution to an RCPS problem which can be used to prune the search space. The same paper also shows that these conditions can be used to adjust time bounds and presents a way to "lazily" impose ordering constraints that mimic the order-based encoding described above.

3.1.3 Scheduling as Satisfiability

In recent years, many AI researchers have studied satisfiability problems as an alternative to general constraint satisfaction problems. A satisfiability problem is a constraint satisfaction problem where each variable is a Boolean variable and each constraint is in the form of a disjunctive clause of literals, each of which represents a variable or the negation of a variable. The work in this area has resulted in the development of various fast and effective satisfiability algorithms that take advantage of the uniform structure of such problems (for examples of recent developments see [14, 90, 82]). To take advantage of these fast algorithms, researchers have looked at solving scheduling problems as satisfiability problems. Just as for scheduling as general constraint satisfaction, the satisfiability approach requires scheduling problems to be translated into satisfiability problems. This process is significantly more involved since temporal information must be represented with Boolean variables, and all constraints must be specified as clauses. The details of mapping job-shop scheduling into satisfiability are given in a comparison study that applied a number of different satisfiability algorithms to the resulting problems [32]. Some of these algorithms are discussed in Section 3.2.

It should, however, be noted that applications of satisfiability translations are limited; the representation of time is discrete, and it is difficult to effectively represent arithmetic relations and functions. Furthermore, there are indications that the expense of representing the temporal constraints as clauses negates any advantages provided by the faster algorithms [74].

3.1.4 Scheduling as Constraint Optimization

Although scheduling problems are often optimization problems, the core constraint approaches we have described above do not provide a mechanism for representing and utilizing evaluation functions. However, this turns out to be a relatively easy problem to fix, as a constraint satisfaction problem is easily extended to a *constraint optimization problem* (COP). All that is required is a function that maps a solution to a real-valued cost or score function[114]. Constraint optimization problems (COPs) have not been studied as extensively as regular CSPs, but the studies have, nonetheless, provided a number of useful techniques for solving COPs. Many of these methods are adaptations of constraint satisfaction methods, while others make use of earlier optimization methods like simulated annealing. Later, when we turn our attention to methods for solving constraint scheduling problems, we will return to the issue of optimizing schedules within the constraint reasoning framework.

3.1.5 Extensions

Various extensions to the RCPS problem have been explored in the AI scheduling community. The simplest extension is where there is a choice between resources, such as in multiple-machine job-shop scheduling. This can effectively be represented by a variable for each task-resource pair that indicates the choice of which resource the task will utilize. Within the constraint representation, it is relatively straightforward to make duration and other aspects of the task depend on the resource chosen and to make the resource demand depend on when the task is performed. Additionally, constraint-based approaches can handle other types of extensions, such as setup steps that may or may not need to occur before a task. Examples of such setup steps include the need to calibrate a camera that has not been used for a certain amount of time and the need to warm up an engine that has long been idle. Setup steps can be represented as optional tasks with zero duration/cost under certain circumstances and non-zero duration/cost under others.

3.1.6 Limitations

Although the constraint framework is very general and can represent complex scheduling problems, there is a limit to how far the paradigm can be extended. Consider all of the choices for our ambitious spacecraft: choosing to do a particular experiment leads to a choice of which instrument to use, which leads to a choice of which calibration target to use. Furthermore, the order in which experiments are performed influences all of these choices. Representing this problem as a CSP would require that every conceivable task, choice, and decision point would have to be represented explicitly in the constraint network. Not only will this result in a large problem with hundreds, if not thousands, of variables and constraints, but each constraint will be quite complex. In effect, when there is a choice of tasks to perform, the constraints involving those tasks must be made conditional on the presence or absence of the task. For example, consider the constraint that an experiment must fit within a particular time window; this constraint would seem to be representable with two inequalities. Taking into account the possible duration of turning, the possibility of having to calibrate the instrument and the choice of possible calibration targets, this simple pair of inequalities has been turned into a complex, conditional arithmetic constraint.

In sum, as more choice is introduced into a scheduling problem, the number of variables becomes large and the constraints become complex and cumbersome. As a result, this approach often becomes impractical for problems (like the spacecraft problem) that have many interacting choices.

3.2 Solving Constraint Satisfaction Problems

There are a wide variety of algorithms for solving CSPs, but most fall into one of two categories. *Constructive search* strategies attempt to build a solution by incrementally making commitments to variables, checking the constraints, and backtracking when violations are found. *Local search* strategies begin with a complete assignment of values to the variables and attempt to "repair" violated constraints by changing the value of a variable in one of those constraints. In the following sections, we discuss these paradigms in more detail and compare the two paradigms.

3.2.1 Constructive Search

Figure 7 shows a simple constructive search algorithm for solving CSPs. At each level of the search, an unassigned

ConstructiveSearch(CSP)

1. If any constraint is violated then **fail**
2. Else, if all variables are assigned values then **return**(CSP)
3. Select an unassigned variable v
4. **Choose** a value v_i for v
5. Let CSP' = propagate($CSP \cup v = v_i$)
6. ConstructiveSearch(CSP')

Figure 7: A simple constructive search algorithm for CSPs using chronological backtracking. **Choose** indicates a backtrack point.

variable is selected, and the procedure tries each of the different possible values for that variable, calling itself recursively on the resulting instantiated version of the original CSP. If an assignment proves inconsistent, the procedure fails, and backtracking occurs. This procedure has several components: *variable ordering*, *value ordering, propagation strategy*, and *backtracking strategy.* The choices made for these components have dramatic impact on performance. There has been extensive research on these topics, leading to significant improvements in the performance of CSP solvers. Because CSP techniques have served as the basis for most AI approaches to scheduling, we summarize the most significant developments next.

Variable Ordering

Variable ordering concerns the selection of which variable to work on next (Step 3). There is considerable debate about what makes for a good variable ordering strategy. The best known general heuristic is the *Minimum Remaining Values* (MRV) heuristic[4] [9, 60, 123, 56, 36, 71], which chooses the variable with the fewest remaining values. Basically, this strategy tries to keep the branching factor of the search space small for as long as possible. Propagation often aids this process by continually reducing the domains of the remaining variables [17]. In practice, MRV is inexpensive to compute and works very well on many CSP problems, at least compared to most other purely syntactic strategies [36, 56, 60].

Unfortunately, for scheduling problems where the variables are ordering decisions, MRV does not help much. The reason is that there are n^2 ordering variables, all of which have two values. In this case, more powerful heuristics are needed to select ordering variables for tasks that share "bottleneck" resources. The slack-based heuristics developed by Smith and Cheng [126] choose ordering variables according to the width of task start windows and the overlap between windows. Sadeh [122] pioneered more elaborate resource profile analysis to determine which tasks are probabilistically most likely to contend for bottleneck resources. The strategy is to select ordering variables for those tasks. Although these heuristics are expensive to compute, they appear to give much better advice for scheduling problems.[5]

4. Other authors have called this heuristic *dynamic variable ordering* [56], *dynamic search rearrangement*, and *fail first* [60, 123].

5. One way to interpret these heuristics is that they perform MRV on the "dependent" variables of the ordering variables, i.e. the domain of values remaining for the start times of jobs.

Value Ordering

Value ordering concerns the choice of a value for a selected variable (Step 4). If the perfect choice is made at each level of the search (and the problem has a solution), then a solution will be found without *any* backtracking. However, even if the value ordering strategy is not perfect, good value choices reduce the amount of backtracking required to find a solution. A common strategy for value ordering is to choose a value that causes the least restriction of the domains of the remaining uninstantiated variables, thereby increasing the chances that a solution will be possible. For scheduling problems, one approach is to choose the ordering for two tasks that leaves the greatest time flexibility for the tasks [126]. Another approach is to use analysis of resource profiles to choose orderings that provide the most reduction in the demand of critical resources, thereby reducing the chances of a resource conflict [122].

For CSP problems where solutions are hard to find, value ordering seems to be less critical than variable ordering. For such problems, much of the search time is spent investigating dead ends (i.e., subproblems that turn out to have no solution). Unfortunately, value ordering does not help if a problem (or subproblem) is unsolvable, because all values must be investigated to discover that fact. Order simply does not matter. Thus, for a value ordering strategy to be useful, it must be accurate enough to avoid a significant fraction of the dead ends. In contrast, for problems where many solutions are possible, value ordering is more important, because even a decent strategy can lead to a solution very quickly. In the case of optimization, a good value ordering strategy can lead to better solutions early, which in turn makes branch-and-bound techniques more effective.

Propagation Strategies

Propagation in CSP solvers (Step 5) is the mechanism for drawing conclusions from new variable assignments. Propagation strategies range from simple, quick consistency checking strategies, like *forward checking*, to powerful but costly *k-consistency* strategies that can infer new k-ary constraints among the remaining variables [53]. In between these two extremes lie several useful strategies that prune the domains of uninstantiated variables. The best known of these strategies is *arc-consistency* [94], which examines individual constraints to eliminate any values of the participating variables that will not satisfy the constraint, given the domains of the other participating variables. For example, suppose that both the variables x and y have domains $\{1, 2, 3\}$, and suppose we have the constraint that $x < y$. By arc-consistency, the value 3 can be eliminated for x and the value 1 can be eliminated for y. In solving a CSP, choosing and instantiating a variable can cause a cascade of propagation using arc-consistency. The advantage of this is that constructive search does not need to consider these eliminated variable assignments, thereby reducing search cost [120]. Arc-consistency has been extended in various ways to work with n-ary constraints and continuous variables. PERT chart analysis is a special case of the latter extension.

