Planning in Dynamic Environments: The DIPART System

Martha E. Pollack[*]
Department of Computer Science
and Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260
pollack@cs.pitt.edu

Abstract

Many current and potential AI applications are intended to operate in dynamic environments, including those with multiple agents. As a result, standard AI plan-generation technology must be augmented with mechanisms for managing changing information, for focusing attention when multiple events occur, and for coordinating with other planning processes. The DIPART testbed (Distributed, Interactive Planner's Assistant for Real-time Transportation planning) was developed to serve as an experimental platform for analyzing a variety of such mechanisms. In this paper, we present an overview both of the DIPART system and of some of the methods for planning in dynamic environments that we have been investigating using DIPART. Many of these methods derive from theoretical work in real-time AI and in related fields, such as real-time operating systems.

Introduction

Many current and potential AI applications are intended to operate in dynamic environments, including those with multiple agents. An important example is crisis action planning, which is typically a distributed process, involving multiple planners each tasked with forming plans to meet some subset of the overall mission objectives. During planning, changes that occur in the world can affect the quality of the plans being created. When planning and execution are interleaved, as they often must be in crisis situations, changes can also affect the quality of plans whose execution has already begun. To operate in such environments, standard AI plan-generation technology must be augmented with mechanisms for managing changing information, for focusing attention when multiple events occur, and for coordinating with other planning processes. In the DIPART project, we have been concerned with the development and analysis of such techniques. Many of the techniques we have explored derive from theoretical work in real-time AI and in related fields, such as real-time operating systems.

To support our research on plan generation in dynamic, multi-agent environments, we built DIPART—the Distributed, Interactive Planner's Assistant for Real-time Transportation planning. DIPART is a prototype simulation system that includes a network of agents, each of which assists a human planner, and a simulated dynamic environment, which implements the Pacifica NEO scenario (Reece *et al.* 1993). In this paper, we present an overview both of the DIPART system and of some of the methods for planning in dynamic environments that we have been investigating using DIPART. Due to space limitations, we can only provide a brief introduction to each piece of research; however, we include pointers to more detailed references throughout this paper.

The DIPART System

System Overview

The DIPART system consists of a network of communicating nodes each assisting a human planner, plus a simulated environment. The underlying idea is that each planner has responsibility for forming and overseeing the execution of some set of plans that are carried out in the (simulated) environment. Each planner may have only a restricted view of the environment and of the activities of the other planners; although cooperation among the planners may be desirable, it is not automatic. Figure 1 illustrates the overall system architecture, highlighting the internal architecture of a single node. Because each node performs the role of an intelligent assistant, we sometimes refer to the nodes as "agent processes".

The internal architecture of each DIPART node is based on a generic model of process scheduling, similar

[*]This work has received primary support from the Rome Laboratory (RL) of the Air Force Material Command and the Defense Advanced Research Projects Agency (Contract F30602-93-C-0038) as part of the ARPA Planning Initiative (ARPI). Additional support has come from the Air Force Office of Scientific Research (Contracts F49620-91-C-0005 and F49620-92-J-0422), an NSF Young Investigator's Award (IRI-9258392), and the Office of Naval Research (N00014-95-1-1161). Members of the DIPART development team have included Taieb Znati, Eithan Ephrati, David Joslin, Sylvain Lauzac, Marina Milshtein, Arthur Nunes, Nilufer Onder, Yagil Ronen, and Sigalit Ur.

