Planning in Continuous Adversarial Domains

Keywords: planning, AI architectures, game playing, real-time systems, simulation

Tracking Number: A356

Abstract

We designed the GRASP planner to operate in continuous, uncertain, adversarial, real-time domains. Important problems in these domains include resource allocation amongst multiple goals, determining plan operator effects, reacting to and exploiting unforeseen events, and generating workable plans quickly. We describe how GRASP combines new and old techniques to effectively handle these problems. We introduce the notion of multi-goal partial hierarchical planning and the efficient evaluation of plans using forward simulation with so-called *critical points*. The paper concludes with an example of how GRASP was applied to the problem of Course of Action generation and evaluation.

Introduction

Continuous and adversarial domains pose particularly challenging problems for today's planners. An adversary confounds the problem of having uncertain information about the state of the world or the effects of operators: An uncertain domain may introduce randomness into the world states that can result from an application of an operator, whereas an adversary will actively try to steer the world into a state that is undesirable. Moreover, since an effective adversary is one who can surprise us, this new state may be very hard to predict.

Adversarial search algorithms assume that the world jumps from one state to another as the the two opponents make their moves. The paradigmatic example is game tree search. It is difficult, however, to fit continuous domains into this mold. Continuous domains are characterized by processes that change continuously over time. By forcing a rigid and often arbitrary set of states upon the world, the planner loses access to the dynamics of these processes, instead of exploiting them.

Furthermore, the number of states greatly affects the tractability of generating a solution to a given problem. If the operators are too primitive, and correspondingly the plan space large, the solution to a given problem will involve a deeper search through the space than if the state space were smaller. If the operators become too abstract, however, important features of the domain may be lost, resulting in a solution that is not useful. In order for a game-tree to be feasible in a continuous domain, the set of operators that is applicable in a given state has to reduced drastically for the branching factor to be reasonable.

Other problems that are particularly pronounced in adversarial planning are the ability to achieve multiple concurrent goals, react gracefully to plan failure, and generate a solution in real-time.

In this paper, we describe how our planner, GRASP (General Reasoning using AbStract Physics), addresses the above concerns:

- Plans are not generated from atomic planning operators at run-time; instead, we view plans as general solution skeletons that have been distilled from previous experiences. This reduces the combinatorics of the planning problem to a feasible level.
- We avoid the problem of having to pre-specify plan post-conditions by using a simulator to establish the world state after a plan has executed.
- State boundaries are not specified by a priori, but are *created dynamically* as plans are executed (or simulated). So-called *critical points* mark the state boundaries.
- The planner operates at a fairly high level and does not plan out every detail, which improves planner performance and reduces the combinatorics. Plan operators are assumed to be competent; they can cope with some unforeseen events.
- The planner is integrated into a general agent control architecture, HAC (Hierarchical Agent Control) that can deal with plan failures, unexpected opportunities, and resource conflicts.

The Capture The Flag Testbed

We have been developing a dynamic and adversarial domain in which to test GRASP. This domain is based on the game of "Capture the Flag" (CtF). In CtF (see Figure 1) there are two teams; each has a number of movable units and flags to protect. They operate on a map which has different types of terrain. Terrain influences movement speed and forms barriers; terrain also affects unit visibility. A team wins when it captures all its opponent's flags. This game appears deceptively simple. The player must allocate forces for attack and defense, and decide which of the opponent's units or flags he should attack. The player must react to plans that do not unfold as expected, and possibly retreat or regroup. We model limited visibility and inaccurate sensor data. This leads to additional strategies involving feints, sneak attacks, and ambushes.

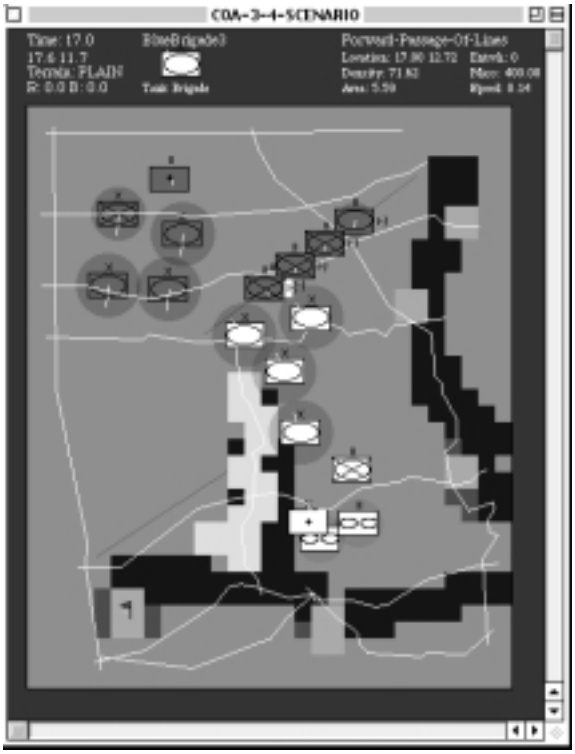

Figure 1: The Capture the Flag domain.