Another form of propagation commonly used in scheduling applications is called *edge-finding* [27, 104] (see also [16] for a more general discussion). Edge-finding is similar to arc-consistency in that it reduces the legal domains for variables. However, instead of dealing with a single constraint, edge-finding deals with a group of constraints that tie a set of tasks to a common resource. Figure 8 shows an example of four tasks, A, B, C, and D, that each require exclusive use of a common resource. Task A can start at any time within the interval [0, 4], while tasks B, C, and D can start

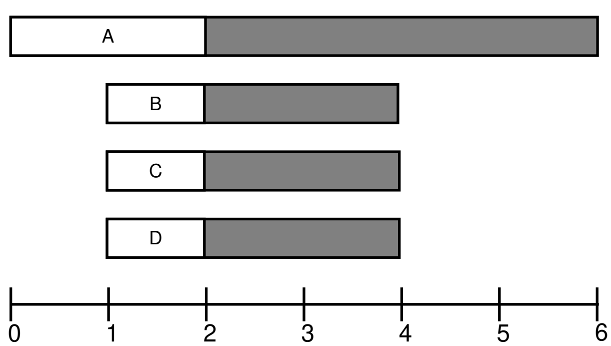

Figure 8: Example of a task window that can be narrowed by edge-finding.

within the interval [1,3]. If we consider any two (or three) of the tasks, none of the start times can be eliminated. However, if we consider the four tasks together, it is possible to conclude that task A must start no earlier than at time 4; if task A is scheduled before that, there is not enough room for the remaining tasks in the window [1,4]. Edge-finding can significantly reduce the time required to solve scheduling problems [104].[6]

For scheduling problems, as well as general CSPs, the simple, fast propagation strategies have generally proven to be the most useful. Forward checking, arc-consistency, edge-finding, and variations of these strategies are the most commonly used. More complex consistency techniques have generally proven too costly in either time or storage requirements [14].

Backtracking Strategies

The procedure in Figure 7 performs *chronological backtracking*. When a variable assignment fails, it backs up one level and tries another assignment for the preceding variable. If the possibilities for that variable are exhausted, it backs up another level, and so on. Many constructive search algorithms for CSPs use more sophisticated backtracking strategies that identify the variables responsible for the failure and backtrack directly to the appropriate level. These strategies include *backmarking*, *backjumping, conflict-directed backjumping* (CBJ) [see 71, 101, 115 for overviews], and *dynamic backtracking* [62, 14]. Of these, CBJ has generally proven to be the best compromise between cost and power.

It is worth noting that different backtracking techniques can also be used for constraint optimization problems. All of the techniques above can be combined with branch-and-bound style algorithms [10] which prune search if the minimum cost of any satisfying solution exceeds a lower bound. The addition of branch-and-bound pruning provides more opportunity for pruning and permits propagation techniques and heuristics to take advantage of the optimization criteria, as well as the constraints.

Not all CSP backtracking techniques are based on backtracking to a recent conflict; an extreme, but often effective, alternative is to backtrack all the way to the first decision. The simplest such techniques, called *iterative* or *stochastic sampling* methods [89, 25], are based on starting over whenever a conflict is found. Slightly more sophisticated techniques [73, 69] allow for limited chronological backtracking before the search process is restarted. Limited discrepancy search (LDS) [72] is an interesting approach that combines elements of traditional backtracking and sampling. LDS uses a basic backtracking scheme to systematically sample the complete set of possible solutions, in order of increasing number of discrepancies from the given heuristics. The intuition behind LDS is that the value heuristics are correct most of the time, so only a small number of deviations need to be made to find a solution. As a result, it is often more effective to search in order of increased number of discrepancies, rather than simply follow the structure of the search space.

3.2.2 Local Search

Local search techniques provide a very different way of solving CSPs than constructive search techniques. Local search typically begins with a full assignment of values to variables, even though some of the constraints may be violated. The basic idea is to then gradually modify the assignment by changing the value of some or all of the variables in an attempt to move closer to a valid solution. This is done by repeatedly generating a "neighborhood" of new assignments, ranking those assignments, and choosing the best one. Each of these actions is called a *move*. Since the effectiveness of this approach is highly dependent on the initial assignment, a new initial assignment is often generated after a specified number of moves have failed to find a solution. Each of these cycles, consisting of generating an initial assignment and performing a number of moves, is called a *try*. After enough tries, without a solution being found, the process terminates with a failure indication. This process is shown in Figure 9.

The sketch in Figure 9 describes a family of local search algorithms. To instantiate an algorithm, one must specify the procedures used to *generate neighbors*, *rank the neighbors*, and *select the neighbor* to move to next. The performance of local search depends heavily on these procedures.

A neighbor is an assignment that is "nearby" the existing assignment; typically, this means an assignment that differs in the value of exactly one variable (although other variations have been explored). Neighborhood generation procedures differ in terms of how many and which neighbors they generate. One approach [96, 30] is to select a single vari-

6. Interestingly, edge-finding appears to be closely related to Freuder's notion of *Neighborhood Inverse Consistency* (NIC) [55]. In NIC, values are eliminated for a variable by trying them in the reduced CSP consisting of all constraints connected to the variable, the variables in those constraints, and the constraints between those "neighboring" variables.

```
LocalSearch(CSP)
1. Repeat for tries
2.     Generate an initial assignment, A, for the variables in CSP
3.         Repeat for moves
4.             If no constraints are violated return(A)
5.             Select a neighboring assignment, A'
6.             Set A ← A'
```

Figure 9: Local Search algorithm sketch.

able that is in a violated constraint and then generate the set of assignments that only differ in the value for the selected variable. This approach generates a relatively small set of neighbors. A more thorough, but more expensive, approach is to generate neighbors for every variable that occurs in any violated constraint [30].

Once the assignments have been generated, they are ranked. Typically, the ranking is based on the number of satisfied constraints [97, 30, 96], but the constraints can also be weighted according to how important they are to satisfy [114]. After the assignments are ranked, one of the neighbors is selected as the new current assignment. A common strategy is to select the highest ranking neighbor, but probabilistic selection strategies have also been employed [30]. Other forms of selection permit any improving state to be selected (see for example [59, 97]).

Local search algorithms may suffer from problems with *plateaus*, *local minima*, and *cycling*. Plateaus are large regions of the search space in which no assignment improves the value of the objective function. In these regions, local search algorithms have little guidance and often wander aimlessly. Local minima are regions where all neighboring assignments are worse than the current assignment. Greedy selection heuristics can lead to cycling behavior in local minima, because a neighboring state is selected in one move and the local minima is selected again during the next move (see [52] for a discussion of these issues for satisfiability problems).

The AI community has developed some techniques to address these problems. One method used to escape local minima and plateaus is *random noise* [121]; rather than always choosing the best neighboring assignment, a random neighbor is occasionally chosen instead. This approach has led to significant improvement for a number of local search methods. Random moves force local search away from local minima, cycles, and plateaus, permitting exploration of other parts of the search space, similar to the popular simulated annealing approach [84]. Another technique for escaping local minima, plateaus and cycles is to change the ranking of neighbors during search. This can be accomplished by either learning new constraints [28] or by increasing the penalty for violating certain constraints [51, 132]. The latter techniques essentially modify the ranking function in order to make the current assignment less attractive, leading the search algorithm to explore other parts of the search space. Tabu search [65] has a similar objective: it directly breaks cycling by preventing the return to recently explored options.

Empirical studies have shown that local search methods often solve problems faster than constructive search techniques. For this reason, these methods have been used for solving a number of large practical problems. There are, however, two significant drawbacks to the use of local search:

Completeness. With constructive search, if there is a solution, it will eventually be found. There is no such guarantee for local search. In addition, a local search procedure can only fail to find a solution; it can never conclude that there is no solution. This means that local search methods always take a long time before failing, even on very small unsolvable problems.

Optimization. It is difficult to do optimization within a local search framework. One cannot simply combine the optimization criterion with a ranking function for choosing neighbors. If the optimization function dominates, the search process is prevented from finding valid solutions. On the other hand, if violated constraints are made paramount, then the optimization function is limited to exploring solutions around the local minimum found by the search. One solution to this problem is to limit the local search neighborhood to only feasible solutions as in [6], but these are often difficult to characterize and the algorithm may be unable to find good solutions. Finally, it should be noted that repeated invocation of local search can generate a set of solutions, from which the best can be chosen. However, this is often inefficient, as there is no apparent analog to branch-and-bound.

For many scheduling problems, including our spacecraft problem, completeness is not a serious issue, as these problems are generally rich in valid solutions. However, as we noted above, the goal is not to simply find a solution, but to find a good solution. As a result, the difficulties associated with combining neighborhood ranking and optimization are a significant problem in using local search for scheduling.

4 Bridging the Gap

As we noted in the introduction, the ambitious spacecraft problem exhibits characteristics of both planning problems and scheduling problems. Like scheduling problems, it involves time constraints, actions with differing durations, resources, metric quantities, and optimization. However, it also involves action choice – choosing to do a particular observation leads to a choice of instrument to use, which leads to a choice of which calibration target to use. The choices for different observations also interact in complex ways. Since performing an observation leaves the spacecraft pointed in a new direction, this influences the choice of subsequent observations, instruments, and calibration targets. This kind of complex and cascading action choice is the principal issue that has been addressed in most planning research.

In the preceding sections, we have discussed both AI planning and scheduling techniques with respect to the spacecraft problem. Most classical planning techniques are unable to represent or reason about resources, metric quantities, or continuous time. Most techniques also ignore optimization. As we hinted in Section 2.1.2, there have been attempts to extend classical planning techniques to treat resources [40, 88] and metric quantities [108, 85, 83, 139], and to allow optimization criteria [83, 139, 131]. There have also been attempts to extend planning techniques to deal with continuous time and time constraints [130, 4, 99, 108, 61, 76, 125]. In this section, we revisit the issues of resources, metric quantities, and continuous time, and we examine some of the recent attempts to extend planning techniques into these areas.