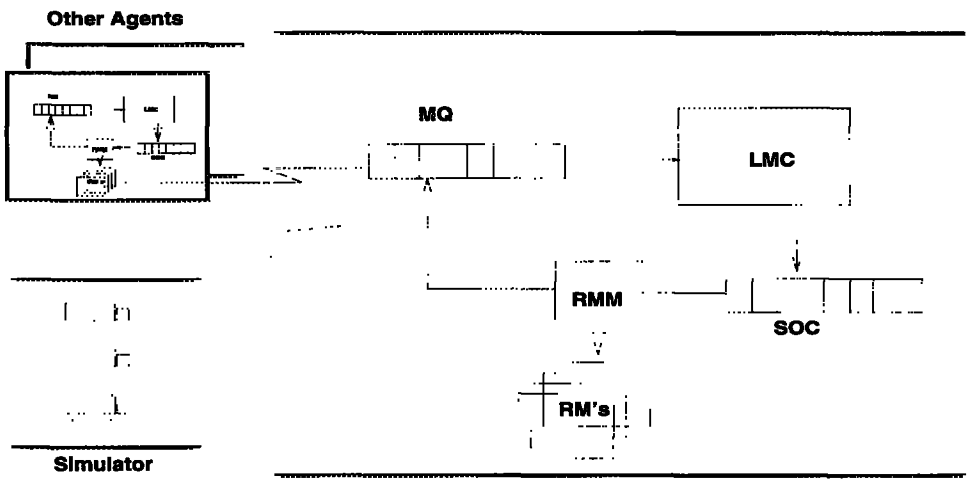

Figure 1: DIPART Architecture

to those found in the literature on operating systems (Tannenbaum 1992). Incoming messages are stored on a Message Queue (MQ), and indicate events that may require attention. Often the value of responding to a particular message is time dependent. Thus, a mechanism is needed to determine what processes should be invoked in response to each message, and to schedule the selected processes. In our model, the module that makes these decisions is the Locus of Meta-Level Control (LMC); it is responsible for invoking various (object-level) processes, which we call Reasoning Modules (RMs). The RMs include a resource estimator, which estimates the amount of computational and external resources required for a given task, a planner, which computes plans to achieve specified goals, and an execution monitor, which tracks the performance of plans.

As shown in the figure, the LMC performs its task by posting entries in a Schedule of Computation (SOC). These entries include information about which RM to invoke, the input to that process, the invocation deadline (i.e., the time after which the system should no longer bother to invoke the process), and, in some circumstances, the amount of time to allocate to the process in question. A process controller, called the Reasoning Modules Manage (RMM) reads entries from the SOC and then invokes the appropriate process. Individual processes may also generate messages if follow-on computation is needed. A global database stores information that can be used by both the LMC and the object-level processes.

In addition to the agent nodes, DIPART includes a simulator which has been tailored to Pacifica NEO scenario, described below. It runs as a separate process in the overall DIPART system. It represents the "actual" state of the world; in contrast, the models of the world kept by individuals agents may be limited or may become out-of-date, as they are intended to represent the views that the agents currently have, given the information they have so far received. The simulator is designed to allow modeling of resource allocation to agents.

The DIPART system has been implemented on DECStation 5000 workstations, under Ultrix 4.3, using Allegro Common Lisp and the Garnet interface-development system. Each of the agent nodes runs on its own processor, as does the simulated environment. Within each node, the LMC runs in one thread, and the RMs in another. A communication package based on UDP has also been implemented to support inter-node communication.

The Pacifica Scenario

To ground our research, we employ the Pacifica NEO scenario, developed by Reece and Tate for the RL/ARPA Planning Initiative as part of the PRECiS environment (Reece *et al.* 1993). This scenario involves the fictional island nation of Pacifica, on which a number of U.S. citizens are located. The island has various natural and man-made features, including cities, an airport, bridges, roads, and a volcano. Because of an expected uprising, the citizens need to be evacuated. For this, they must first be brought by truck to the capital city, where the airport is located. Evacuation can be complicated by unexpected road or bridge closings, either as a result of natural forces, e.g., a volcano, or hostile human forces; it can also be complicated by the fact that the citizens may be scattered around the island, and must themselves get to major cities before being taken by truck to the capital.

We assume that the NEO is to be planned and overseen by several human planners (typically, we run DIPART with between 2 and 6 planning nodes). Each human planner is responsible for a different component of the operation; although the task may be divided in various ways, we generally assign each planner the task of moving citizens from one city to the capital. The exact number of citizens and their current location may not be fully known to each planner. Each human planner is assisted by a DIPART node; the human submits goals to the node, and can query the node for current status information. The nodes are then responsible for forming plans to satisfy the user's goals, for coordinating communication with other planners, and for alerting the user to reports from agents in the (simulated) world.

Approaches to Planning

The key task performed by DIPART nodes is plan generation: human users input goals, such as evacuating a certain number of citizens from some city, and the DIPART node generates, dispatches, and monitors the execution of a plan to carry out that goal. Consequently, a central focus of our research has concerned the development of efficient planning algorithms.