HAC: Hierarchical Agent Control

As we will see, GRASP takes full advantage of our agent control architecture, HAC. HAC can be viewed as a language for writing agent actions. HAC takes care of the mechanics of executing the code that controls an agent, passing messages between actions, coordinating multiple agents, arbitrating resource conflicts between agents, updating sensor values, and interleaving cognitive processes such as planning.

HAC organizes the agent's actions in a hierarchy (see Figure 2). As one goes up the hierarchy, actions become increasingly abstract and powerful. They solve more difficult problems, such as path planning, and can react to wide range of eventualities. Although actions lower in the hierarchy will tend to be more reactive, whereas those higher up tend to be more deliberative, the transition between them is smooth and completely up to the designer. Unlike other architectures, we do not prescribe a preset number of behavioral levels (Georgeff & Lansky 1987; Cohen *et al.* 1989).

A hierarchy of sensors parallels the action hierarchy. Just as a more complex action uses simpler ones to accomplish its goal, complex sensors use the values of simpler ones. These are *abstract sensors*. They are not physical, since they do not sense anything directly from the world. They take the output of other sensors and integrate and re-interpret it. Abstract sensors are used throughout HAC and GRASP to notify actions and plans of unexpected or unpredictable events.

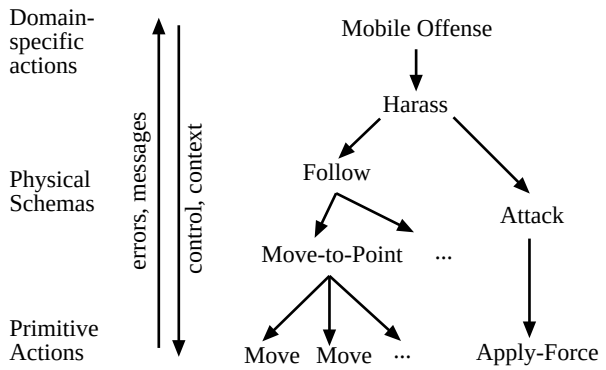

Figure 2: Actions form a hierarchy; control information is passed down, messages are passed up. The lowest level are agent effectors; the middle layer consists of more complex, yet domain-general actions called *physical schemas* (Atkin *et al.* 1998). Above this level we have domain-specific actions.

HAC executes actions by scheduling them on a queue. The queue is sorted by the time at which the action will execute. Actions get taken off the queue and executed until there are no more actions that are scheduled to run at this time step. Actions can reschedule themselves, but in most cases, they will be rescheduled when woken up by messages from their children.

HAC is a *supervenient* architecture (Spector & Hendler 1994). It abides by the principle that higher levels should provide goals and context for the lower levels, and lower levels provide sensory reports and messages to the higher levels ("goals down, knowledge up"). A higher level cannot overrule the sensory information provided by a lower level, nor can a lower level interfere with the control of a higher level. Supervenience structures the abstraction process; it allows us to build modular, reusable actions. HAC simplifies this process further by enforcing that every action's implementation take the following form:

1. React to messages coming in from children.
2. Update state.
3. Schedule new child actions if necessary.
4. Send messages up to parent.

Figure 2 shows a small part of an action hierarchy. The **follow** action, for example, relies on a **move-to-point** action to reach a specified location. **Move-to-point** will send status reports to **follow** if necessary; at the very least a completion message (failure or success). The only responsibility of the **follow** action is to issue a new target location if the agent being followed moves. HAC is an architecture; other than enforcing a general form, it does not place any constraints on how actions are implemented. Every action can choose what messages it will respond to. Actions can be deliberative or reactive. Parents can run in a parallel with their children or only when the child completes.