4.1 Resources

In the AI community, the term *resource* has been used to refer to a number of different things, from discrete sharable facilities (like a group of identical machines in a factory) to continuous quantities that are consumable and renewable (like fuel). Here we will use the term resource in the former sense: to refer to something like a machine, instrument, facility, or personnel that is used exclusively during an action, but is otherwise unaffected by the action. In our spacecraft example, the attitude control system, the cameras, and many other instruments would be resources. Resources have received relatively little attention in the planning community, perhaps because many early planning formalisms did not allow concurrent actions.

In scheduling, resources are often classified as single-capacity or multiple-capacity. Single-capacity resources present no special difficulty for most planning formalisms. If two actions both require the same resource, the actions conflict and cannot be allowed to overlap. In the POCL framework, this can be enforced by adding ordering constraints between any such actions that can possibly overlap.[7] In Graphplan, any actions with resource conflicts can be made mutually exclusive in the plan graph. This is enough to prevent them from being scheduled concurrently. In SAT planning, mutual exclusion axioms can be added for every pair of actions with resource conflicts at each time point. These mechanisms are all straightforward, but the presence of resource conflicts may substantially increase the difficulty of finding solutions to a problem.

Multiple-capacity resources present much more difficulty, for two reasons: 1) it is more difficult to identify groups of actions with potential resource conflicts and 2) the number of potential ways of resolving the conflict grows exponentially with the number of actions involved. These are the same issues faced when dealing with multiple-capacity resources in scheduling problems. As a result, the techniques used in planning bear a close resemblance to techniques that have been developed in scheduling. The O-Plan planner uses optimistic and pessimistic resource profiles [40] to detect and resolve potential resource conflicts. The IxTeT planner [88] uses graph search algorithms to identify minimum critical sets of actions with conflicting resources. A disjunction of ordering constraints is then added to resolve the conflict.

7. This is quite similar to the mechanism of promotion and demotion for handling threats.

4.2 Metric Quantities

A number of the planning techniques discussed in Section 2.1 have been extended to allow real-valued or *metric* quantities. For the spacecraft example, the direction that the spacecraft is pointing and the fuel remaining are metric quantities. The difficulty with metric quantities is that when performing an action, the change in one quantity is often a mathematical function of the change in another. For example, in a turn operation, the amount of fuel consumed is a function of the angular distance between the current and target orientations of the spacecraft. One very simple approach is to augment the preconditions and effects of STRIPS operators with equality and inequality constraints involving arithmetic and functional expressions. Borrowing notation from Koehler [85], we might describe the turn operation as:

```
Turn (?target):
    Preconditions:   Pointing(?direction), ?direction ≠ ?target,
                     Fuel ≥ Angle(?direction, ?target) * ConsumptionRate
    Effects:         ¬Pointing(?direction), Pointing(?target),
                     Fuel –= Angle(?direction, ?target) * ConsumptionRate
```

The inequality in the precondition specifies that the available fuel must be sufficient to turn the spacecraft through the desired angle. The equality in the effects specifies that the fuel will be reduced by that same amount at the conclusion of the turn. Note that, in this representation, we have not specified how the fuel changes during the turn, only what it must be before and after the turn. For many purposes this representation is sufficient. However, if two actions are allowed to use or affect the same metric quantity simultaneously, it becomes necessary to describe how the quantity changes over the course of the action. Most planners that deal with metric quantities adopt a simple discrete representation like that above [136, 85, 139] and do not allow concurrent actions to impact the same metric quantity. A few planners, like Penberthy's Zeno planner [106, 108], use a more detailed representation of how metric quantities change over time, but still place restrictions on concurrent actions affecting the same metric quantity.

The presence of metric quantities and constraints can cause great difficulty in planning. In general, the metric constraints on actions can be nonlinear or can involve derivatives. The simplest approach to dealing with metric quantities in planning is to ignore a metric constraint until the values of the variables are known precisely. At that time the constraint can be checked to see whether or not it is satisfied. If not, backtracking occurs. For a turn action, the current and target orientations, the fuel, and fuel consumption rate would all have to be known before the inequality precondition could be checked. This passive approach has been used in a few planning systems [130, 39], but the approach is quite weak, because it only detects difficulties late in the planning process and provides no guidance in actually choosing appropriate actions.

4.2.1 LP techniques

Several planners have been constructed that attempt to check the consistency of sets of metric constraints even before all variables are known. In general, this can involve arbitrarily difficult mathematical reasoning, so these systems typically limit their analysis to the subset of metric constraints consisting of linear equalities and inequalities. Several recent planners have used linear programming (LP) techniques to manage these constraints [128, 106, 108, 139].

One such planner is Zeno [106, 108], a POCL planner that continually checks metric constraints using a combination of Gaussian elimination for equalities and an incremental Simplex algorithm for inequalities. Zeno deals with nonlinear constraints by waiting until enough variables are determined that they become linear. To see how Zeno works, consider the subgoal of pointing at a particular asteroid A37. Zeno would add a turn operation to the plan, would generate the subgoal Pointing(?direction), and would post the constraint Fuel ≥ Angle(?direction, A37) * ConsumptionRate. Since Fuel and ?direction are not yet known, Zeno cannot yet do anything with this constraint. Instead, Zeno would work on the outstanding subgoal Pointing(?direction). If the spacecraft is initially pointing at Earth, the subgoal could be accomplished by requiring that ?direction = Earth. Now the inequality constraint reduces to the simple linear inequality Fuel ≥ number, which Zeno would check using its incremental Simplex algorithm. If the required fuel exceeds a maximum constraint, Fuel ≤ MaxFuel, this inconsistency would be detected by Simplex. If no violation occurs, Fuel ≥ number would be introduced as a subgoal. While Zeno exhibits an impressive collection of technical features and techniques, the planner can only handle very small problems. Basically, Zeno suffers from the same kinds of search control difficulties that plague less ambitious POCL planners. However, these problems are compounded by the fact that Zeno models continuous time and continuous change, which often means that it has a much larger set of possible action choices to investigate.

A more recent effort to deal with metric quantities is the LPSAT planner [139]. LPSAT uses LP techniques in conjunction with SAT planning techniques. The planning problem is encoded as a SAT problem as described in Section 2.1.2, except that metric preconditions and effects are replaced by Boolean trigger variables in the SAT encoding. During planning, if a trigger variable becomes true, the corresponding metric constraint is passed to an incremental Simplex solver. If the Simplex solver reports that the collection of metric constraints is inconsistent, the SAT solver backtracks in order to change one or more of the trigger variables. The performance of LPSAT is quite promising, but it shares the same disadvantages as the underlying SAT approach:

Encoding size. The number of variables and constraints can be very large because all possible actions and propositions are represented explicitly for each discrete time point.

Continuous time. The usual encoding is limited to discrete time and, therefore, cannot deal with actions that have varying durations or involve temporal constraints.

In addition, metric quantities raise the possibility that there may be an infinite number of possible actions. For example, consider a refueling action, where the quantity is a continuous function of the duration. This corresponds to an infinite number of possible ground actions which simply cannot be handled in the SAT framework.

4.2.2 ILP Planning

Another approach to handling metric quantities is to represent a planning problem as a mixed integer linear program (ILP). Two papers in this special issue [83, 131] discuss techniques for translating planning problems into ILP problems. For the most part, the encodings follow the same form as the encodings of planning problems as satisfiability problems. Variables are defined for:

- each possible action at each discrete time step
- each possible proposition at each discrete time step

Instead of true/false values, the variables take on 0/1 values, with 1 indicating that the proposition is true or the action takes place. In this formulation, the constraints between actions and propositions take the form of linear inequalities that can be constructed directly from the clauses used in the SAT planning formalism. For example, an action A having an effect E would lead to SAT clauses of the form:

$$\neg A_t \vee E_{t+1}$$

for each possible time instant t. These would be translated into the following linear inequalities:

$$(1 - A_t) + E_{t+1} \geq 1$$

As with SAT planning, similar inequalities would be required for action preconditions, for explanatory frame axioms, and for action mutual exclusion. Actions involving metric constraints can be translated in a manner similar to that described for LPSAT. A standard ILP solver can then be used to solve the set of inequality constraints.

The principle advantages to the ILP approach are:

Uniform mechanism. All axioms, including metric constraints, translate into equalities and inequalities. A standard ILP solver can then be used to solve the set of equations.

Optimization. It becomes relatively easy to specify optimization criteria, and the ILP solver can do the optimization naturally.

Dixon and Ginsberg [38] point out that many types of exclusionary constraints can be modeled much more compactly using inequalities, although it is not yet clear whether this will have a significant impact for planning problems.

Of course the ILP approach shares two significant disadvantages with basic SAT planning approaches discussed earlier:

Encoding size. The number of variables and constraints can be very large because all possible actions and propositions are represented explicitly for each discrete time point. As mentioned earlier, various tricks can be used [43] to reduce the size of the encoding, but the size can still be unmanageable for even modestly sized problems.

Continuous time. The encoding described above is limited to discrete time and, therefore, cannot deal with actions that have varying durations or that involve temporal constraints. A causal encoding [81] could conceivably be used, but this has not yet been attempted.

So far, ILP methods are not yet competitive with SAT planning methods, because the LP relaxation step is somewhat costly and seems to provide very limited pruning power. However, this area has only begun to receive serious attention.

4.3 Time

The STRIPS representation uses a discrete model of time, in which all actions are assumed to be instantaneous (or at least of identical duration). However, as we mentioned earlier, the POCL approach to planning does not depend on this assumption. In POCL planning, actions can be of arbitrary duration, as long as the conditions under which actions interfere are well defined. Vere was the first to exploit this in the development of the Devisor system [130][8]. Devisor allows actions of arbirary duration, and it also allows both goals and actions to be restricted to particular time windows. For a POCL planner, this complicates both the detection and resolution of threats, because the representation and maintenance of the time constraints between actions is more complex.

