

Learning to Better Coordinate in Joint Activities

A Dissertation

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Department of Computer Science

Richard Alterman, Advisor

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Andrew Eliot Garland

May, 2000

This dissertation, directed and approved by Andrew Eliot Garland's Committee, has been accepted and approved by the Graduate Faculty of Brandeis University in partial fulfillment of the requirements for the degree of:

DOCTOR OF PHILOSOPHY

Dean of Arts and Sciences

Dissertation Committee:

Richard Alterman, Chair

James Pustejovsky

David Waltz

Candace Sidner, Mitsubishi Electric Research Laboratory

©Copyright by
Andrew Eliot Garland
2000

Acknowledgments

I would like to thank my advisor, Richard Alterman, for his boundless enthusiasm, unwavering support, and unlimited patience. I would also like to thank the members of my committee — James Pustejovsky, David Waltz, and Candace Sidner — for their time and energy in improving the quality of this thesis. I have benefitted from the knowledge and teachings of other Computer Science faculty members: Ed Balkovich, Martin Cohn, Ira Gessel, Tim Hickey, Harry Mairson, Maja Matarić, George Mou, Jordan Pollack, and Jim Storer. This work was supported in part by the ONR (N00014-96-1-0440).

The other students in Rick's research group were friends as well as colleagues. Betsy Sklar and I always seemed to be facing the same hurdles and feeling the same sentiments. Dan Griffin generated a steady stream of interesting opinions. Seth Landsman, Alex Feinman, and Josh Introne were always ready and willing to help or listen despite their own heavy workloads.

Myrna Fox was an invaluable font of practical information and wisdom. Morris Dworkin, a man of many talents, enlivened life during my time in the Mathematics department. Friends in the Computer Science department without whom I could not imagine surviving graduate school are Marc Verhagen, Julio Santana, Hugues Juillé, Pablo Funes, Sevan Ficici, Jeanne DeBaie, Paul Buitelaar, and David Abrams. My research would not have been possible without Alan Danziger, David Wittenburg and the gurus keeping the system running smoothly. Tamitha Carpenter, Scott Waterman, and Pat Tufts offered guidance and unique perspectives when I switched to Computer Science.

No expression of thanks is sufficient for my wife Helene and my parents, Carl and Joan. They never doubted me or chided me for being “almost done.” Instead, they provided love, encouragement, and support. Helene shared in the joys and trials of graduate school and this work would not be the same without her constant faith and love.

ABSTRACT

Learning to Better Coordinate in Joint Activities

A dissertation presented to the Faculty of
the Graduate School of Arts and Sciences of
Brandeis University, Waltham, Massachusetts

by Andrew Eliot Garland

Research on groups of agents that work together is a large and growing field. A fundamental difficulty faced by such agents is how to efficiently coordinate their efforts with those of other participants. This *coordination problem* is both ubiquitous and challenging, especially in environments where agents have limited knowledge about, and control over, other agents and the world around them.

Coordination problems are constantly faced, and solved, in the course of *joint activities* ranging from everyday settings such as conversations and shopping to highly structured settings such as musical ensembles or athletic teams. Much research has focused on techniques that allow agents to efficiently solve coordination problems with the help of common built-in knowledge.

This dissertation builds on those efforts by showing that distributed learning is an effective addition to, and possible replacement for, built-in knowledge. Learning is an important part of individual agents acting in dynamic, uncertain environments; coordinating with independent, heterogeneous agents increases the amount of uncertainty that agents face. Learning how to better coordinate is essential to achieve high levels of performance in many joint activities.

In the model presented in this work, agents learn from experience so that their behavior more accurately reflects what works (or does not work) in practice. Two learning structures created and maintained by each agent individually are a case-base of *coordinated procedures* and a set of *operator probability trees*. A contribution of this research is a technique to learn new procedures from past, unplanned successes.

Empirical results show that learning significantly improves the performance of a community of agents that have been engineered with common and accurate knowledge of the domain. Even so, there will always be coordination problems that must be solved by the agents at run-time that are outside the scope of built-in knowledge. Results show that agents who initially lack knowledge of the abilities of their co-participants can also achieve high levels of performance. The conclusion is that the learning techniques are both robust and powerful.

Contents

Acknowledgments	iv
Abstract	v
List of Tables	ix
List of Figures	x
1 Introduction	1
1.1 Joint activity	3
1.2 Learning to better coordinate	7
1.3 Domains with coordination problems	10
1.4 MOVERS-WORLD	14
1.5 Scientific contributions	16
1.6 A guide to the reader	18
2 Principles of the architecture	20
2.1 Characteristics of experts	21
2.2 Generality versus commonality	24
2.3 Philosophical considerations	29
2.4 Performance considerations	32
2.5 Formal specification	37
2.6 Related research	42
3 Overview of the architecture	47
3.1 MOVERS-WORLD	48
3.2 Cycle of activity	50
3.3 Planning	52
3.4 Communication	54
3.5 Coordinating joint activity	56
3.6 Impossibility of optimal performance	60

4	Overview of learning	63
4.1	A new procedure learned from experience	65
4.2	Learning coordinated procedures	67
4.3	Example: learning to interleave goals	72
4.4	Coordinating from experience	73
4.5	Learning operator probabilities	76
4.6	Discussion and related research	78
4.6.1	PRODIGY	83
5	Empirical results and analysis	86
5.1	Cumulative impact of learning	88
5.2	Learning curves	93
5.3	Why is learning improving performance?	99
5.4	The benefits of coordinated procedures	110
6	Architecture details	123
6.1	The baseline planner	124
6.2	Operator probabilities	128
6.3	Execution-time dynamics	133
6.3.1	Frustration	136
6.3.2	Idling	138
6.4	Communication details	141
6.4.1	Initial requests	143
6.4.2	Basic responses	144
6.4.3	Alternative request responses	144
6.4.4	Other request responses	145
6.4.5	Terminal responses	147
6.4.6	Neither agreements nor refusals	148
6.4.7	Distribution of conversations	151
7	Learning coordinated procedures	154
7.1	Cleaning the execution trace	155
7.2	Segmenting the execution trace	157
7.3	Removing inefficiencies	160
7.3.1	Which primitive actions to compare	161
7.3.2	Identifying inefficient primitive actions	163
7.3.3	Order of removal	167
7.3.4	Removing communicative acts	168
7.3.5	Prettying chunks	174
7.4	Optimizing coordinated procedures	176
7.5	Summarizing coordinated procedures	178
7.6	Preparing summarized procedures	178