Control during Planning

Many current state-of-the-art planners make use of partial-order causal link (POCL) algorithms (McAllester & Rosenblitt 1991; Penberthy & Weld 1992). POCL planning involves searching through a space of partial plans, where the successors of a node representing partial plan P are defined to be the refinments of P. As with any search process, POCL planning requires effective control; in POCL planning, search control has two components. The first, *node selection,* involves choosing which partial plan to refine next. Most POCL algorithms use best-first search to perform node selection. Once a partial plan has been selected, the planner must then perform *flaw selection,* which involves choosing either a threat to resolve (typically, by promotion, demotion, or separation) or an open condition to establish (by adding a new step to the plan or adding a new causal link to an existing step). Unless it is impossible to repair the selected flaw, new nodes representing the possible repairs are added to the search space.

In (Joslin & Pollack 1994), we explored a flaw-selection strategy, the Least-Cost Flaw Repair (LCFR) strategy, which can be seen as a generalization of the DUnf strategy that had been proposed by Peot and Smith (1993). In LCFR, we define the *repair cost* of any flaw—either threat or open condition—to be the number of nodes generated as possible repairs. LCFR is the strategy of always selecting a flaw with the lowest possible repair cost at a given node. LCFR will delay any threat that is unforced (repair cost > 1) in favor of a threat that is forced (repair cost $<= 1$.) By treating all flaws uniformly, LCFR also applies a similar strategy to open conditions, preferring to handle open conditions that are forced over open conditions, or threats, that are not. Similarly, LCFR handles the case in which all that remain are unforced threats: the LCFR strategy will select a threat with minimal repair cost.

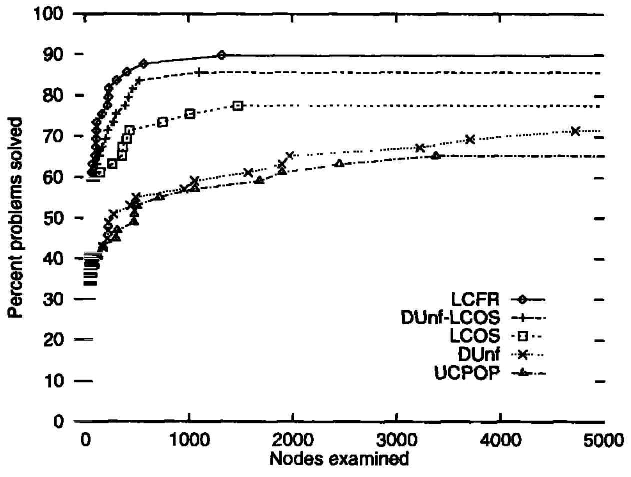

Figure 2: Comparison of planner search spaces

Our experimental assessment of LCFR demonstrated that the power of DUnf does not come from delaying threat repairs *per se*, but rather from that fact that this delay has the effect of imposing a partial preference for least-cost flaw selection. Our experiments also showed that extending this to a complete preference for least-cost selection, as in LCFR, reduces search-space size even further. Details of the experiments can be found in (Joslin & Pollack 1994). Here we simply present the results of a key experiment, in which we compared 5 search strategies on 49 test problems from a variety of domains. Figure 2 plots the percentage of test problems solved by each planner with a fixed number of nodes examined. (Each point $\langle x, y\rangle$ denotes that x% of the 49 test problems were solved by examining no more than y nodes.) As can be seen, the LCFR-based planner outperforms any of the others, including the two based on Peot and Smith's DUnf strategy.

As might be expected, the benefit of the LCFR strategy is not without a cost: specifically, performing least-cost flaw selection can incur a significant computational overhead. We therefore developed QLCFR, which reduces this overhead by approximating repair costs, and we demonstrated its effectiveness experimentally. Again, complete details can be found in (Joslin & Pollack 1994); subsequent work that builds on the LCFR approach includes (Srinivasan & Howe 1995; Schubert & Gerevini 1995).