Devisor is only the first of several systems [48, 106, 108, 61, 99, 76] that have combined ideas from POCL planning with a more general representation of time and action. While we will not discuss the details and differences for all of these systems, we will describe some common ideas and a common framework that seem to be emerging from this work. In particular, many of these systems use an interval representation for actions and propositions, and they rely on constraint-satisfaction techniques to represent and manage the relationships between intervals. We will refer to this as the *Constraint-Based Interval* (CBI) approach.

4.3.1 The Interval Representation

The idea of using an interval representation of time was first introduced and popularized in AI by James Allen [3]. Rather than describing the world by specifying what facts are true in discrete time slices or states, Allen describes the world by asserting that propositions hold over intervals of time. Similarly, actions and events are described as taking place over intervals. Constraints between intervals describe the relationships between actions (or events) and the propositions that must hold before and after. In our spacecraft example, we could say that the spacecraft is turning towards asteroid A37 over a particular interval I as $\text{Turning(A37)}_{\text{I}}$. Similarly, we would say that the spacecraft is pointing at A37 over a particular interval J as $\text{Pointing(A37)}_{\text{J}}$. Following Joslin [77, 78], we will use the term *temporally qualified assertion* (TQA) to refer to a proposition, action, or event taking place over an interval.

Allen introduced a set of seven basic interval relations (and their inverses) that can be used to describe the relationships between intervals. These are summarized pictorially in Figure 10. Using these interval relationships, we can describe how actions (or events) affect the world. To see how this can be done, recall the simple set of STRIPS operators introduced in Section 2.1 for turning, calibrating, and taking images.

```
Turn (?target):
    Preconditions:  Pointing(?direction), ?direction ≠ ?target
    Effects:        ¬Pointing(?direction), Pointing(?target)

Calibrate (?instrument):
    Preconditions:  Status(?instrument, On), Calibration-Target(?target), Pointing(?target)
    Effects:        ¬Status(?instrument, On), Status(?instrument, Calibrated)

TakeImage ( ?target, ?instrument):
    Preconditions:  Status(?instrument, Calibrated), Pointing(?target)
    Effects:        Image(?target)
```

Using the interval representation, the intervals and constraints implied by a TakeImage operation over an interval A would be:

8. Technically, Devisor is based on Nonlin [127], which was a precursor to the modern notion of POCL planning.

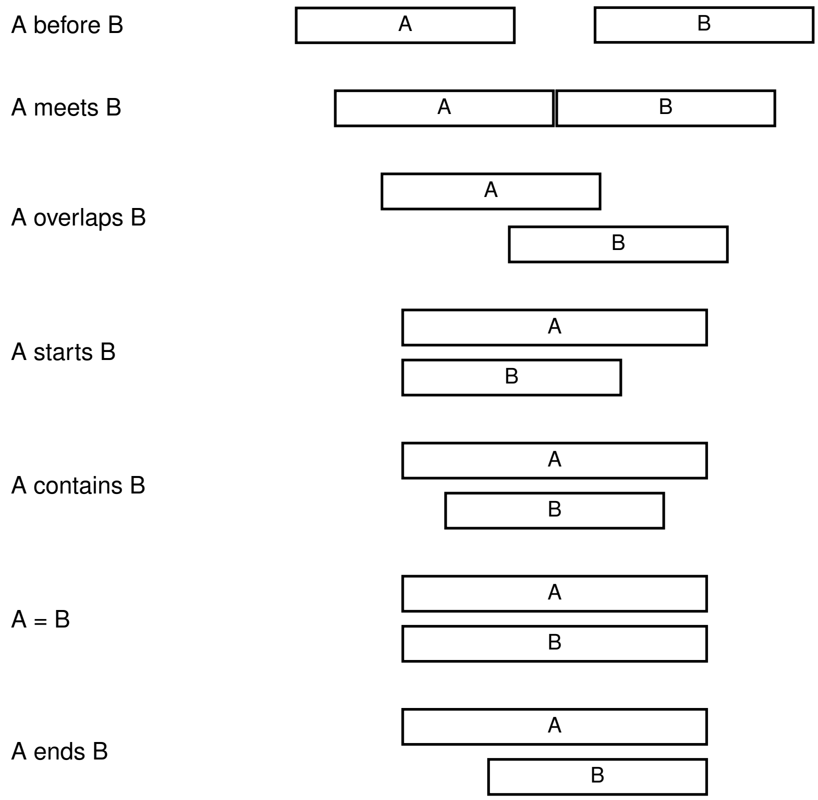

Figure 10: Graphical depiction of Allen's basic interval relationships.

$$
\begin{aligned}
\mathrm{TakeImage(?target,\ ?instrument)}_A \rightarrow\ & \exists P\ \{\mathrm{Status(?instrument,\ Calibrated)}_P \wedge \mathrm{Contains}(P, A)\} \\
& \&\ \exists Q\ \{\mathrm{Pointing(?target)}_Q \wedge \mathrm{Contains}(Q, A)\} \\
& \&\ \exists R\ \{\mathrm{Image(?target)}_R \wedge \mathrm{Meets}(A, R)\}
\end{aligned}
$$

This axiom states that if there is an action TakeImage(?target, ?instrument) over the interval A, then Status(?instrument, Calibrated) and Pointing(?target) must hold for intervals P and Q containing the TakeImage action, and Image(?target) will hold over some interval R immediately following the action. This is depicted graphically in Figure 11.

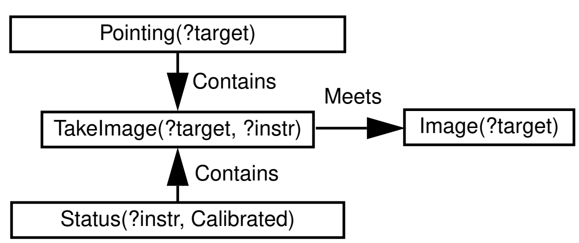

Figure 11: Graphical depiction of the interval constraints for TakeImage.

The interval representation permits much more flexibility and precision in specifying temporal relationships than is possible with simple STRIPS operators. For example, we can specify that a precondition only need hold for the first part of an action, or that some temporary condition will hold during the action itself. We can also specify that two actions must be performed simultaneously in order to achieve some condition, or that there are particular time constraints on goals, actions, or events.

Although quantified logical expressions like that given above are very general, they are also cumbersome and somewhat difficult to understand. Muscettola [99, 75] has developed a shorthand for specifying such axioms that makes interval existence implicit. For example, the TakeImage axiom above can be specified as:

TakeImage (?target, ?instrument)	contained-by	Status(?instrument, Calibrated)
	contained-by	Pointing(?target)
	meets	Image(?target)

This is interpreted to mean that if an interval exists in which a TakeImage action occurs, then other intervals exist in which Status(?instrument, Calibrated), Pointing(?target), and Image(?target) hold, and these intervals have the indicated relationships with the interval for TakeImage.[9] Using this notation, the constraints for the Turn and Calibrate operators would be:

Turn (?target)	met-by	Pointing(?direction)
	meets	Pointing(?target)
Calibrate (?instrument)	met-by	Status(?instrument, On)
	contained-by	CalibrationTarget(?target)
	contained-by	Pointing(?target)
	meets	Status(?instrument, Calibrated)

4.3.2 The CBI Planning Algorithm

With the interval representation, planning is accomplished in a way very similar to POCL planning (discussed in Section 2.1.2). The planner works backwards from the goals, adding new action TQAs to the plan, which in turn introduce new subgoal TQAs by virtue of the interval constraints. Throughout this process, additional ordering constraints may be needed between TQAs in order to eliminate conflicts. If there are no outstanding subgoals and no conflicts remain, the plan is complete. Backtracking occurs if there is no way to achieve a particular subgoal or if there is no way to satisfy the constraints governing the intervals in a partial plan. A sketch of the algorithm is shown in Figure 12.

Expand(TQAs, constraints)

1. If the constraints are inconsistent, **fail**
2. If all TQAs have causal explanations, **return**(TQAs, constraints)
3. Select a $g \in$ TQAs with no causal explanation
4. **Choose**:

    Choose another $p \in$ TQAs such that g can be coalesced with p under constraints C

        Expand(TQAs-g, constraints $\cup$ C)

    Choose an action that would provide a causal explanation for g

        Let A be a new TQA for the action,
        and let R be the set of new TQAs implied by the axioms for A

        Let C be the constraints between A and R

        Expand(TQAs $\cup$ {A} $\cup$ R, constraints $\cup$ C)

Figure 12: A non-deterministic CBI planning algorithm. The algorithm recursively finds a TQA without a causal explanation and either coalesces that interval with an existing interval or introduces a new interval into the plan that will provide a causal explanation for the interval. Statements beginning with **choose** are backtrack points.