7.7	Storing procedures in memory	182
7.7.1	Storage indices for coordinated procedures	184
7.7.2	Tailoring general memories	185
7.7.3	Retrieving coordinated procedures	188
7.8	A note about intentionality	189
8	Conclusion	191
A	Sample MOVERS-WORLD problem	195
B	Other empirical investigations	202
B.1	Cooperation strategies	202
B.2	Synchronous activity	207
B.3	Varying action and communication durations	210
B.4	Goal selection strategies	213
C	Pseudo-code	216
	Bibliography	221

List of Tables

2.1	Philosophical considerations for a multi-agent framework.	30
2.2	Performance considerations for a multi-agent framework.	33
5.1	Community statistics: baseline versus learned behavior	89
6.1	Typical MOVERS-WORLD action durations.	125
6.2	How probabilities change for failures.	127
6.3	Distribution of conversation sequences.	151
7.1	Breakdown of the different reasons agents idle	156
7.2	A simple taxonomy of wait operators.	177
7.3	Coordinated procedures case-base statistics.	183
A.1	Splitting the data on the number of times the hand-truck is used. . .	200
B.1	Performance in the first episode for possible cooperation strategies. .	205
B.2	Performance in the fifth episode for possible cooperation strategies. .	205
B.3	Synchronous compared to asynchronous for the tenth episode. . . .	208
B.4	Performance for the first episode when varying communication costs.	211
B.5	Performance for the fifth episode when varying communication costs.	211
B.6	Performance of first episode for goal-selection strategies.	215
B.7	Performance of fifth episode for goal-selection strategies.	215

List of Figures

1.1	Group performance versus architectural constraints.	5
1.2	Learning agents perform at high levels.	16
2.1	Performance versus uncertainty. (Reprint of Figure 1.1.)	24
2.2	Graphical depiction of the relationship between computational expense and regions of domain uncertainty and goal interactions.	35
2.3	Agent communication interfaces.	38
3.1	Sample MOVERS-WORLD problem.	48
3.2	Stages of a joint activity.	50
3.3	Simple example of coordinated agents	58
3.4	Coordination based on first-principles: same goals.	59
3.5	Coordination based on first-principles: different goals.	59
3.6	Distribution of problem-solving knowledge in MOVERS-WORLD.	60
3.7	Near optimal coordination in simple MOVERS-WORLD problem.	62
4.1	Flowchart of the community routine.	64
4.2	Coordination based on explicit expectations: same goals.	74
4.3	Coordination based on implicit expectations: same goals.	75
4.4	Coordination based on implicit expectations: different goals.	76
4.5	Differences between theory and practice.	79
5.1	The community approaches optimal overall performance.	94
5.2	Overall community performance improves.	95
5.3	Community communication decreases.	96
5.4	Community action effort decreases.	97
5.5	Planning effort decreases when agents learn coordinated procedures.	98
5.6	Measuring how learning coordinated procedures improves goal selection.	101
5.7	Showing how better domain models lead to easier decisions.	102
5.8	CP+OP agents become attached to their plans.	104
5.9	Approaching minimal action effort.	105
5.10	Learning who to ask and when not to ask	107
5.11	Coordinated procedures lead agents to simultaneously work on two top-level goals more frequently.	109

5.12	Contributions of pieces of the coordinated procedures case-base.	112
5.13	Contributions of the optimizations on ticks and conversations.	113
5.14	Contribution of plans to achieve multiple goals.	114
5.15	Matching coordinated procedures to common knowledge.	116
5.16	Common knowledge leads to more planner search.	117
5.17	Coordinated procedures improve upon built-in knowledge.	119
5.18	Summary: learning agents perform at high levels. (Reprint of Figure 1.2.)	121
5.19	Revisiting optimizations and interleavings with built-in knowledge.	122
6.1	Flowchart of agent activity cycle.	124
6.2	Algorithm to calculate general operator probability.	131
6.3	Operator probability tree.	132
7.1	Schematic of a time-grouping segment.	158
7.2	Criteria for actions A_1 and A_2 to reverse each other.	165
7.3	The truth values of goals helps determine if communication is kept.	172
7.4	Annotating actions and preconditions for plan P that achieves goals G	179
7.5	Sample annotation structure for a LOAD action.	181
7.6	Multiple case-base entries for a single coordinated procedure.	186
A.1	Distribution of simulated execution time.	200
A.2	Distributions when the data are split.	201
B.1	Overall effort for various cooperation strategies.	206
B.2	Synchronous versus asynchronous learning curves.	209
B.3	Overall effort when varying communication costs.	212

Chapter 1

Introduction

Research on groups of agents that work together is a large and growing field. This field covers many diverse interests and motivations such as mobile autonomous robots, software agents, and smart objects. In all of these fields, a fundamental difficulty faced by agents is how to coordinate their efforts with those of other participants. This *coordination problem* (Schelling, 1963) is both ubiquitous and challenging, especially in environments where agents have limited knowledge about, and control over, other agents and the world around them.

In human terms, coordinating our actions with other people is so commonplace and constant that it is easy to overlook or trivialize its importance and the difficulty of achieving it. Coordination problems are constantly faced, and solved, in the course of activities ranging from everyday settings such as conversations and shopping to highly structured settings such as musical ensembles or athletic teams. During such *joint activities* (Clark, 1996), actors negotiate at and about *coordination points* (Alterman and Garland, 2001), which are defined as points in the activity where an actor cannot progress without the assistance of someone else.

The coordination problem is an essential issue to address in order to implement autonomous, heterogeneous agents that work together. One possible way to address the issue is to pre-determine solutions to all of the coordination problems of the domain (Malone and Crowston, 1994). However, this is problematic for complex environments where coordination problems cannot be foreseen or for groups of autonomous agents who may not be willing to participate in the canonical solution. As an alternative, the current preferred approach is to design agents to have extensive built-in knowledge about the group, such as common knowledge of the planning or execution abilities of all agents (Durfee and Lesser, 1991; Decker and Lesser, 1992; Tambe, 1997). This common knowledge makes agents more likely to be able to efficiently solve coordination problems that occur at run-time.