An action or a plan posts a set of goals $G = \{g_1, g_2, ...g_n\}$. This invokes the following process:

1. For every g_i:
   1.1 Search the list of plans for those that can satisfy g_i.
   1.2. Evaluate each potential plan's pre-conditions and only keep only those whose pre-conditions match.
   1.3. For each remaining plan, estimate it's required resources.
2. Sort G by the priority of g_i.
3. *candidate_plan_sets* := nil.
4. Loop over g_i in order of priority:
   4.1 If only one plan achieves g_i, instantiate it (bind unbound variables) and add it to every plan set in *candidate_plan_sets*; otherwise:
   4.2 If several plans achieve g_i, score each one based on:
   - how many resources it uses
   - how many other goals in G it (partially) satisfies
   - other plan-specific heuristics

   4.3 Choose m (m is rarely > 1 to limit combinatorics) of the highest scoring plans: $p_1, ..., p_m$
   4.4 Loop over remaining $g_j (j > i)$: if g_i partially satisfies g_j, merge g_j into $p_1, ..., p_m$
   4.5 Copy the plan sets in *candidate_plan_sets* m times; add p_k to copy k.
5. Loop over *plan_set* in *candidate_plan_sets*:
   5.1 Evaluate *plan_set* using forward simulation.
6. Execute the plan set (make them child actions of the goal poster) that in simulation, results in a world state with the highest score.

Figure 3: The planning algorithm.

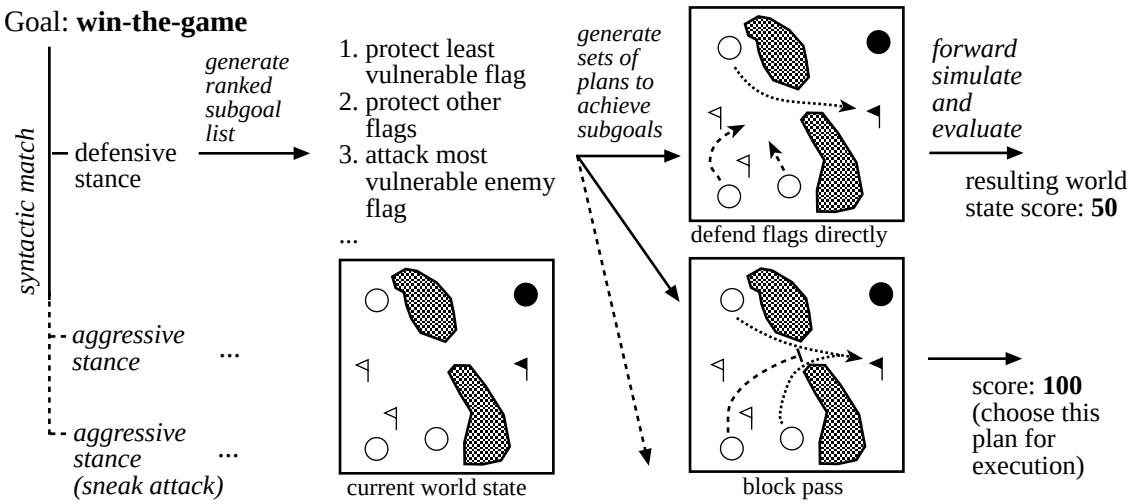

Figure 4: A planning example: White is trying to satisfy the goal **win-the-game**. Several top-level plans match this goal; the example explores what happens when **defensive-stance** is expanded. This plan emphasizes defense, which is reflected in the list of subgoals generated. There are several sets of plans that achieve these subgoals, and many ways to allocate resources to these plans. The planner uses heuristics to prune this set. In the first case, two units are allocated to flag defense, and one is sent out to attack. In the second case, only one unit is needed to block the mountain pass, thus protecting the flags, leaving two units for the attack. This plan set is more likely to succeed and is ranked higher.

Multi-goal Partial Hierarchical Planning

A number of difficult problems face any planner operating in a real-time, continuous, uncertain, and adversarial domain:

- In the presence of multiple competing goals, how are resources allocated to plans?
- How are plan post-conditions determined if the domain is inherently unpredictable?
- How do plans react to unexpected pitfalls or opportunities?
- How can plans be generated quickly, under real-time pressure?

GRASP integrates new and established techniques to deal with these problems. GRASP is a least-

commitment partial hierarchical planner (Georgeff & Lansky 1986). By having a set of pre-compiled skeletal solutions, we avoid the enormous branching factor a generative planner would face in this domain. GRASP further reduces the combinatorics by not planning for every eventuality. Plans are built within the HAC framework, using operators that are assumed to be flexible and competent.