Cost-Directed Planning

The LCFR strategy described above is quite effective for planning problems in which alternative solutions to a planning problem are considered to be roughly equal—an assumption that is, in fact, made in much of the plan generation literature. In many domains, however, this assumption is not warranted: for any given planning problem, some solutions have lower execution cost, some are more likely to succeed, and so on. To handle such cases, we developed a "cost-directed" heuristic planner, which is capable of finding low-cost plans.[1] The algorithm performs POCL planning, using an A^* strategy for node selection. The heuristic evaluation function is computed by a deep lookahead that calculates the cost of complete plans for a set of predefined top-level subgoals, under the (generally false) assumption that those subgoals do not interact. In our work so far, we have assumed that flaw selection is performed randomly, leaving it to future work to explore the question of which flaw selection strategies can best be integrated into our approach.

In (Ephrati, Pollack, & Milshtein 1996), we show that the cost-directed planning algorithm not only leads to finding lower-cost plans, but in many circumstances it does this without negatively impacting the efficiency of planning: in fact, it can lead to a significant decrease in total planning time. This result is due in part to the fact that generating plans for a set of independent subgoals is exponentially less costly than generating a complete plan taking interactions into account(Korf 1987). At least in the limit, the cost of forming plans for subgoals treated independently does not significantly effect the computational complexity of the complete planning problem. Moreover, while focusing on lower-cost plans, the heuristic function effectively prunes the search space. Thus, the use of the deep evaluation in node selection can outweigh the marginal additional complexity. Our experiments demonstrate that the advantages of cost-directed planning increase with the complexity of the planning problem, where this is measured in terms of the amount of subgoal interdependence, the heterogeneity of the cost of actions, the average branching factor, and the number of subgoals and length of the minimal-cost plan.

Constraint-Based Planning

Both of the strategies for controlling planning described above build directly on a traditional POCL style of planning. To a large extent, POCL planning was originally motivated by an observation of the advantages of taking a "least-commitment" approach to planning. Least-commitment planning involves postponing decisions until constraints force them to be made. Any decision made when it is not forced is an "early commitment." POCL planning, as opposed to the earlier, state-spaced planning, made it possible to take a least-commitment approach to some decisions, particularly to the ordering of plan steps. However, POCL planners continue to rely to some degree on early commitments for other decisions, including variable binding, threat resolution, and choice of an operator to satisfy open conditions.

Because the least-commitment approach has, by and large, been successful where it has been tried, an obvious question is whether the least-commitment approach should be applied to *every* planning decision; in other words, is early commitment ever a good idea? An obstacle to addressing this question experimentally arises from the way in which POCL planners manage decision-making. They take what we call a *passive postponement* approach, choosing one decision at a time to focus on, and keeping all the other, postponed decisions (about how to achieve certain goals and how to resolve threats) on an "agenda," where they play no role in the plan generation process until they are selected for consideration. The items on the agenda may in fact impose constraints on the plan being generated, but these constraints are not available to the planning algorithm so long as the items remain on the agenda. The fact that constraints exist but are not always accessible makes it difficult if not impossible for a POCL planner to be made more "least commitment". Postponing decisions until they are forced implies being able to recognize whether any decision is forced, and this in turn implies that all the constraints that might affect a decision must be available to (and must be used by) the planning algorithm.

In response to these difficulties, we developed a new approach to planning, called *active postponement*, in which even postponed decisions play a role by constraining the plan being generated. This technique has been implemented in the Descartes system. The key idea in Descartes is to transform planning problems into Constraint Satisfaction Problems (CSPs) which can then be solved by applying both planning and CSP techniques. In general, a planning problem cannot be transformed into a single static CSP, however; instead it must be transformed into a *dynamic* CSP to which new constraints and variables can be added during the solution process. The dynamic CSP is then solved by breaking it down into static CSPs, to which standard CSP techniques may be applied.

As with the approaches discussed above, we have conducted a number of experiments to explore the power of constraint-based planning. These experiments demonstrate that passive postponement—even "smart" passive postponement, using a selection strategy like LCFR—can result in significant performance penalties. Further experiments show that it is worthwhile to extend the least-commitment approach much further than has been done in prior work. These results also suggest, however, that there are some fundamental limits to the effectiveness of the least-commitment approach, and that sometimes early commitments can increase planning efficiency. We have proposed a principled approach to deciding when to make early commitments in planning, based on an analysis of the on-

[1]More generally, the algorithm could be applied to find plans that satisfy any measure of quality, but for simplicity we equated the quality of any plan with the sum of the costs of the actions it includes.

going constraint processing: specifically, early commitment is needed when the planning process is forced to make what we call unrestricted expansions. Details of the constraint-based planning approach, its implementation in Descartes, and the results of experiments using Descartes can be found in (Joslin & Pollack 1995; 1996; Joslin 1996).