This dissertation builds on that approach by showing that learning is an effective addition to, and possible replacement for, much built-in knowledge. The importance of learning for embodied agents in dynamic, uncertain environments has long been recognized (Kaelbling, 1990; Maes and Brooks, 1990). This is both because of engineering hurdles and the general uncertainty of the world. The importance of learning is increased by the presence of other agents; working with independent agents whose internal reasoning processes are not completely determinable makes any multi-agent task environment uncertain. Consequently, learning how to coordinate with others should be a core component of agents designed for joint activities.

Current frameworks to support group activity either ignore learning or assume that learning is secondary to the framework (Sugawara and Lesser, 1998; Ho and Kamel, 1998; Haynes and Sen, 1998; Ohko *et al.*, 1996). In contrast, a tenet of this research is that learning and the underlying framework should be treated as equals. So in addition to asking “Given the specifications of a framework, how much can an agent learn?”, one must also ask “Given learning agents, how much does a framework

need to specify?” A challenge is to develop learning techniques given that an agent will not have complete knowledge of the motives, needs, or capabilities of the other agents in the community.

Using the techniques developed in this research, an agent acquires knowledge about the environment and other agents from experience, supplementing any *a priori* common knowledge she might have. Thus, individual agents learn to better coordinate their actions so that the agent’s future behavior more accurately reflects what works (or does not work) in practice. The learning techniques are memory-based and a novel contribution is a technique to learn procedures based on past, unplanned successes.

The empirical results show that learning leads to high levels of performance even in the worst case: when none of the internals of other agents — even cooperating agents — are commonly known at the outset. The only commonality is in a communication language organized around goals and coordination points and the ability to make unambiguous external references. The results also show that procedures that are learned from past success significantly improves performance even in the best case: when the agents in the community are initially given key pieces of knowledge about other agents.

1.1 Joint activity

This thesis concerns group efforts that can be classified as *joint activities* (Clark, 1996). Crossing the street, going to the movies, riding a tandem bicycle, conversing with another, and so on, are all examples of joint activities. A joint activity is essentially a complex of coordination problems (Clark, 1996, p. 60).

Joint activities emerge as a hierarchy of joint actions and joint (sub)activities (p. 38). Joint activities advance one increment at a time, mostly through joint actions

(p. 38). Joint actions have phases that have entry and exit points, each of which require coordination (p. 38). Coordination on the entry and exit times to each phase of a joint action can be achieved by means of different strategies (p. 64). The entry and exit boundaries of the overall joint activity are also jointly engineered by the participants (p. 38).

Clark's model of joint activity distinguishes a joint action from other actions by the intentions of the individual actors. Other prominent models specify mental attitudes that need to be commonly held by the group of agents to be engaged in a *shared cooperative activity* (Bratman, 1999), a *collaboration* (Grosz and Sidner, 1990), or to have *joint intentions* (Levesque *et al.*, 1990). An area of future research is to characterize the relationship between the mental states of participants mandated by these models and the learning framework presented in this dissertation.

Within Artificial Intelligence, research on joint activity among autonomous agents is part of a field of research called *distributed continual planning* (desJardins *et al.*, 1999). Distributed continual planning seeks theories and techniques to support cooperation and coordination of groups of agents in the face of environments where:

1. Planning knowledge or execution ability are distributed among the agents.
2. The agents are acting in complex, real-world domains. The complexity can be manifested in several ways, notably that aspects of the world can change dynamically, beyond the control and knowledge of the agents. Agents may face other difficulties such as that aspects of the world are revealed incrementally, time pressures require execution before a complete plan can be generated, or objectives evolve over time.

One approach to such environments is to design algorithms for a tightly-coupled group to collectively construct (preferably optimal) group plans as the primary result

(Corkill, 1979; Wilkens and Myers, 1998; Wolverton and desJardins, 1998). Another approach, of which this work is a part, is to investigate groups of agents that interleave planning and execution. This research investigates joint activities where the individual agents are motivated by personal objectives and do not reason about group-wide objectives or attempt to establish or maintain group-wide mental attitudes. The extent to which the activity can be successful naturally depends on the degree to which the local objectives converge.

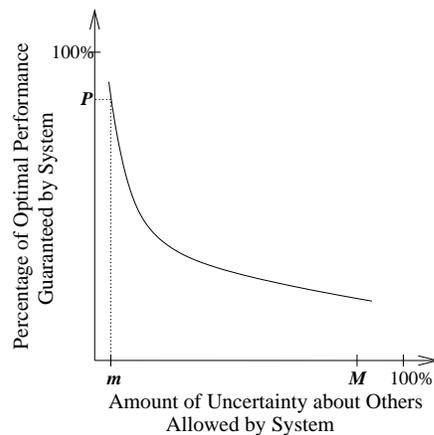


Figure 1.1: Group performance versus architectural constraints.

Figure 1.1 depicts a decision that must be weighed by a multi-agent system designer. On one hand, the designer wants to be able to guarantee performance (measured in a task-appropriate metric) as close to optimal as possible. On the other hand, such a guarantee requires that each agent has little uncertainty about what the other agents will do. How can the amount of uncertainty about other agents be reduced by the system designer? In short, the more knowledge about other agents that is available, the less uncertainty there will be. This can be manifested in ways such as the following:

Common planning knowledge. This can be knowledge about, for example, all agents' planning operators, goals, and cooperation strategies.

Exchange of detailed data structures. Exchanging detailed data structures during the activity substantially reduces the amount of uncertainty that the agents face. However, there are drawbacks. First, communication usually “takes time,” i.e., counts in the performance metric. Also, complicated ontologies are only a benefit if all agents in the community are able to generate, parse, and reason about them.

Controlled distribution of (sub-)goals. The amount of uncertainty can be reduced if a central agent assigns goals and informs agents about other agents' goals. A middle ground between this and fully autonomous agents would be a hierarchical organization where agents can assign goals to their subordinates.

In current distributed continual planning frameworks, the focus has been on optimizing initial performance by building in common knowledge in order to reduce the uncertainty faced by the community. *An equally important research question is how to design a framework that accommodates a dearth of common knowledge while still guaranteeing that a system is able to solve the task.* In other words, when solutions are theoretically possible, it is essential that the group achieves the solution; optimal group performance is desirable, but is a secondary concern.