To see how this algorithm works, consider our simple spacecraft problem where the goal is to obtain an image of asteroid A37. Initially, the model contains intervals for each of the initial conditions, as well as intervals for each of the

9. There is a subtle difference between the relations in the shorthand notation and the underlying interval relations of the same name. The interval relations can be inverted (i.e., Contains(I, J) implies Contained-by(J, I)), but we cannot invert the relations in the shorthand notation. For example, stating that:

Status(?instrument, Calibrated)	contains	TakeImage (?target, ?instrument)

would be wrong; just because an interval exists in which Status(?instrument, Calibrated) does not mean that a TakeImage action must occur. Thus, the shorthand is directional, whereas the underlying interval relations are not.

goal conditions. For our simple spacecraft example, this might look like Figure 13. There are, as yet, no constraints on

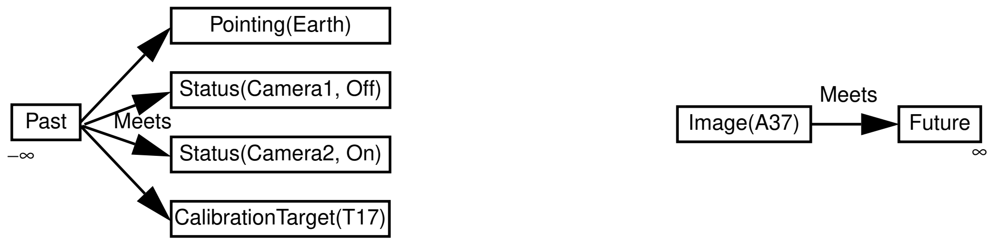

Figure 13: Initial partial plan for the spacecraft problem.

when the intervals for the initial conditions (Pointing(Earth), Status(Camera1, Off), Status(Camera2, On), and CalibrationTarget(T17)) might end. Likewise, there are no constraints on when the interval corresponding to the goal condition Image(A37) will start (except that it must be after the Past interval). There is no causal explanation for Image(A37), so that TQA is selected by the algorithm. In our example, there is only one way of producing this TQA, and that is to introduce a TakeImage action. According to the constraints, TakeImage must be contained by an interval in which the spacecraft is pointing at the target asteroid and by an interval in which the (as yet unspecified) instrument is calibrated, so these two intervals are introduced into the plan along with the corresponding constraints. We can also infer at this stage that the interval for Pointing(A37) must come after the interval for Pointing(Earth), because Pointing(Earth) meets Past, and because the spacecraft cannot point in more than one direction at once. The resulting plan is shown in Figure 14.

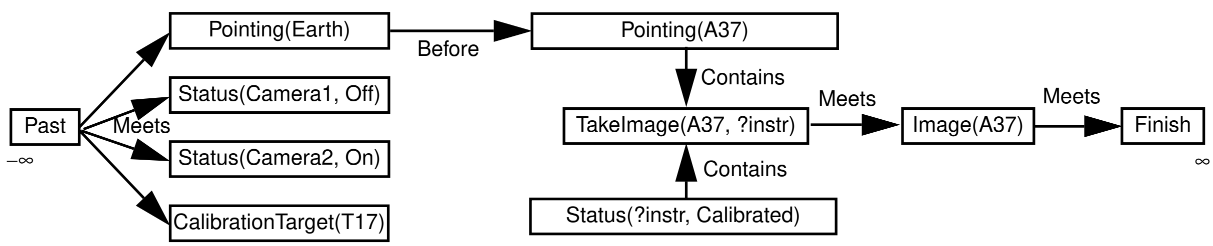

Figure 14: Partial plan for the spacecraft problem after adding a TakeImage action.

There are now two TQAs in the plan without causal explanations. To achieve Pointing(A37) a Turn step must be introduced, and to achieve Status(?instr, Calibrated) a Calibrate step must be introduced. The axioms associated with these steps cause the introduction of several more TQAs as shown in Figure 15. At this stage, we cannot yet infer anything

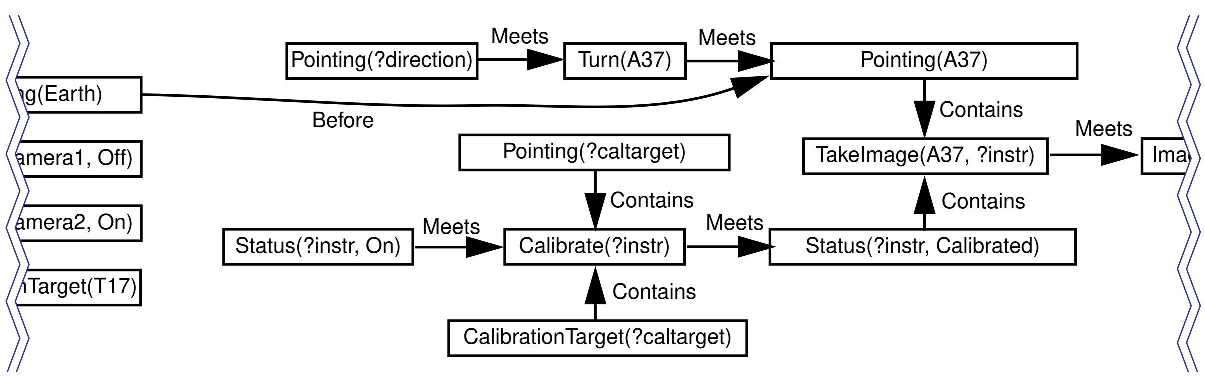

Figure 15: Partial plan for the spacecraft problem after adding a Turn action.

about the temporal relationships between Pointing(?caltarget) and any of the other three pointing TQAs, because we have not yet ruled out A37, Earth, or ?direction as possible calibration targets. However, we can choose to coalesce several pairs of intervals at this stage. In particular, we can coalesce both CalibrationTarget(?caltarget) and Status(?instr, On) with initial condition intervals, and we can coalesce Pointing(?caltarget) with Pointing(?direction). The resulting partial plan is shown in Figure 16. Only Pointing(T17) remains without a causal explanation, and it can only be achieved

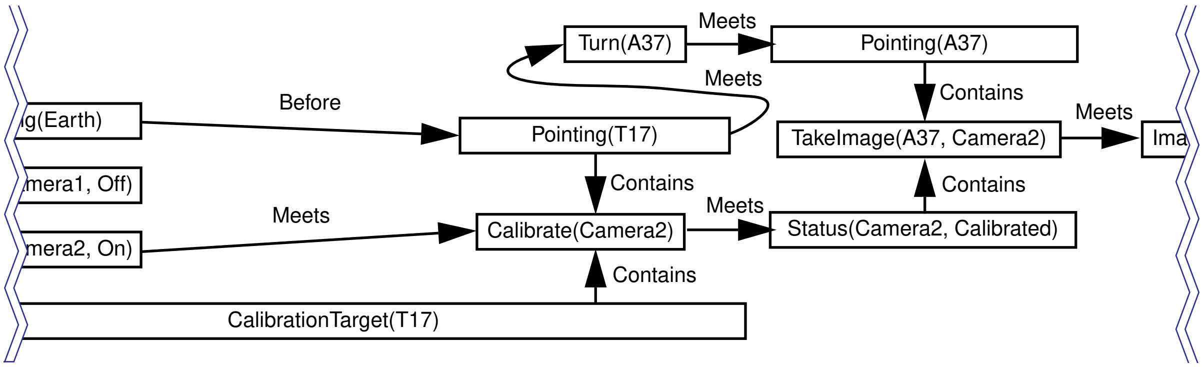

Figure 16: Partial plan for the spacecraft problem after coalescing compatible intervals.

by introducing another Turn step. After adding this step and the intervals required by the constraints on turn, the remaining unexplained pointing interval can be coalesced with the initial conditions to give the finished plan shown in Figure 17.

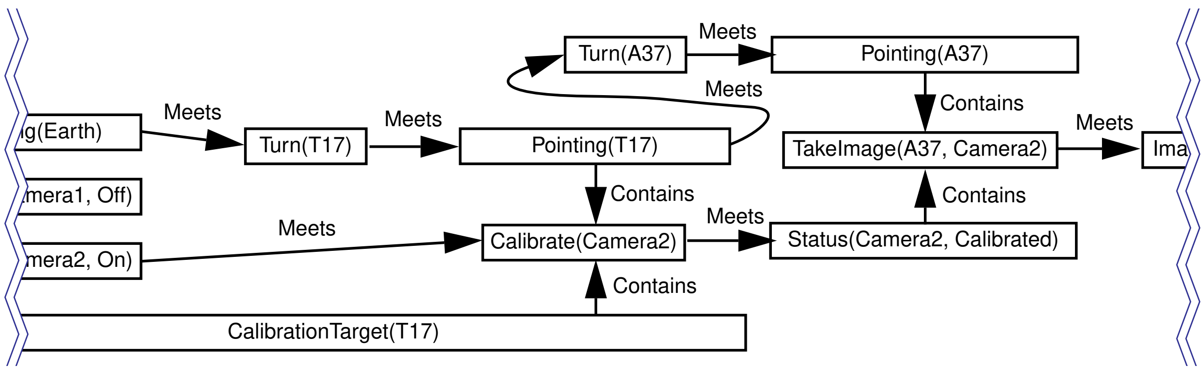

Figure 17: Complete plan for the spacecraft problem.

For this simple example, the order in which we made decisions was quite lucky. After the second step, we could have coalesced Pointing(?caltarget) with Pointing(A37) or Pointing(Earth). Likewise, we could have coalesced Pointing(?direction) with Pointing(Earth). None of these possibilities would have worked, so the planner would have ultimately been forced to backtrack and try different alternatives.

4.3.3 Using Constraint Satisfaction

So far, we have presented CBI planning in a way that emphasizes the similarities with POCL planning, but there are some important differences. For CBI planners, the temporal relationships and reasoning are often more complex. As a result, CBI planners typically make use of an underlying constraint network to keep track of the TQAs and constraints in a plan. For each interval in the plan, two variables are introduced into the constraint network, corresponding to the beginning and ending points of the interval. Constraints on the intervals then translate into simple equality and inequality constraints between end points. Interval durations translate into constraints on the distance between start and end points. Inference and consistency checking in this constraint network can often be accomplished using fast variations of arc-consistency, such as Simple Temporal Propagation [37].

In addition to handling temporal relationships, the constraint network can also be used to explicitly encode other choices, such as the possible values for uninstantiated parameters. In our example above, the variables ?instr, ?caltarget, and ?direction could appear explicitly in the network with values corresponding to the possible instantiations. If variable typing information is provided, this mechanism can be very effective in reducing the possibilities for coalescing intervals and introducing new TQAs. The constraint network can also be used to explicitly encode alternative choices for achieving a particular TQA. For example, if there were more than one possible way of orienting the spacecraft in a particular direction, an explicit variable could be introduced that corresponds to that choice, and constraint propagation could potentially be used to help refine this set of choices.