Meta-Level Control of Reasoning

As noted in the Introduction, planning in DIPART occurs in a dynamic environment; often, one planning problem will have to be interrupted so that attention can be given to another planning problem. A central focus of the DIPART project has thus been the development and assessment of alternative strategies for meta-level reasoning, i.e. deciding how to allocate computational resources. Within the DIPART system this task is performed by the LMC. The LMC must decide what to do from messages that can arrive from four different sources:

1. the human user, who posts a new goal to the system or tells the system a new fact.
2. other nodes, which may be seeking information, or may have information to share, or may have goals that they would prefer to be handled by someone else.
3. agents situated in the simulated world, who may transmit a message to their supervising agent (i.e. DIPART node) to report an unexpected change in the environment.
4. reasoning modules within the node itself, which post messages identifying information about tasks that are in need of further processing by other RMs.

The problem of allocating reasoning resources is sometimes called the *deliberation-scheduling problem*. Previous approaches to deliberation scheduling in AI include the use of off-line allocation of on-line deliberation time for tasks with known computational demands (Greenwald & Dean 1994; Boddy & Dean 1989; Zilberstein & Russell 1996), and the application of decision-theoretic estimations of optimal computational sequences (Russell & Wefald 1991). Heuristic strategies have been proposed as well (Pollack 1992).

The deliberation-scheduling problem bears a strong similarity to the problems of process scheduling in real-time operating systems (Tannenbaum 1992), job scheduling in operations research (Pinedo 1995), and transmission scheduling in local area networks (Nassehi & Tobagi 1987). Not all process- or job- or transmission-scheduling algorithms are applicable to deliberation scheduling, however. In particular, we require scheduling algorithms that are:

- *on-line*, i.e., construct schedules at run time;
- *dynamic*, i.e., support the random arrival of tasks;
- *stochastic*, i.e., support tasks with random computation times; and
- *soft real-time*, i.e., support the scheduling of tasks that yield less than maximal value if completed after some critical period.

Two simple and well-researched scheduling algorithms are Earliest Deadline First (EDF) and Least Slack First (LSF). These both incorporate deadline information and consequently achieve better results than algorithms that do not, such as First-in-first-out (FIFO) and Round-robin (Jensen, Locke, & Tokuda 1985; Nassehi & Tobagi 1987). It is known that, for a *schedulable* set of processes, i.e., one for which there exists an optimal schedule where all deadlines can be met, EDF and LSF produce a schedule that meets all deadlines, and hence performs optimally (Baker 1974). However, the performance of these two algorithms degrades sharply when the system is *saturated*, i.e., it has to deal with a non-schedulable set of tasks.

To schedule saturated job sets effectively, scheduling algorithms must take into account the cost of missing a deadline. This is particularly true when there are trade-offs in the acceptance rate and the deadline miss rate of tasks in the system. The environments of autonomous agents typically present such trade-offs: it may well be worth missing the deadlines for some tasks in order to achieve higher-quality performance on other tasks. Such trade-offs can be evaluated with the aid of *value-density* assessment tools. The value density of any task t is defined to be the value to the system of completing t divided by its remaining computation time. Value-density assessments are included in scheduling algorithms such as Best-Effort (BE) and Dynamic-priority (DP); previous research has shown that these algorithms perform better than EDF and LSF in saturated environments (Jensen, Locke, & Tokuda 1985; Nassehi & Tobagi 1987).

We have explored the usefulness for deliberation scheduling of the value-density measure and the algorithms that rely on it. Specifically, we identified appropriate candidate algorithms, conducted preliminary experiments to compare their performance of these algorithms, demonstrated a proof-of-concept use of these algorithms for deliberation scheduling in the DIPART system, and analyzed current limitations of the proof-of-concept system, i.e., identified certain assumptions that are made in the existing algorithms that must be relaxed to support full-fledged deliberation scheduling. In addition, we developed a modification of the Best Effort algorithm that results in improved performance for the DIPART job mix. Details can be found in (Ronen 1995; Ronen, Mossé, & Pollack 1996).