There are important reasons why researchers should be interested in learning, as a supplement to built-in knowledge, to solve coordination problems. Even an agent who is designed to act autonomously in a single-agent system will face situations outside of the scope of her programming. “It is rarely the case that the human programmer completely understands the interactions between a robot's sensors and effectors and the environment well enough to write a correct program, except through a process of

trial and error. It is much simpler and more generally useful if the robot can conduct that trial-and-error learning on its own.” (Kaelbling, 2000)

Since building individual agents to do “real-world” tasks is already a formidable engineering burden, it is unlikely that agents can be designed to include all relevant engineering knowledge about all possible collaborators. And the additional complexity of interacting with other agents over whom she has limited control increases the likelihood of facing unexpected circumstances. It is it also unlikely that efficient coordination can be achieved solely by exchanging information at run-time because of communication costs. These constraints combine to make the coordination problems difficult to solve efficiently in joint activities.

1.2 Learning to better coordinate

Given a task environment with heterogeneous, independent agents who have limited knowledge about, and control over, other participants makes the coordination problem very challenging. Careful design will provide the basis for solutions, but there will inevitably be situations unforeseen by the original designers. If the task domain exhibits regularity, learning from the experience at the activity provides an effective means for agents to solve problems more efficiently.

In the learning model presented in this work, agents individually apply memory-based techniques in order to use past experiences to guide future behaviors. Agents can overcome a lack of initial common knowledge with learning. Even when there is initial common knowledge, acting based upon past shared experience improves performance. Empirical results from the implemented test-bed, when agents are built with or without common knowledge, demonstrate the viability of this framework.

Solving coordination problems efficiently can be translated into the following question that an agent needs to answer:

What is the best way for another agent and I to coordinate our efforts in order to achieve my goal?

The techniques developed in this research allow an agent to use past run-time experience in order answer this question indirectly by allowing the agent to address the following questions:

1. In the past, did I need to ask another agent to help me achieve my goals?
2. When I asked another agent to help me achieve my goals, was the other agent able to help me?
3. In the past, how frequently was the other agent willing to help me when I asked?
4. Based on past experience, is the other agent likely to help me without being asked?
5. In past similar circumstances, did another agent ask me to help achieve her goals?
6. If I expect another agent to ask me to help achieve her goals, what has she asked me to do in the past?
7. If I expect another agent to ask me to do something, should I wait for her to ask me, should I do it without being asked, or should I contact the other agent to say that I expect her to ask me?

Solving coordination problems with less effort requires that an agent learns to change the way she acts at run-time. In turn, this requires that the agent is able to

update the model of the world that drives her behavior to reflect past experiences interacting with other agents and the world.

Although the focus of this research is on learning to better coordinate, there are larger learning issues involved. At an abstract level, there are three ways an agent can update her model of the world through learning. First, the agent may add to the set of plans she can generate if the execution-time activity led to a solution in a way that was previously unknown. Second, the agent can revise previous domain estimates, such as probabilities measuring how likely another agent is to agree to help or how likely a primitive action is to change the state as desired (cf. *reinforcement learning* (Kaelbling *et al.*, 1996) or *explanation-based learning* (DeJong and Mooney, 1986; Mitchell *et al.*, 1986)). Third, based on past execution problems, the agent can learn when not to use a known plan (q.v. Hammond (1990)). A complete learning framework would include techniques of all three types.

Consider the case of a new branch sales office of a photocopier company. There are two kinds of employees — sales representatives and network specialists — and copiers can be sold as stand-alone units or integrated into a local network. Initially, the sales representatives know several ways to close deals, and the network specialists know a handful of possible network configurations. Sales will improve as the employees better learn their own jobs and learn about the skills of their co-workers. Eventually, an employee's actions will largely be based upon sales experiences with the company, not (directly) upon knowledge the employee had before being hired. There are three qualitatively different ways in which a salesperson learns to improve her job performance:

1. The salesperson will learn new ways to sell equipment as a result of working with a network specialist. This does not mean that a salesperson will learn

the technical skills or planning knowledge of the specialist. Instead, the sales representative will learn the situations in which a network specialist can help the salesperson make a sale, as well as what kind of support the specialist needs in turn from the sales representative.

2. The salesperson will operationalize abstract knowledge. For example, rookie salespeople are told of many sales techniques, but will be better sellers after using the techniques in practice.
3. The salesperson will learn to avoid past failures. For example, as a result of lost deals, sales representatives learn when they should not do a “hard-sell.”

In the framework presented in this thesis, agents use separate learning structures to revise their models of the world in two of the three general ways described. One structure, a case-base of *coordinated procedures*, is specifically aimed at acquiring new ways to solve problems based on past successful joint behaviors. The second structure, a set of *operator probability trees*, revises domain estimates — the likelihood of success of actions — based upon the outcome of past action attempts, both successes and failures. A third structure to prevent generating certain plans based on past experience is an area of future research. These structures are created and maintained by agents individually.

1.3 Domains with coordination problems

There are several problem areas that inherently contain both coordination problems and too much uncertainty to rely solely on built-in knowledge to solve them efficiently. The domains described below are cooperative domains, in the sense that agents have individual goals whose achievement requires that assistance of others. The first part

of this section briefly outlines relevant current research areas and latter parts present three possible future applications.

- There are systems in which a *software agent* (Bradshaw, 1997) assists a human to use a complex interface (Maes and Kozierok, 1993; Lashkari *et al.*, 1994; Rich and Sidner, 1998; Rhodes and Starner, 1996; Alterman *et al.*, 1998a). The underlying interface (as opposed to the agent) is generally designed via the principle of *direct manipulation*, which entails that all user actions are reversible, so a reasonable goal is to reduce the number of mouse and key-press events by the user. The agent and the user must be able to coordinate their efforts in order to realize this goal.
- Software agents that support conducting business on the Internet (among many: Ma, 1999; Papazoglou, 1999) is a burgeoning field. One approach is for a single organization to be represented by group of agents that have different capacities. For example, the group might contain agents specialized for shipping, sales, marketing, accounting, and warehousing. In order to conduct business with agents outside the community (i.e., representing other organizations), the members of the community must be able to coordinate actions among themselves.
- Other software agents that need to learn are personal assistants that act as surrogates for human users (Maes, 1997; Dent *et al.*, 1992). The assistant needs to contact various information sources in order to develop a suggested itinerary including dates and costs for airlines, rental cars, hotels and other events that match the user's personal interests (e.g., plays, concerts, fine dining). In addition to learning user preferences, assistants need to coordinate with other on-line agents in order to provide information that is detailed, accurate, and up to date.