CBI planners can be viewed as dynamic constraint satisfaction engines – the planner alternately adds new TQAs and constraints to the network, then uses constraint satisfaction techniques to propagate the effects of those constraints and to check for consistency. This generality is the strength of the CBI approach:

- The interval representation can be used to express a wide variety of temporal constraints on actions, events, and fluents.
- The underlying representation of partial plans as a constraint network provides a single uniform mechanism for doing temporal inference, resolving conflicts between TQAs, and refining parameter choices for actions.

As a result, the planning algorithm is particularly simple, and much of the control of the algorithm takes the form of variable and value ordering in the underlying constraint network. In short, the interval representation and dynamic constraint satisfaction provides a clean, expressive framework for tackling planning problems. This framework also seems to mesh nicely with the CSP approach to scheduling, discussed in Section 3. In particular, we believe that edge-finding techniques could be incorporated into CBI planners to narrow the possible times for intervals. Similarly, we believe that texture-based heuristics could be incorporated to help guide the process of resolving resource conflicts between intervals.

The biggest potential drawback of CBI planning is performance. To date, CBI planning has not been studied widely, and it is not as well understood as most other planning techniques. Serious comparisons with other planning techniques have not been carried out, partly because CBI techniques have been aimed at problems that require a much richer representation of time and action. From a theoretical point of view, it is not clear why CBI planners would perform any better than POCL planners (which are currently outperformed by Graphplan and SATPLAN). As we illustrated in the example above, a CBI planner goes through the same decisions and steps as a POCL planner, yet a POCL planner only keeps track of the propositions that are relevant between actions (causal links) rather than the state of all affected propositions over time.

One might argue that using an underlying constraint network and constraint satisfaction techniques allows a CBI planner to deal with temporal constraints, variable bindings, and alternative choices more efficiently. However, this seems unlikely to have a significant impact on performance, because the same fundamental set of choices must be considered by the planner. Similarly, one might argue that the use of general constraint propagation allows a CBI planner to weed out more possibilities that lead to dead-end plans. This also seems unlikely, since there is little evidence that extreme least commitment strategies (such as those suggested by Joslin [77, 78]) actually pay off.

Several planning systems have been built that use a CBI approach, including Allen's TRAINS planner [48], Joslin's Descartes [77, 78], the HSTS/Remote Agent (RA) planner [99, 75, 76], IxTeT [61, 88], and (to a large extent) Zeno [108]. The focus of Allen's work has been primarily on formal properties and on mixed initiative planning, while Joslin's planner was intended as a demonstration of how far a least-commitment approach to planning could be taken. As we mentioned earlier, Zeno concentrated on metric quantities rather than on efficient temporal reasoning. None of these planners have been extensively tested on real world or benchmark problems.

The two most practical CBI planners are IxTeT and the HSTS/RA planner, both of which have been applied to space related problems. The HSTS/RA planner was used to generate plans during the Remote Agent Experiment [100] onboard the NASA Deep Space One spacecraft. During the experiment, the planner successfully generated complex plans that included turns, observations, navigation, thrusting, communications, and other aspects of spacecraft operations, while taking into account limited resources, task durations, and time limits. Unfortunately, in order to achieve this performance, the search process had to be carefully controlled with problem-dependent, hand-crafted heuristics.

There are two unique characteristics of the HSTS/RA planner that may have also contributed to its success. First of all, the planner makes no distinction between propositions holding over intervals and actions taking place over intervals. Instead of working on proposition intervals that do not have a causal explanation, the planner works on intervals whose constraints have not yet been satisfied. For this strategy to work, explicit constraints must be provided that indicate the possible ways of achieving each proposition. (These are often referred to as explanatory frame axioms.) For example, a pointing proposition must either be true initially or it must be achieved by turning. As a result, we would need to state that:

Pointing(?target)	met-by	Past
	or	
	met-by	Turn (?target)

Having supplied this constraint, new turning intervals would be introduced automatically whenever a pointing interval could not be coalesced with an existing pointing interval. The potential advantage of this approach is that it allows the designer to carefully control which actions are considered for given subgoals in much the same way as in an HTN planner.

A second characteristic of the HSTS/RA planner that may have contributed to its success is that it represents the world using variable/value assignments rather than propositions. In our spacecraft example, there would be one variable or *timeline,* for each camera's status, for the direction in which the spacecraft is pointing, and for the contents of the image buffer. The constraints are therefore specified in terms of the intervals over which variables take on particular values. For example, the constraints on a turning activity could be specified as:[10]

Pointing=Turn (?target)	met-by	Pointing=?direction
	meets	Pointing=?target

In this case, there is an interval in which the Pointing variable has the value ?direction, followed by an interval in which the Pointing value is in transition, followed by an interval in which the Pointing value is ?target. This kind of variable/ value or functional representation has a subtle advantage over the propositional representation: no explicit axioms or reasoning steps are required to recognize that Pointing cannot take on two values at once. Recognizing that two intervals are in conflict happens automatically as a result of the fact that different valued intervals can never overlap for any given variable.[11]

5 Conclusions

If we consider the different methods of planning discussed in this paper, fundamentally, there are three different ways that planning and scheduling has been done:

Stratified P&S. The approach usually taken in the scheduling community is to separate planning from scheduling. Action choices are made first, then the resulting scheduling problem is solved. For example, in scheduling problems where there are process alternatives for certain steps, these decisions are usually made first, then the resulting scheduling problem is solved. If there is no solution, or the solution is not good enough, an alternative set of choices is made and the scheduling problem is solved again.

Interleaved P&S. The approach taken in POCL planning (and many other classical planning techniques) allows interleaving of action choices with ordering decisions. Typically an action choice is made, and conflicts are resolved with other actions by imposing ordering constraints on the actions. Traditionally, ordering decisions for these planners have been relatively simple. The CBI approach uses a representation much closer to that used in scheduling and offers hints that powerful scheduling techniques and heuristics could be integrated into such a framework.

Homogeneous P&S. The approach taken in SAT planning and ILP planning is to turn a planning problem into a scheduling problem. In the resulting scheduling problem there is no distinction between action choices and ordering decisions and the decisions are made in a single homogeneous manner. To do this, the planning problem must be bounded in some way, usually by the number of discrete time steps needed.

Table 1 summarizes our assessment of four of the most promising planning approaches. Each entry in the table indicates how difficult we think it will be to extend the approach to handle the indicated issue in an effective manner. If there has already been work on an issue, we have included a reference. However, the presence of a reference does not necessarily mean that the issue has been completely or adequately addressed. For Graphplan, performance is very good. There has been some work on extending the approach to allow metric quantities [86] and continuous time [125], but this work is still preliminary and of limited scope. These extensions also seem to be complex and difficult to engineer. As a result, we believe it will be relatively hard to extend Graphlan to fully cover any of these issues. SAT planners

10. For pedagogical reasons we have taken liberties with the actual notation and algorithm used in the HSTS/RA planner.

11. Peot [109] and Geffner [22] have also argued that there are significant advantages to a functional representation.

have also exhibited very good performance. The work on LPSAT [139] shows that the framework can be extended to allow fairly general reasoning with metric quantities. We expect that multiple-capacity resources will yield to the same kinds of techniques used in IxTeT, although this has not yet been attempted. Branch-and-bound can be used to do optimization with systematic solvers, but when combined with metric quantities, optimization is not as straightforward or seamless as in an ILP framework. A big problem for both SAT and ILP planning is continuous time – a discrete time encoding will not work, and causal encodings have not proven very practical. The ILP approach is attractive because of the natural ability to handle multiple-capacity resources, metric quantities, and optimization. However, performance is not yet up to the standards of Graphplan or SAT planning techniques. CBI planning is attractive because of its ability to handle continuous time. In particular, the HSTS/RA planner [76] exhibits efficient handling of complex temporal constraints. Zeno [108] has shown that general metric quantities can be handled within the CBI framework. Zeno had serious performance limitations, but we believe that this was related to its handling of time, rather than to the use of LP techniques for handling metric quantities. Similarly, IxTeT [88] has shown that multiple capacity resources can be handled efficiently within the CBI framework. The primary difficulty for CBI techniques has been in controlling the underlying goal-directed planning search. We speculate that CBI planners need stronger heuristic guidance – the kind of guidance provided by a planning graph or by the kind of automated analysis used by Geffner for FSS planning [23].

	Multi-capacity Resources	**Metric Quantities**	**Optimization**	**Continuous Time**	**Speed**
Graphplan	hard?	hard (IPP [86])	hard?	hard (TGP [125])	good
SAT planning	moderate?	moderate (LPSAT [139])	moderate?	very hard?	good
ILP planning	easy?	easy (ILP-PLAN [83])	easy (ILP-PLAN [83])	very hard?	fair?
CBI planning	moderate (IxTeT [88])	moderate (Zeno [108])	moderate?	easy HSTS/RA[76]	fair?

Table 1: The prospects for addressing scheduling related issues in planning. Citations indicate that at least some work has been done on the topic, while “question marks” indicate our current speculation for areas that have not yet received much attention.

For many years there has been a wide gulf between planning technology and scheduling technology. Work on planning has concentrated on problems involving many levels of action choice, where the relationships between actions are varied and complex. Unfortunately, most work on classical planning has assumed a very simple, discrete model of time and has ignored issues of resources, metric quantities, and optimization. These issues are critical in most scheduling problems and, we would argue, in most realistic planning problems, such as our spacecraft problem. As a result, much of the work on classical planning does not apply to realistic scheduling problems or to our spacecraft problem.

Fortunately, there are some encouraging signs that this situation may be changing. Much of the work described in Section 4 is promising. There are still many speculative entries in Table 1, but there is hope for the ambitious spacecraft, even though much work remains to be done.