Multiple Goals

GRASP extends the traditional partial hierarchical planning framework by allowing multiple goals to be associated with a resource or set of resources. These are not simply conjunctive goals; instead, goals are prioritized. GRASP uses heuristics in order to achieve the largest set of high priority goals possible.

In CtF, winning involves coordinating multiple subgoals: protecting your own flags, thwarting enemy offensives, choosing the most vulnerable enemy flag for a counter-attack, and so on. Each requires resources (units) to be accomplished. Sometimes one resource can be used to achieve several tasks. For instance, if two flags are close together, one unit might protect both. Or, advancing towards an opponent's flag might also force the opponent to retreat, thus relieving some pressure on one's own flags.

Plans are part of the HAC action hierarchy. Plans can be viewed as actions that explicitly state the goal they achieve. Every plan must have associated with it a set of functions to assist in the resolution of multiple goals:

- *pre-condition(plan)*: is the plan applicable in the current situation?
- *estimate-resources(plan)*: what resources is this plan likely to need?
- *goal-congruence(planA, planB)*: to what degree do plans A and B achieve the same goal?
- *merge-plans(planA, planB)*: create a new plan that achieves both of plan A and B's goals.

GRASP uses these functions and the algorithm outlined in Figure 3 to solve the resource allocation problem in the presence of multiple goals. Figure 4 shows an example of the plan generation procedure. Each goal is prioritized, then plans are generated to achieve each one. Heuristics are used to generate a small number of possible plan sets. If resource problems arise *during* a plan's execution (because a resource was destroyed and the plan using it cannot succeed without it, for example), a resource error message is sent to the plan initiator using the HAC messaging mechanism, possibly causing resources to be re-assigned or a complete replan to take place.

Plan Evaluation Using Critical Points

When several plans apply, partial hierarchical planners typically select one according to heuristic criteria. GRASP instead performs a qualitative simulation on each candidate plan (or plan set). Potential

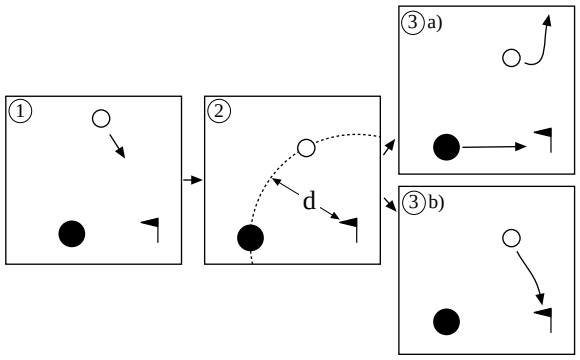

Figure 5: An example for a critical point while executing an attack action.

plans are simulated forward, then a static evaluation function is applied to select the best plan. The static evaluation function incorporates such factors as relative strength and the number of captured and threatened flags of both teams to describe how desirable the resulting world state is. Simulation helps alleviate the problem of not being able to specify exact post-conditions for every plan operator. Uncertainty in the world can be addressed by Monte Carlo analysis. The world and your opponent(s) are simply another set of processes to simulate.

The downside is that simulation is a costly operation. In order to do it efficiently and thus be able to evaluate plans quickly, GRASP evaluates plans at a level that is more abstract than the domain being operated in. This is much in keeping with Minsky's original conception of planning (Minsky 1961). GRASP ignores certain details of the domain, such as obstacle avoidance, during plan evaluation. More importantly, GRASP attempts to identify the time periods during which *no important interactions between agents are likely to occur* and skips over them.

The problem that GRASP faces is having to impose "states" on a continuous domain. The world moves into a new state if and only if an event takes places that might affect the outcome of the plan evaluation process. GRASP defines states boundaries using *critical points*, which are established dynamically, as the plan simulation unfolds. A critical point is a time during the execution of an action or plan where a decision might be made. If this decision can be made at any time during an interval, it is the *latest* such time.

Simple actions, such as moving from point A to point B, only have one critical point: the time at which the action completes. This is the time at which a new decision has to be made about what to do with the unit that was moving. The critical time can easily be estimated given the terrain and the unit's typical movement speed. More complicated actions have larger critical point sets. The attack action depicted in Figure 5 makes a decision during its execution: it will abandon the attack if the

1. Add all plans in the plan set P to all the actions currently ongoing in the simulator.
2. In simulation, loop either until a fixed time in the future or until too many errors have accumulated in the simulation:
   2.1 Compute the minimum critical time t of all actions being simulated.
   2.2 Advance all actions by t time units.
3. Evaluate the resulting world state; return this value as the score for the plan set P.