- Robotic agents replacing humans in environments that include an element of danger to the humans who now must undertake the task is a future application area attracting current research interest. For example, in construction (Stentz *et al.*, 1999) or in space (Truszkowski *et al.*, 1999). In each case, agents should be adaptable to unforeseen and changing circumstances, including the capabilities of other agents with whom they must coordinate their efforts.
- An autonomous driving car (Franke *et al.*, 1998; Brumitt *et al.*, 1998) of the future will face numerous coordination problems when interacting with other vehicles on the road. While the car's driving will be constrained by social laws, such as always driving to the right of the divider, stopping at red lights and stop signs, *et cetera* it will be impossible for the social laws to completely determine the car's actions. There will always be situations not explicitly considered or in which the "correct" action is subject to each driver's interpretation of the situation. Therefore, there will be many coordination problems that the truck (and nearby drivers) must solve dynamically on the road.
- A kitchen of the future will be a community of collaborating intelligent appliances and objects (Norman, 1998) that face a diverse set of coordination problems. The objects in the environment would not, in general, be designed or implemented by the same party, so they would not contain detailed knowledge about the other agents with which they must interact.

An example task for a smart kitchen is to prepare a complete dinner menu, requiring that several dishes be prepared concurrently, and perhaps several objects need to use the same appliance at the same time. The preparation of each dish in isolation will be a challenging engineering feat that requires that the appliances and the food stuffs, rather than a set of general instructions, deter-

mines when and how the appliance manipulates the food. Since the particulars of preparing each dish cannot be determined beforehand, the preparation of the meal as a whole cannot be planned with certainty solely by analysis. This means that there is an opportunity for learning to help the appliances better coordinate their efforts at making meals.

- Mobile warehouse agents are an example of a futuristic domain that will contain coordination problems. The purpose of warehousing robots would be to process incoming requests from a person (or agent) in the shipping department to move various items from inside the warehouse to a loading dock. As long as the group of agents moves inventory from the interior of the warehouse to the loading dock within a “reasonable” time period, the system would be useful, but it is desirable for the agents to process requests as quickly as possible.

There are numerous coordination problems that would arise in this setting. Any time an object was too bulky or heavy for an agent to manipulate on her own, she would have to coordinate her efforts with another agent. Autonomous hand-trucks or forklifts would only be capable of moving boxes indirectly and so need to coordinate with supporting agents of different types. The warehouse agents need to coordinate their efforts with agents from a different company, the shipper, in order to load the delivery truck. Complicating matters, the agents may not have worked together before, may not have dealt with similar objects and tools, and may not have been faced with similar arrangements of objects in the warehouse or the loading dock in the past.

This list gives a flavor for the many settings in which coordination problems will occur. In order for a system to be an effective solution to these challenging domains, solutions to common coordination problems will have to be part of the initial design.

However, the tasks are too complex to guarantee that all coordination problems can be envisioned or that anticipated solutions will be effective in practice.

1.4 MOVERS-WORLD

The domains listed in the previous section were quite varied. However, several key features can be distilled out in to design a test-bed system. First of all, the environment contains a heterogeneous collection of agents. Second, it is unlikely that each agent could be designed with all relevant engineering information about the others. This is partially due to the complexity of the agents and partially due to the likelihood of different engineers. Third, although the agents must cooperate at times, they do not necessarily have the same set of goals. Finally, the agents are able to autonomously decide whether or not to cooperate on a request made by a peer.

These domain characteristics make coordinating a group of agents engaged in a joint activity very difficult. Although the test-bed domain, called MOVERS-WORLD, might appear on the surface to be simple, there are hidden complexities. MOVERS-WORLD is designed to contain the essential elements that make the problem of coordinating independent agents challenging without extraneous window dressing — MOVERS-WORLD is a streamlined domain, not a simplified one.

In MOVERS-WORLD, the task is to move boxes from a house onto a truck and is analogous to the autonomous warehouse domain. MOVERS-WORLD has multiple agents who can be differentiated by an outside observer by their types: some are “lifters” (e.g., L1 and L2) and some are “hand-truck operators” (e.g., HTO). Agents of the same type may have differing capabilities due to attributes such as strength. Most actions are type-specific (such as LOAD-TOGETHER for lifters and STAND-HANDTR for hand-truck operators), meaning that only agents of that type

can attempt them, but all agents are able to move and communicate. Different actions have different durations and the durations of conversation is determined by the amount of information that is exchanged, The agents do not know their type or even have an internal representation of the concept of type. Furthermore, they have no built-in knowledge about the special purpose operators for agents of other types; for example, lifters do not know initially how the hand-truck can be utilized.

In terms of the coordination problems faced by the agents, a central feature of this domain is that the agents are coordinated at run-time through communication. Agents do not communicate at planning time; they plan independently, act independently, and only communicate when necessary to establish cooperation or to maintain coordination. Communication happens at and about coordination points.

Many rigorous empirical studies were conducted in the MOVERS-WORLD test-bed. Two key results are summarized by Figure 1.2. Chapter 5 explains these results more fully and presents other metrics for characterizing the benefits of learning. Chapter 5 also describes the experimental methodology and provides a detailed technical analysis of the mechanisms by which learning influences the performance of the community.

Figure 1.2 plots the performance (measured in simulated time steps) of three groups of agents. The standard for comparison is a group of agents with “Engineered Knowledge.” The two groups being compared to the standard are a community of *tabula rasa* agents who are learning from past experience (“Learning w/o Engineered Knowledge”) and a group of agents that have some of the engineered knowledge built-in and who are learning (“Learning with Engineered Knowledge”). The error bars shown correspond to 99% confidence intervals.