Agent Communication

One addition research topic that centrally concerns us is the specification of appropriate communication and coordination strategies for multi-agent, dynamic planning. We first briefly describe the communication package we have implemented to support communication among DIPART nodes, and then sketch approaches to multi-agent planning that we have been investigating.

Software Support for Communication

Communication among nodes in DIPART is built on a group management model (Cheriton & Zwaenpoel 1985). Groups of processes (or, in our case, agents) cluster into a single logical entity such that all communications sent by a member of the group are received by all members of the group. Thus multi-process communication is achieved by a single operation rather than by a series of operations to a (potentially unknown or partially known) set of individual agents. Group operations can take advantage of network-multicast capability, thus reducing communication overhead and increasing concurrency.

Using a group management model, we have implemented a set of communication primitives that enable the basic group operations (e.g., form a group, dissolve a group, join a group, leave a group, invite to a group, and exclude from a group) and communication actions (e.g., send, send and block, receive, group-cast, group-cast and block, receive any, receive any and block). Groups may have different structures, which determine the relationship among group members. In a *coordinated group*, the owner of the group must approve any new members, while in *peer groups*, all new members are accepted. There are also different group types: in a private group, communication is restricted only to the members of the group, while non-members may send messages to public groups. We have also implemented a group server, which maintains information about the status and membership of each group, and is responsible for synchronizing group actions. Additional details can be found in (Znati & Pollack 1994; Lauzac 1993).

Load Balancing for Distributed Planning

The communications package can be used to support a process of load balancing among the DIPART agents, so that no agent falls behind as a result of having too many responsibilities, while other agents sit idle. We have investigated a range of load-balancing techniques developed in the distributed operating-systems literature, focusing in particular on those that use dynamic thresholds.

The purpose of dynamic thresholds is to give the agent more flexibility to adapt itself to a changing environment. Consider a gift shop as an example: if the shop receives a hundred customers in a regular week day it is a busy day. However, if the shop receives a hundred customers just before Christmas, it is not a busy day. Similarly, in a military application, one would expect higher processing loads during crisis situations than during routine, peacetime operations. Instead of determining *a priori* what is a high load, dynamic load balancing evaluates the load of an agent at running time according to the partial information it possesses about the environment. As a consequence, given the same amount of tasks to perform the same agent may consider itself highly loaded or lightly loaded depending on its estimation of the system load. Dynamic thresholds are suitable in dynamic environments when a system must avoid unnecessary communications that would add an extra overhead to an already overloaded system.

Another way to lessen communication laod is to employ selective unicasting. In load balancing, one faces a trade-off between the cost of exchanging messages and the necessity of having an information accurate enough to provide efficient load balancing. When initiating load balancing an agent could send a message to every other agent asking information about their loads and wait for the answers before selecting the best agent to balance with. However this classical scheme has two drawbacks: it requires many messages to be exchanged, and it is not fault-tolerant—it does not include provisions for the case in which one or more agents is unable to respond. In contrast, in selective unicasting the messages concerning the exchange of information about the load of the system are piggy-backed to task balancing messages and therefore induce almost no overhead. Also, the scheme is non-blocking, and will not collapse should one or more agents fail. However, selective unicasting with piggybacking results in the agent being unable to access complete information about the system's current load. Thus, each agent must estimate this information by using data about the previous load history.

The load-balancing algorithms we constructed were implemented and subjected to experimentation to assess their performance relative to a variety of alternatives, including a broadcasting scheme; we also studied the relative effectives of client-driven, server-driven, and hybrid variants. We measured two things: *throughput* and *efficiency*, both of which were defined in terms of the Pacifica scenario. For instance, throughput was taken to be the ratio p/d, where p is the number of passengers for whom transportation has been requested, and d is the delay between the time of the first goal submitted and the completion of the last request. The throughput is given by the ratio p/d.