Figure 6: The plan evaluation algorithm.

thing being attacked is protected by a unit larger than the attacker. In this example, a white unit is attacking a black flag and there is a large black unit nearby. The critical point is the time at which the white unit is closer to the flag than the black unit is now. This is the latest point in time at which Black could interfere with the attack action. If Black has started moving to the flag by this time, White will abandon the attack. If Black has remained stationary or gone somewhere else, the attack will be successful and the flag will be captured.

Every simulatable action (and plan) must have two functions associated with it. The first computes the next critical time for the action. The second, **(advance** t**)**, takes as an argument a time parameter t and will change the world state to reflect the execution of this action t time units into the future. These functions are currently written by the designer of the action. Critical point estimations are local to the action; they are based on what this action is likely to do based on the *current state* of the world and predictions that can be made from it.

Using GRASP for COA Analysis

GRASP and HAC were used to evaluate abstract plans called Courses of Action (COA's) in the High Performance Knowledge Base project (HPKB). We used our simulator to model a military domain and simulated the possible outcomes of a COA using Monte Carlo analysis. The initial conditions of a scenario were then varied slightly and the effect on the overall outcome of the scenario was measured.

We used GRASP to fill in gaps in a COA that had been sketched by a human planner. An underspecified COA fails to provide actions and goals for all the units under their control, which can happen when an engagement goes in a direction that was not predicted. Adding the planner serves two purposes: creating more realistic opponents and aiding human planners. A COA must specify both what the friendly units are told to do and what the enemy units are likely to do. Rather than being forced to create plans for both sides, it is much more desirable to use the planner to create a reactive and intelligent opponent who can truly test the strength of the COA.

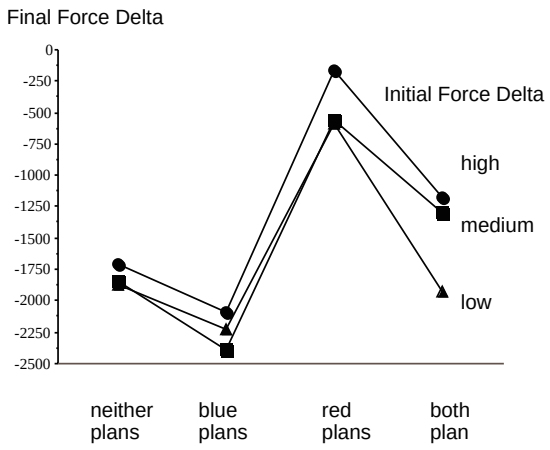

Figure 7: The effect of planning condition and initial force delta on final force delta.

Our goal was to evaluate scripted engagements between Red and Blue forces. We compared four conditions: the COA as specified, the COA plus the Blue planner, the COA plus the Red planner, and the COA plus the both Red and Blue planners.

We ran the simulation 100 times, collecting final Red and Blue mass (an abstract measure of unit strength), the total time of the simulation and the end result. A simulation ended either when one team won or after 300 time units (12 simulated hours). Each simulation randomized the positions and masses of the units on each side. The basic composition of the forces and the tasks specified in the COA were the same in every simulation.

As expected, running either planner alone greatly improves matters for the side that is planning and running both planners moved the averages towards the middle (Figure 7). We found that even in the "no planning" condition, variance in outcome (e.g., final force delta: Red mass minus Blue mass at the end of a trial) was very large, driving home the point that in domains such as this, small differences in the start state can greatly affect the end state. The initial force delta accounts for only 15% of the final force delta. Out of all possible factors, the ability to plan was the best predictor of success.

On the surface this is not a surprising result. It is, however, a qualitative measure of GRASP's ability to produce workable plans in a continuous complex domain under real-time pressure. Furthermore, these plans have the same level of quality that plans designed by human planners have: using the planner, we were able to confirm that one variant of the scenario labeled "Red Most Dangerous" was in fact the most dangerous variant for Blue, and that certain events will lead to Blue's defeat, for example that a Blue counterattack must begin before a certain time.