Figure 1.2 shows that even when agents have been engineered with common knowledge, acting from shared experience leads to improved performance. However, the

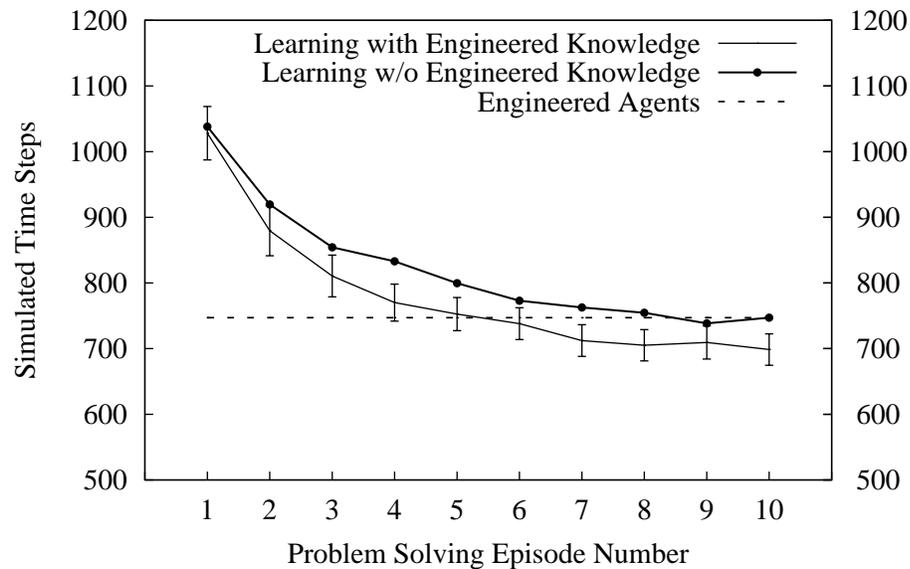


Figure 1.2: Learning agents perform at high levels.

point of this graph is not to emphasize that learning can benefit agents engineered with common knowledge. There will always be coordination problems that are not anticipated by the engineer and must be solved by the agents at run-time. Figure 1.2 also shows that even *tabula rasa* agents can achieve high levels of performance. The conclusion is that the learning techniques are both robust and powerful.

1.5 Scientific contributions

This thesis reports contributions to the fields of multi-agent systems, machine learning, case-based reasoning, and planning.

Contributions to Multi-agent Systems

- Implements a fully automated learning system, where agents incrementally increase their domain knowledge based upon interactions with the environment.

- Implements a system where agents coordinate principally by communicating at and about coordination points in their activity.

Contributions to Machine Learning

- Develops learning techniques where agents add to their set of known procedures to include the result of successful experience. The learned knowledge is acquired from activity and is not operationalized or cached versions of previous knowledge.
- Develops techniques to learn more accurate probability estimates based on run-time experiences.

Contributions to Case-based reasoning

- Develops techniques to add cases to memory based upon success rather than failure.
- Develops techniques to extend single-agent procedural learning systems into multi-agent domains.

Contributions to Planning

- Planning for actions of varying durations.
- Planning using self-generated probabilities.
- Develops a heuristic that merges the reinforcement learning concept of discounted reward into a traditional state-space planner.

1.6 A guide to the reader

This section provides a summary of the major chapters of the thesis.

Chapter 2 motivates the rationale for supplementing built-in common knowledge with learning techniques in a framework to support joint activity among heterogeneous, independent agents.

Chapter 3 provides an overview of the basic system architecture. It briefly describes the baseline planner of the agents, how communication is implemented, and how joint activity is coordinated.

Chapter 4 provides an overview of the learning techniques of the system architecture. It includes a summary of the seven steps in the process to learn new coordinated procedures, the basics of how operator probabilities are used and maintained, and examples of learned coordinated procedures.

Chapter 5 presents a thorough empirical analysis of the implemented test-bed. Various hypotheses regarding how the agent community is improving their performance are discussed and tested.

Chapter 6 revisits the architecture to provide more detailed technical information. It includes sections on the baseline planner, operator probabilities, execution-time dynamics, and the communication protocol.

Chapter 7 gives extensive technical details about the memory-based techniques that agents use to learn coordinated procedures outside of the scope of their baseline planners. The procedures are not instantiations of more general procedure schema that the agents already knew.

Chapter 8 is the conclusion.

Appendix A includes output of the system when the community is solving a sample problem, demonstrating the typical behavior of agents that lack common knowledge.

Appendix B contains other empirical investigations that demonstrate the effectiveness of the learning techniques under a variety of other possible implementational decisions for MOVERS-WORLD.

Appendix C contains pseudo-code for some routines in order to clarify the exposition in the main text.

Chapter 2

Principles of the architecture

The most realistic way to expect agents to be able to solve coordination problems efficiently is through a combination of engineering and learning. However, the most realistic setting for artificial agents to be involved in joint activities in the near future is over the Internet, where common built-in knowledge may be problematic to engineer. With that in mind, this chapter emphasizes the importance of a general framework for acting and learning that does not rely heavily on common knowledge. Experiments in Chapter 5 verify that the implemented framework is effective when agents are engineered with common knowledge as well.

A community of heterogeneous, independent artificial agents can be likened to groups of human experts. In general, the agents will have different areas of expertise that is localized, not all-encompassing. They will plan, predict and evaluate alternatives using highly specialized structures that are tailored to the activity they are designed for. This premise has important consequences for agent architectures designed to support joint activity. The first is that it becomes burdensome to engineer agents with common knowledge sets. The second is that communication costs mean that exchanging detailed internal structures will not solve this problem alone.

Techniques that rely on hard-wired commonality to establish cooperation and coordination are unlikely to solve all coordination problems efficiently. This makes learning a necessary feature of a multi-agent architecture. MOVERS-WORLD uses explicit communication about *coordination points* as the primary mechanism to handle cooperation and coordination. Communicated coordination points form an essential piece of the learned coordinated procedures that lie outside the scope of an agent's initial planner.

The implementation was influenced by several other pragmatic and performance-related considerations, such as the belief that communication costs should be included in the overall measure of community performance. As a result, communication happens at run-time (rather than before the activity begins), specifically at the moment when coordination becomes necessary. Delaying communication in such a manner is well-suited to dynamic domains, where planning too far into the future is not sensible because ongoing developments can render such contingencies moot.

2.1 Characteristics of experts

Interest in modeling the group efforts of experts comes from two directions. From one direction, inter-disciplinary studies of human expert collaborations are taking increasingly prominent roles in academic and industrial settings. From another direction, there is a great amount of interest in developing communities of artificial agents with different knowledge sets who can work together to solve problems.