Details of the experiments, and complete results, can be found in (Lauzac 1994; Lauzac & Znati 1995). The most important result is that selective load balancing (hybrid) yields a good throughput, even compared with load balancing with broadcast. When compared with the lower bound, hybrid load balancing achieves a performance 34% higher, client driven load balancing achieves a performance 15% higher, and server driven load balancing achieves a performance 11% higher. Compared to the upper bound, selective load balancing performs only 7% worse, while using many fewer messages. Thus it appears possible to achieve effective load balancing by using dynamic thresholds, even if communication must be minimized.

Plan Merging The load-balancing work involves agents sharing the work, but each individually forming their own, more or less complete plans. Sometimes this is feasible, but at other times, agents need to form partial plans, which are then merged together. We identified four different types of situations in which some merging may be needed. In the first, a group of agents has to cooperatively achieve one common global goal. In the second type of situation, due to time constraints, execution of the plan is interleaved with the planning process itself. In the third third, each agent has its own private, individual goal. There is also a fourth situation, in which planning and execution are interleaved for a group of agents with private goals. The DIPART scenario can be viewed either as an instance of the second of the fourth type, depending on how much knowledge each of the human planning agents has about the plan for the overall mission.

For each of these situations, we described how a global plan is constructed through the process of incrementally merging sub-plans. By making use of the computational power of multiple agents working in parallel, the process is able to reduce the total elapsed time for planning as compared to a central planner. For the case in which agents do not have complete knowledge of the overall mission, we show how agents can reach consensus about what multi-agent plan to carry out using a voting procedure, without having to reveal full goals and preferences (unless that is actually necessary for consensus to be reached). Our technique also does away with the need to generate final alternatives ahead of time (instead, candidate states arise at each step as a natural consequence of the emerging plan). The agents iteratively converge to a plan that brings the group to a state maximizing the overall system utility. Details and experimental results can be in (Ephrati, Pollack, & Rosenschein 1995; 1994; Ephrati & Rosenschein 1994).

Multi-Agent Filtering In addition to plan merging, which involves explicit coordination among agents, it is sometimes useful for agents to have a means of achieving coordination implicitly. We have been investigating a strategy for implicit coordination called *multi-agent filtering*. It extends a single-agent strategy, filtering, which was developed as a way of controlling reasoning in dynamic environments. The notion of single-agent filtering derives from the work of Bratman (Bratman 1987); it involves an agent committing to the goals it has already adopted, and tending to bypass (or "filter out") new options that would conflict with their successful completion (Bratman, Israel, & Pollack 1988; Pollack 1992; Pollack *et al.* 1994). We and others have studied the effectiveness of filtering in domains with various characteristics(Pollack & Ringuette 1990; Kinny & Georgeff 1991; Pollack *et al.* 1994).

Where single-agent filtering means tending to bypass options that are incompatible with an agent's *own* goals, multi-agent filtering means tending to bypass options that are incompatible with *any* agent's known or presumed goals. We examined several forms of multi-agent filtering, which range from purely implicit, in which agents have rules of legal action that lead to their avoiding conflict without ever reasoning explicitly about one another's goals, to minimally explicit, in which agents perform very shallow reasoning to assess whether their actions are incompatible with the likely intended actions of other agents. In no cases do the agents engage in any explicit negotiation.

Our experimental results on the efficacy of multi-agent filtering are presented in (Ephrati, Pollack, & Ur 1995). The most interesting and surprising result is that, at least for the simple, abstract environments so far studied, multi-agent filtering is a dominant strategy: no matter what proportion of the agents in some environment choose not to filter, those that do filter perform better.

References

Baker, K. 1974. *Introduction to Sequencing and Scheduling.* N.Y.: John Wiley.

Boddy, M., and Dean, T. 1989. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 979–984.

Bratman, M. E.; Israel, D. J.; and Pollack, M. E. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4:349–355.

Bratman, M. E. 1987. *Intention, Plans and Practical Reason.* Cambridge, MA: Harvard University Press.

Cheriton, D. R., and Zwaenpoel, W. 1985. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems* 77–107.

Ephrati, E., and Rosenschein, J. S. 1994. Divide and conquer in multi-agent systems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94).*

Ephrati, E.; Pollack, M. E.; and Milshtein, M. 1996. A cost-directed planner. Submitted for publication.

Ephrati, E.; Pollack, M. E.; and Rosenschein, J. S. 1994. Exploitation of decision theory techniques in multi-agent planning. In *1994 AAAI Spring Symposium on Decision Theoretic Planning.* AAAI Press.