Acknowledgements

Thanks to John Bresina, Carla Gomes, Keith Golden, Dan Weld, and two anonymous reviewers for comments on earlier drafts of this paper

6 References

1. Aarts, E, and Lenstra, J. 1997. *Local Search in Combinatorial Optimization*. Wiley, 1–16.
2. Allen, J, Hendler, J, and Tate, A. 1990. *Readings in Planning*. Morgan Kaufmann.
3. Allen, J. 1984, Towards a general theory of action and time. *Artificial Intelligence* 23(2), 123–154.

4. Allen, J, and Koomen, J. 1983. Planning using a temporal world model. *Proc. 8th Int. Joint Conf. on AI*, 741–747. Reprinted in [2]

5. Anderson, C, Smith, D. and Weld, D. 1998. Conditional effects in Graphplan. *Proc. 4th Int. Conf. on AI Planning Systems* 44–53.

6. Baar, T. and Brucker, P. and Knust, S. 1998. Tabu search algorithms and lower bounds for the resource constrained project scheduling problem. In S. Voss, S. Martello, I. Osman and C. Roucairol (eds) *Meta Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, 1-18

7. Bacchus, F, and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116, 123–191.

8. Bacchus, F, and Teh, Y. 1998. Making forward chaining relevant. *Proc. 4th Int. Conf. on AI Planning Systems*, 54-61.

9. Bacchus, F, and van Run, P. 1995. Dynamic variable ordering in CSPs. *Proc. 1st Int. Conf. on Principles and Practice of Constraint Programming*, 258–275, Lecture Notes in Computer Science #976, Springer-Verlag.

10. Baker, K. 1974. *Introduction to Sequencing and Scheduling*. Wiley.

11. Baker, K. 1998. *Elements of Sequencing and Scheduling*.

12. Baptiste, P, LePape, C, and Nuijten, W. 1998. Satisfiability tests and time bound adjustments for cumulative scheduling problems. Research Report, Universitè de Technologie de Compiègne.

13. Barrett, A. 1997. *Hierarchical Task Network Planning with an Expressive Action Language*. Ph.D. dissertation, Dept. of Computer Science and Engineering, U. Washington.

14. Bayardo, R. 1996. A complexity analysis of space bounded learning algorithms for the constraint satisfaction problem. *Proc. 13th National Conf. on AI*, 298–303.

15. Beck, J, Davenport, A, Sitarski, E, and Fox, M. 1997. Texture-based heuristics for scheduling revisited. *Proc. 14th Nat. Conf. on AI*. 241–248.

16. Beck, J, and Fox, M. 1998. A generic framework for constraint-directed search and scheduling. *AI Magazine* 19(4), 101–130.

17. Bessière, C, Règin, J. 1996. MAC and combined heuristics: two reasons to forsake FC (and CBJ?). *Proc. 2nd Int. Conf. on Principles and Practices of Constraint Programming*, 61–75.

18. Blum, A, and Furst, M. 1995. Fast planning through planning graph analysis. *Proc. 14th Int. Joint Conf. on AI*, 1636–1642.

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

20. Blum, A, and Langeford, J. 1998. Probabilistic planning in the Graphplan framework. *Proc. AIPS-98 Workshop on Planning as Combinatorial Search*, 8-12.

21. Blythe, J. 1999. Decision-theoretic planning. *AI Magazine* 20(2), 37–54.

22. Bonet, B, and Geffner, H. 1999. Functional STRIPS: a more general language for planning and problem solving. *Proc. Logic-based AI Workshop*, Washington, D. C.

23. Bonet, B, and Geffner, H. 1999. Planning as heuristic search: new results. *Proc. 5th European Conf. on Planning*.

24. Boutilier, C, Dean, T, and Hanks, S. 1999. Decision theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11, 1–94.

25. Bresina, J. 1996. Heuristic-biased stochastic search. *Proc. 13th National Conf. on AI*, 271–278.

26. Brucker, P. 1998. *Scheduling Algorithms, second edition.* Springer-Verlag.

27. Carlier, J. and Pinson, E. 1989. An algorithm for solving the job shop scheduling problem. *Management Science, 35(2)*. 164-176

28. Cha, B, and Iwama, K. 1995. Performance tests of local search algorithms using new types. *Proc. 14th Int. Joint Conf. on AI*, 304–310

29. Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3), 333–377.

30. Clark, D, Frank, J, Gent, I, McIntyre, E, Tomov, N, and Walsh, T. 1996. Local search and the number of solutions. *Proc. 2nd Int. Conf. on the Principles and Practices of Constraint Programming*, 119–133.

31. Crawford, J. 1996. An approach to resource constrained project scheduling. *Proc. 1996 AI and Manufacturing Research Planning Workshop*. Albuquerque, NM, AAAI Press.

32. Crawford, J, and Baker, A. 1994. Experimental results on the application of satisfiability algorithms to scheduling problems. *Proc. 12th National Conf. on AI*, 1092–1097.

33. Dean, T, Kaelbling, L, Kirman, J, and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *AI,* 76, 35–74.

34. Dean, T, and Kambhampati, S. 1996. Planning and scheduling. In *CRC Handbook of Computer Science and Engineering*.

35. Dearden, R, and Boutilier, C. 1997. Abstraction and approximate decision theoretic planning, *Artificial Intelligence* 89, 219–283.

36. Dechter, R, and Meiri, I. 1994. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence* 68(2), 211–241.

37. Dechter, R, Meiri, I, and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49, 61-95.

38. Dixon, H, and Ginsberg, M. 2000. Combining satisfiability techniques from AI and OR. *The Knowledge Engineering Review* 15(1).

39. Drabble, B. 1993. Excalibur: a program for planning and reasoning with processes. *Artificial Intelligence* 62, 1–40.

40. Drabble, B, and Tate, A. 1994. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. *Proc. 2nd Int. Conf. on AI Planning Systems*, 243–248

41. Draper, D. Hanks, S, and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. *Proc. 2nd Int. Conf. on AI Planning Systems*, 31–36.

42. Drummond, M, Bresina, J, and Swanson, K. 1994. Just-In-Case scheduling. *Proc. 12th National Conf. on AI*, 1098–1104.

43. Ernst, M, Millstein, T, and Weld, D. 1997. Automatic SAT-compilation of planning problems. *Proc. 15th Int. Joint Conf. on AI*, 1169–1176.

44. Erol, K. 1995. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. Ph.D. dissertation, Dept. of Computer Science, U. Maryland.

45. Erol, K, Hendler, J, and Nau, D. 1994. *Semantics for Hierarchical Task Network Planning*. Technical report CS-TR-3239, Dept. of Computer Science, U. Maryland.

46. Erol, K, Nau, D, Subrahmanian, V. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence 76*, 75–88.

47. Etzioni, O, Hanks, S, Weld, D, Draper, D, Lesh, N. and Williamson, M. 1992. An approach to planning with incomplete information. *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 115–125.

48. Ferguson, G, Allen, J, and Miller, B. 1996. TRAINS-95: towards a mixed-initiative planning assistant. *Proc. 3rd Int. Conf. on AI Planning Systems*, 70–77.

49. Ford Jr, L. 1956. *Network Flow Theory.* Technical report, Rand Corporation.

50. Fox, M, 1994. ISIS: a retrospective. In [141], 3–28.

51. Frank, J. 1996. Weighting for Godot: learning heuristics for GSAT. *Proc. 13th National Conf. on AI*, 338–343.

52. Frank, J, Cheeseman, P. and Stutz, J. 1997. When gravity fails: local search topology. *Journal of Artificial Intelligence Research* 7, 249–282.

53. Freuder, E. 1978. Synthesizing constraint expressions. *Communications ACM* 21(11), 958–966.

54. Freuder, E. 1990. Complexity of K-tree structured constraint satisfaction problems. *Proc. 8th National Conf. on AI*, 4–9.

55. Freuder, E, and Elfe, C. 1996. Neighborhood inverse consistency preprocessing. *Proc. 13th Nat. Conf. on AI*, 202–208.

56. Frost, D, and Dechter, R. 1994. In search of the best constraint satisfaction search. *Proc. 12th Nat. Conf. on AI*, 301–306

57. Gazen, B, and Knoblock, C. 1997. Combining the expressivity of UCPOP with the efficiency of Graphplan. *Proc. 4th European Conf. on Planning*, 223–235.

58. Genesereth, M, and Nilsson, N. 1987. *Logical Foundations of AI*. Morgan Kaufmann.

59. Gent, I, and Walsh, T. 1993. Towards an understanding of hill-climbing procedures for SAT. *Proc. 11th National Conf. on AI*, 28–33.

60. Gent, I, McIntyre, E, Prosser, P, Smith, B, and Walsh, T. 1996. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. *Proc. 2nd Int. Conf. on the Principles and Practices of Constraint Programming*, 179-193.

61. Ghallab, M, and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. *Proc. 2nd Int. Conf. on AI Planning Systems*, 61–67.

62. Ginsberg, M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research 1*, 25–46.

63. Ginsberg, M. 1993. *Essentials of Artificial Intelligence*. Morgan Kaufmann.

64. Ginsberg, M. 1995. Approximate planning. *Artificial Intelligence 76*, 89–123.

65. Glover, F. 1989. Tabu search I. *ORSA Journal on Computing 1(3)*, 190-206.

66. Golden, K. 1998. Leap before you look: information gathering in the PUCCINI planner. *Proc. 4th Int. Conf. on AI Planning Systems*, 70–77.

67. Golden, K. and Weld, D. 1996. Representing sensing actions: the middle ground revisited. *Proc. 5th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 174–185.

68. Goldman, R. and Boddy, M. 1994. Conditional linear planning. *Proc. 2nd Int. Conf. on AI Planning Systems*, 80–85.

69. Gomes, C, and Selman, B. 1998. Boosting combinatorial search through randomization. *Proc. 15th National Conf. on AI*, 431-437.

70. Haddawy, P, and Hanks, S. 1998. Utility models for goal-directed decision-theoretic planners. *Computational Intelligence* 14(3), 392–429.