Defining who is an expert is problematic. Ericsson and Charness (1994) define it as someone who is an outlier (two standard deviations above the mean) when measuring their performance. However, this requires a concrete metric such as chess rating, typing speed, or measured athletic prowess. Ericsson points out that, in most

domains, it is easier to identify people who are socially recognized as experts than it is to specify the observable performance at which these individuals excel. In other words, perceived expertise is not equivalent to being an expert. For example, financial experts' performance does not differ from the result of essentially random selection of stocks. Ericsson cites Camerer and Johnson (1991) in noting that:

A large body of evidence has been accumulated showing that experts frequently do not out-perform other people in many relevant tasks in the domains of expertise.

In this light, the term “expert” is best interpreted as a specialist. An expert is an agent with an area of expertise (e.g., cardiothoracic surgery or natural language processing), but this area does not necessarily encompass the entire domain (e.g., all surgical procedures or all artificial intelligence techniques). Furthermore, an expert may be a novice at working with others. This means that the expert cannot have in-depth knowledge about the abilities of all her co-workers.

Collaboration of experts occurs frequently in the real-world, but usually when the areas of expertise overlap to a large extent. Committees of many types fall into this category: program committees for conferences, U.S. Senate and House committees and sub-committees, boards of chairmen for corporations, and Internet Engineering Task Force working groups. More everyday examples are when sub-contractors collectively construct a home or bridge, when a team of attorneys jointly develop a case, or when a sales staff works out a complicated proposal for a prospective client.

The premise that humans can be experts who are novices at collaboration is unrealistic. By the time someone has enough experience in their domain to attain expertise, they will usually have filled many different roles in group projects (Lave and Wenger, 1991; Hutchins, 1995). However, even someone with experience collaborating will need to learn the strengths and weaknesses of participants in newly formed groups.

In contrast, the premise that participants are novices at working with others is well-suited to artificial agents, especially networked software agents. In such environments, new agents are constantly being introduced with novel skill sets. Also, they may be implemented by different parties and have little or no experience interacting with other agents before they are deployed. By contrast, these same agents will, in general, be programmed with a great deal of task-specific knowledge.

A key characteristic of experts is that they develop and maintain representations — used for planning, prediction and evaluation — that are highly adapted to their area of expertise (Ericsson and Lehmann, 1996). Although prolonged experience in the domain is necessary (Ericsson advocates a “ten-year rule of necessary preparation”), experience alone is not enough:

In sum, expert memory performance is not an automatic consequence of domain-specific experience but rather requires the acquisition of specific memory skills tailored to the demands of working memory by the particular activity.

Furthermore, Voss *et al.* (1983) showed the specificity of expert reasoning: experts in domains such as chemistry and social science lacked the knowledge and strategies to successfully analyze a problem in political science.

The highly adapted and localized nature of experts’ representations means that a framework to support joint activity of experts cannot rely on much commonality. Participants with different areas of expertise will not reason in the same manner or with the same structures; furthermore, they will not be able to make use of the internal representations of other experts even if they had access to them. The lack of common knowledge results in a system where there is a high degree of initial uncertainty about other other agents and the world.

2.2 Generality versus commonality

This thesis presents a framework for distributed continual planning (desJardins *et al.*, 1999) that emphasizes generality rather than high-quality initial performance. Generality is achieved by assuming limited initial common knowledge (namely, only a base vocabulary and a simple communication language). While this generality can lead to sub-optimal initial solutions, it is a more appropriate framework for supporting activity of experts; also, if the participants are learning agents, solution quality will improve significantly over time.

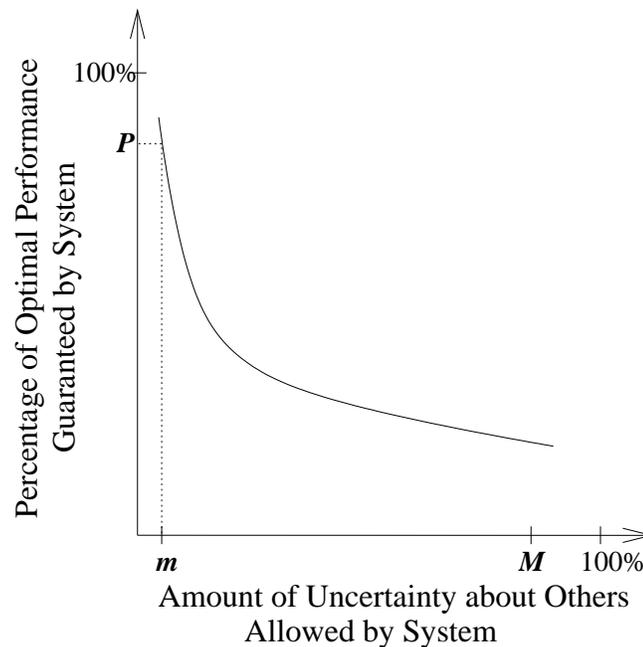


Figure 2.1: Performance versus uncertainty. (Reprint of Figure 1.1.)

Figure 2.1 reproduces an earlier figure that abstractly depicts a trade-off that must be weighed by a multi-agent architect.¹ On one hand, the system designer

¹Figure 2.1 can be viewed as a re-interpretation of the *knowledge-search trade-off* in terms of a system with multiple agents acting together. The bias of the system is interpreted as the amount of certainty each agent has about her co-participants. And computational complexity is the amount of effort the community requires, at run-time, in order to solve problems.

wants to be able to guarantee performance (measured in a task-appropriate metric) as close to optimal as possible, marked as P . On the other hand, such a guarantee requires that each agent has little uncertainty about what the other agents will do. An individual implementation corresponds to a single point in this uncertainty \times performance space although there might be families of related implementations (e.g., the same framework populated with agents who are learning).

Performance in Figure 2.1 is measured as the percentage of optimal, where optimal performance may contain “mistakes” or inefficiencies due to unforeseeable exogenous events. In other words, optimal does not mean best possible under the best circumstances, rather it means the best possible given the actual dynamics of the environment. For most domains where agents are engaged in an activity, the performance metric will be dominated by action effort and communication costs rather than planning costs. Also, different task distribution schemes will impact the metric indirectly, by changing the amount of planning or communication needed.

The uncertainty versus performance tradeoff in Figure 2.1 is relevant to a discussion of multi-agent frameworks since uncertainty can be controlled to some extent (to what extent depends on the domain) by endowing the agents with common knowledge. Common knowledge can be built into a framework by proscribing agent design to use common planning knowledge, to exchange detailed internal structures or by having common knowledge of (and possibly control over) the distribution of tasks within the agent community. Each of these topics will be addressed in more detail below.