Ephrati, E.; Pollack, M. E.; and Rosenschein, J. S. 1995. A tractable heuristic that maximizes global utility through local plan combination. In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS).*

Ephrati, E.; Pollack, M. E.; and Ur, S. 1995. Deriving multi-agent coordination through filtering strategies. In *Proceedings of the 14th Joint International Conference on Artificial Intelligence (IJCAI).*

Greenwald, L., and Dean, T. 1994. Solving time-critical decision-making problems with predictable computational demands. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS)*, 25–30.

Jensen, E. D.; Locke, C. D.; and Tokuda, H. 1985. A time-driven scheduling model for real-time operating systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, 112–122.

Joslin, D., and Pollack, M. E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, 1004–1009.

Joslin, D., and Pollack, M. E. 1995. Active and passive postponement of decisions in plan generation. In *Proceedings of the Third European Workshop on Planning*.

Joslin, D., and Pollack, M. E. 1996. When is 'early commitment' in plan generation a good idea? Submitted for publication.

Joslin, D. 1996. Passive and active decision postponement in plan generation. Ph.D. dissertation, Intelligent Systems Program, University of Pittsburgh.

Kinny, D. N., and Georgeff, M. P. 1991. Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 82–88.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.

Lauzac, S., and Znati, T. 1995. Comparative evaulation of cooperative plan execution strategies in multiagent environments. In *Proceedings of the 28th Annual Simulation Symposium*.

Lauzac, S. 1993. The DIPART communications package. Technical Report 93-24, University of Pittsburgh, Pittsburgh, PA.

Lauzac, S. 1994. Load balancing algorithms for cooperative planning. Technical Report 95-26, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634–639.

Nassehi, M. M., and Tobagi, F. A. 1987. Transmission scheduling policies and their implementation in integrated-services high-speed local area networks. In *he Fifth Annual European Fibre Optic Communications and Local Area Networks Exposition*, 185–192.

Penberthy, J. S., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, 103–114.

Peot, M., and Smith, D. E. 1993. Threat-removal strategies for partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 492–499.

Pinedo, M. 1995. *Scheduling: Theory, Algorithms, and Systems.* Englewood Cliffs, NJ 07632: Prentice Hall.

Pollack, M. E., and Ringuette, M. 1990. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 183–189.

Pollack, M. E.; Joslin, D.; Nunes, A.; Ur, S.; and Ephrati, E. 1994. Experimental investigation of an agent-commitment strategy. Technical Report 94-31, Univ. of Pittsburgh Dept. of Computer Science, Pittsburgh, PA.

Pollack, M. E. 1992. The uses of plans. *Artificial Intelligence* 57:43–68.

Reece, G. A.; Tate, A.; Brown, D. I.; and Hoffman, M. 1993. The PRECiS environment. Technical report, ARPA-RL/CPE.

Ronen, Y.; Mossé, D.; and Pollack, M. E. 1996. Value-density algorithms for the deliberation-scheduling algorithm. *SIGART Bulletin* 7(2).

Ronen, Y. 1995. Meta-level deliberation as scheduling: The use of operating-systems and operations-research techniques in meta-level control. M.S. Thesis, Intelligent Systems Program, University of Pittsburgh.

Russell, S. J., and Wefald, E. H. 1991. *Do the Right Thing.* Cambridge, MA: MIT Press.

Schubert, L., and Gerevini, A. 1995. Accelerating partial order planners by improving plan and goal choices. In *Proceedings of the International Conference on Tools with AI (ICTAI '95)*.

Srinivasan, R., and Howe, A. E. 1995. Comparison of methods for improving search efficiency in a partial-order planner. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1620–1626.

Tannenbaum, A. S. 1992. *Modern Operating Systems.* Englewood Cliffs: Prentice Hall.

Zilberstein, S., and Russell, S. 1996. Optimal composition of real-time systems. *Artificial Intelligence* 79(2).

Znati, T., and Pollack, M. E. 1994. DIPART, an interactive simulation platform for plan development and monitoring in dynamic environments. In *Proceedings of the 27th Annual Simulation Society*. IEEE Computer Society Press.