Contributions and Related Work

The GRASP planner integrates a number of new and old ideas to deal with continuous and adversarial domains in real-time. GRASP builds upon the partial hierarchical planners used in RESUN (Carver & Lesser 1993), PHOENIX (Cohen *et al.* 1989) and the data analysis system AIDE (St. Amant 1996). GRASP is unique in that it extends the partial hierarchical planning framework by explicitly representing multiple goals and integrating the planner into an action hierarchy that handles resource arbitration and failure recovery. This hierarchy, implemented in HAC, allows us to plan with operators that are flexible and competent. The HPKB COA evaluation experiment provides a qualitative demonstration of GRASP's ability to generate good and timely plans in a realistic application.

Others have used simulation to evaluate (Lee & Fishwick 1994) or test plans (Beetz & McDermott 1994; Hammond 1990; Lesh, Martin, & Allen 1998). To the best of our knowledge, the idea of combining simulation with critical points in order to improve its efficiency is novel. Not all planning approaches represent state in the same way, and there is indeed an entire subfield of planning that seeks to reason about continuously changing processes (e.g. (Dean & Wellman 1991; Penberthy & Weld 1994)).

Critical points themselves are well known in Qualitative Physics (Weld & deKleer 1989; Forbus 1984). Roboticists, in particular those dealing with motion planning (Canny 1988; Latombe 1991), have long had to face the problem of continuous search spaces. In both these fields, however, states are defined a priori, before the reasoning or search algorithms begin operation. We, on the other hand, induce the state space dynamically based on information that is local to every action.

Acknowledgments

We wish to thank General Charley Otstott for his expertise and enthusiastic advocacy, as well as everyone at Alphatech, particularly Eric Jones, for helping run the COA evaluation.

This research is supported by DARPA/USAF under contract numbers N66001-96-C-8504, F30602-97-1-0289, and F30602-95-1-0021. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Defense Advanced Research Projects Agency/Air Force Materiel Command or the U.S. Government.

References

Atkin, M. S.; Westbrook, D. L.; Cohen, P. R.; and Jorstad, G. D. 1998. AFS and HAC: Domain-general agent simulation and control. In *Working Notes of the Workshop on Software Tools for Developing Agents, AAAI-98*, 89–95.

Beetz, M., and McDermott, D. 1994. Improving robot plans during their execution. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, 7–12.

Canny, J. F. 1988. *The Complexity of Robot Motion Planning.* Cambridge, Massachusetts: MIT Press.

Carver, N., and Lesser, V. 1993. A planner for the control of problem solving systems. *IEEE Transactions on Systems, Man, and Cybernetics, special issue on Planning, Scheduling, and Control* 23(6):1519–1536.

Cohen, P. R.; Greenberg, M. L.; Hart, D. M.; and Howe, A. E. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3):32–48.

Dean, T., and Wellman, M. 1991. *Planning and Control.* Morgan Kaufmann.

Forbus, K. D. 1984. Qualitative process theory. *Artificial Intelligence* 24:85–168.

Georgeff, M. P., and Lansky, A. L. 1986. Procedural knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation* 74(10):1383–1398.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 677–682. MIT Press.

Hammond, K. J. 1990. Explaining and repairing plans that fail. *Artificial Intelligence Journal* 45:173–228.

Latombe, J.-C. 1991. *Robot Motion Planning.* Dordrecht, The Netherlands: Kluwer.

Lee, J., and Fishwick, P. A. 1994. Simulation-based planning for computer generated forces. *Simulation* 63(5):299–315.

Lesh, N.; Martin, N.; and Allen, J. 1998. Improving big plans. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 860–867. AAAI Press.

Minsky, M. 1961. Steps towards artificial intelligence. *Proceedings of the I.R.E.* 49:8–30.

Penberthy, J., and Weld, D. S. 1994. Temporal planning with continuous change. In *Proceedings of the Twelfth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI/MIT Press.

Spector, L., and Hendler, J. 1994. The use of supervenience in dynamic-world planning. In Hammond, K., ed., *Proceedings of The Second International Conference on Artificial Intelligence Planning Systems*, 158–163.

St. Amant, R. 1996. *A Mixed-Initiative Planning Approach to Exploratory Data Analysis.* Ph.D. Dissertation, University of Massachusetts, Amherst.

Weld, D., and deKleer, J. 1989. *Readings in Qualitative Reasoning about Physical Systems.* Los Altos, CA: Morgan Kaufmann Publishers.