71. Haralick, R, and Elliot, G. 1980. Increasing tree search efficiency for constraint-satisfaction problems. *Artificial Intelligence* 14(3), 263–313.

72. Harvey, W, and Ginsberg, M. 1993. Limited discrepancy search. *Proc. 14th Int. Joint Conf. on AI*, 607–613.

73. Harvey, W. 1995. *Nonsystematic Backtracking Search*. Ph.D. dissertation, Dept. of Computer Science, Stanford U.

74. Jónsson, A. 1997. *Procedural Reasoning in Constraint Satisfaction*. Ph.D. dissertation, Dept. of Computer Science, Stanford U.

75. Jónsson, A, Morris, P, Muscettola, N, and Rajan, K. 1999. Next generation remote agent planner. *Proc. 5th Int. Symposium on AI, Robotics and Automation in Space*. ESTEC, Noordwijk, Netherlands, 363–370.

76. Jónsson, A, Morris, P, Muscettola, N, Rajan, K, and Smith, B. 2000. Planning in interplanetary space: theory and practice. *Proc. 5nd Int. Conf. on AI Planning & Scheduling.*

77. Joslin, D. 1996. *Passive and Active Decision Postponement in Plan Generation*. Ph.D. dissertation, Intelligent Systems Program, U. Pittsburgh.

78. Joslin, D, and Pollock, M. 1996. Is “early commitment” in plan generation ever a good idea? *Proc. 13th Nat. Conf. on AI*. 1188–1193.

79. Kambhampati, S, and Hendler, J. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55, 193–258

80. Kambhampati, S, Knoblock, C, and Yang, Q. 1995. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial order planning. *Artificial Intelligence* 76, 167–238.

81. Kautz, H, McAllester, D. and Selman, B. 1996. Encoding plans in propositional logic. *Proc. 5th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 374–384.

82. Kautz, H, and Selman, B. 1996. Pushing the envelope: planning, propositional logic, and stochastic search. *Proc. 13th Nat. Conf. on AI*. 1194–1201.

83. Kautz, H, and Walser, J. 1999. State-space planning by integer optimization. *The Knowledge Engineering Review* 15(1).

84. Kirkpatrick, S, Gelatt, C. and Vecchi, M. 1983. Optimization by simulated annealing. *Science* 220(4598).

85. Koehler, J. 1998. Planning under resource constraints. *Proc. 13th European Conf. on AI* . 489–493

86. Koehler, J, Nebel, B, Hoffmann, J. and Dimopoulos, Y. 1997. Extending planning graphs to an ADL subset. *Proc. 4th European Conf. on Planning*, 275–287.

87. Kushmerick, N, Hanks, S, and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76, 239–286.

88. Laborie, P, and Ghallab, M. 1995. Planning with sharable resource constraints. *Proc. 14th Int. Joint Conf. on AI*, 1643–1649.

89. Langley, P. 1992. Systematic and nonsystematic search strategies. *Proc. 1st Int. Conf. on AI Planning Systems* , 145-152.

90. Li, C. and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability. *Proc. 15th Int. Joint Conf. on AI*, 366–371.

91. McAllester, D, and Rosenblitt, D. 1991. Systematic nonlinear planning. *Proc. 9th National Conf. on AI*, 634–639.

92. McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine*, to appear.

93. McDermott, D. and Hendler, J. 1995. Planning: what it is, what it could be, an introduction to the special issue on planning and scheduling. *Artificial Intelligence* 76, 1–16.

94. Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8, 99–118.

95. Mackworth, A. and Freuder, E. 1985. The complexity of some polynomial network consistency algorithms for constraint satisfaction. *Artificial Intelligence* 25(1), 65-74.

96. Minton, S, Johnston, M, Phillips, A, and Laird S. 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58, 161–205.

97. Morris, P. 1993. The breakout method for escaping from local minima. *Proc. 11th National Conf. on AI*, 40–45.

98. Moscato, P. and Schaerf, A. 1997. *Local search techniques for scheduling problems: a tutorial*. Unpublished draft manuscript of ECAI-98 tutorial.

99. Muscettola, N. 1994. HSTS: integrating planning and scheduling. In [141], 169–212.

100. Muscettola, N, Nayak, P, Pell, B, and Williams, B. 1998, Remote Agent: to boldly go where no AI system has gone before, *Artificial Intelligence* 103, 5–48.

101. Nadel, B. 1989. Constraint satisfaction algorithms. *Computational Intelligence* 5, 188-224

102. Nau, D, Cao, Y, Lotem, A, and Muñoz-Avila, H. 1999. SHOP: simple hierarchical ordered planner. *Proc. 16th Int. Joint Conf. on AI*, 968–973.

103. Nau, D, Gupta, S, and Regli, W. 1995. Artificial Intelligence planning versus manufacturing-operation planning: a case study. *Proc. 14th Int. Joint Conf. on AI*, 1670–1676.

104. Nuijten, W. 1994. *Time and Resource Constrained Scheduling.* Ph.D. dissertation, U. Eindhoven.

105. Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, 324–332.

106. Penberthy, J. 1993. *Planning with Continuous Change*. Ph.D. dissertation 93-12-01, Dept. of Computer Science and Engineering, U. Washington.

107. Penberthy, J. and Weld, D. 1992. UCPOP: a sound, complete, partial order planner for ADL. *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, 103–114.

108. Penberthy, J. and Weld, D. 1994. Temporal planning with continuous change. *Proc. 12th National Conf. on AI,* 1010–1015.

109. Peot, M. 1998. *Decision-Theoretic Planning*. Ph.D. dissertation, Dept. of Engineering-Economic Systems, Stanford U.

110. Peot, M, and Smith, D. 1992. Conditional nonlinear planning, *Proc. 1st Int. Conf. on AI Planning Systems*, 189-197.

111. Peot, M, and Smith, D. 1993. Threat-removal strategies for partial-order planning, *Proc. 11th National Conf. on AI*, 492–499.

112. Pinedo, M. 1995. *Scheduling Theory, Algorithms and Systems*. Prentice Hall.

113. Pinson, E. 1995. The job shop scheduling problem: a concise survey and some recent developments. In P. Chrétienne, E. Coffman, J. Lenstra and Z. Liu (eds) *Scheduling Theory and its Applications*. Wiley.

114. Plaunt, C, Jónsson, A, and Frank, J. 1999. Satellite tele-communications scheduling as dynamic constraint satisfaction. *Proc. 5th Int. Symposium on AI, Robotics and Automation in Space*, 277–284.

115. Prosser, P. 1993. Domain filtering can degrade intelligent backjumping search. *Proc. 13th Int. Joint Conf. on AI*, 262-267.

116. Pryor, L. and Collins, G. 1996. Planning for contingencies: a decision-based approach. *Journal of Artificial Intelligence Research* 4, 287–339.

117. Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley.

118. Rich, E, and Knight, K. 1991. *Artificial Intelligence, second edition*. McGraw Hill.

119. Russell, S, and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

120. Sabin, D, and Freuder, E. 1997. Understanding and improving the MAC algorithm. *Proc. 3rd Int. Conf. on Principles and Practices of Constraint Programming*, 167-181

121. Selman, B. and Kautz, H. and Cohen, W. 1994. Noise strategies for local search. *Proc. 12th National Conf. on AI*, 337-343

122. Sadeh, N. 1994. *Micro-opportunistic Scheduling: the Micro-boss Factory Scheduler*. Technical report CMU-RI-TR-94-04, Carnegie Mellon Robotics Institute.

123. Smith, B. 1998. Trying harder to fail first. *Proc. 13th European Conf. on AI* , 249–253.

124. Smith, D, and Weld, D. 1998. Conformant Graphplan. *Proc. 15th National Conf. on AI*, 889–896.

125. Smith, D, and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. *Proc. 16th Int. Joint Conf. on AI*, 326–333.

126. Smith, S. and Cheng, C. 1993. Slack-based heuristics for constraint satisfaction scheduling. *Proc. 11th Nat. Conf. on AI*, 139–144.

127. Tate, A. 1977. Generating project networks. *Proc. 5th Int. Joint Conf. on AI*, 888–893. Reprinted in [2].

128. Tate, A, Drabble, B, and Kirby, R. 1994. O-Plan2: an open architecture for command, planning, and control. In [141], 213–239.

129. Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

130. Vere, S. 1983. Planning in time: windows and durations for activities and goals, *Pattern Analysis and Machine Intelligence* 5, 246–267. Reprinted in [2].

131. Vossen, T, Ball, M, Lotem, A, and Nau, D. 1999. On the use of integer programming models in AI planning. *The Knowledge Engineering Review* 15(1).

132. Wah, B, and Shang, Y. 1997. Discrete Lagrangian based search for solving max-SAT problems, *Proc. 15th Int. Joint Conf. on AI*, 378–383.

133. Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4), 27–61.

134. Weld, D. 1999. Recent advances in AI planning. *AI Magazine* 20(2), 93–123.

135. Weld, D, Anderson, C. and Smith, D. 1998. Extending Graphplan to handle uncertainty & sensing actions. *Proc. 15th National Conf. on AI*, 897–904.

136. Wilkins, D. 1990. Can AI planners solve practical problems? *Computational Intelligence* 6(4), 232–246.

137. Williamson, M. 1996. *A Value-directed Approach to Planning*. Ph.D. dissertation 96-06-03, Dept. of Computer Science and Engineering, U. Washington.

138. Williamson, M, and Hanks, S. 1994. Optimal planning with a goal-directed utility model. *Proc. 2nd Int. Conf. on AI Planning Systems*, 176–180.

139. Wolfman, S, and Weld, D. 1999. Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review* 15(1).

140. Yang, Q. 1990. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence* 6, 12–24.

141. Zweben, M, and Fox, M. 1994. *Intelligent Scheduling*. Morgan Kaufmann.