Common planning knowledge. The amount of uncertainty about other agents can be reduced substantially if the agents have access to, and can assume others have access to, common planning structures. Information could be common knowledge by

design or by the exchange of detailed internal structures before or during the activity. Previous research has assumed that agents have most of the following common pieces of planning knowledge (Corkill, 1979; Wolverton and desJardins, 1998; Wilkens and Myers, 1998; Durfee and Lesser, 1991; Decker and Lesser, 1992; Tambe, 1997; Grosz and Kraus, 1996; Levesque *et al.*, 1990; Stone and Veloso, 1999):

1. Common knowledge of all agents' capabilities (planning operators).
2. Common knowledge of all agents' reasoning strategies used for planning and/or assessment.
3. Common knowledge of all agents' planning task (goals).
4. Common knowledge of all agents' cooperation strategies.
5. Common knowledge about group-oriented reasoning structures (team operators).

In the framework developed in this thesis, none of these pieces of common knowledge are available to the agents. The agents have no initial knowledge of other agents' capabilities (in terms of either planning operators or primitive action ability), goals, and attitudes about cooperating. In the absence of this information, agents make optimistic assumptions: they assume that the other agents are able to help them somehow. So, if an agent needs to achieve a subgoal and cannot construct a plan to do so, the agent requests another agent achieve the subgoal. This can lead to misguided, time-wasting requests. However, agents can learn to associate positive and negative responses to requests using the techniques in Section 6.2.

Exchange of common data structures. The ability to communicate is an essential piece of a model of joint activity. However, the way in which communication

is implemented can substantially influence the amount of uncertainty. As the level of detail in the structures passed from another agent increases, the amount of knowledge about the other agent's mental state (e.g., planning operators, beliefs, desires, or intentions) can be inferred. However, in order to make such inferences, the transmitted structure must have its ontology pre-specified as part of the common knowledge of the community.

Communication presents the most complicated issue regarding the uncertainty versus performance trade-off. Since the performance metric is task-specific, the cost of transmitting larger data structures or transmitting data more frequently may be included in the metric. In this case, more common knowledge cannot be engineered in "for free," by merely designing in ubiquitous communication. Rather, it is an empirical question to determine the costs and benefits of transmitting information. The issue of how frequently data should be transmitted within a particular system has been addressed (Tambe, 1997; Wolverton and desJardins, 1998; Gmytrasiewicz *et al.*, 1991; Stone and Veloso, 1999), but it is equally important to consider the representation for the data being transmitted.

The request R that A makes to a possible helper B (possibly a set of agents) will typically be a subgoal expressed in some basic pre-defined vocabulary that describes the domain (e.g., a predicate language). This might be accompanied by additional information of several types; the following types are listed in order of increasing detail (and therefore decreasing level of uncertainty about A 's intended future actions):

Information that can be represented in the base vocabulary. For example, postconditions or local state of world assessments. (Corkill, 1979; Wolverton and desJardins, 1998)

Information that represents a simple relationship. Namely, a relationship between items that can each be represented in the base vocabulary. For example, coordination relationships or a goal tree that shows the relationship between R and A 's other goals. This is more complex than the previous item because the syntax and meaning of relationships have to be commonly known at the outset. (Decker and Lesser, 1992).

An intermediate structure that supports plan generation. For example, by referring to plans or actions, such as an activity map or variable or ordering constraints. (Durfee and Lesser, 1991; Wolverton and desJardins, 1998)

A single-agent plan. If B is capable of understanding A 's plan representation, this allows B to infer all possible relevant postconditions, state of world assessments, goal relationships and ordering constraints. (Pollack, 1996)

A multi-agent plan. B has access to all of the same information as above, as well as A 's interpretation of how A 's plan interacts with those of other agents (including, but not limited to, B). (Durfee and Lesser, 1991; Grosz and Kraus, 1996)

In the framework presented in this work, only information that can be represented in the base vocabulary ever accompanies the sub-goal request. Also, communication is explicitly considered in the performance metric by occurring at run-time. Commu-

More generality

More commonality

nication requests occur at, and relate to, *coordination points* (Alterman and Garland, 2001), which are defined as a point during the activity where A needs the assistance of another agent to achieve R before A can continue with her plan.

Controlled distribution of goals. The amount of uncertainty can be reduced if a central agent assigns the agents their goals and informs them of other agents' goals. A middle ground between this and fully autonomous agents would be to have a meta-level organization where agents (e.g., "captains" or team leaders) can legislate what goals their subordinates (e.g., squad or team members) work on. Other possibilities to reduce uncertainty about task allocations are blackboard structures and broadcasting. (Corkill, 1979; Lesser *et al.*, 1975; Davis and Smith, 1983; Grosz and Kraus, 1996; Levesque *et al.*, 1990; Pollack, 1996; Wolverton and desJardins, 1998; Durfee and Lesser, 1991; Stone and Veloso, 1999)

In the basic (i.e., without learning) framework presented in this chapter, agents make no assumptions about what goals other agents may or may not be pursuing. At run-time, the agents may adopt new goals when they receive a request. Also, as mentioned above, when an agent reaches a point in their activity where they need the assistance of another agent (a coordination point), they assume assistance will only be given in response to a direct request. Eventually, the learning mechanisms of Chapter 7 allow an agent to learn the situations where assistance is likely without an explicit request.

2.3 Philosophical considerations

This section summarizes the philosophical considerations that underly the architecture developed in this dissertation. This thesis is interested in a framework to support

joint activity among heterogeneous, independent agents. Current approaches have relied on agents that have common knowledge to achieve high performance, as explained in the previous section. However, the first section of this chapter argued that it is not always appropriate for such a framework to rely solely on detailed common knowledge. The uncertainty that results from limiting the amount of knowledge is a challenge that is met by the learning techniques of the individual agents.

There are two other major philosophical considerations that went into the development of the final system. One reflects a belief that agents should communicate at and about coordination points; the other is that agents should attempt to solve problems pragmatically. Table 2.1 summarizes the three major philosophical biases of the system and their practical consequences.

Consideration	Implication	Consequence
Heterogeneous, independent agents can be viewed as artificial experts	Agents have specialized internal representations	Agents cannot be built with knowledge of the internal structures of other agents
Group efforts of cooperative agents can be considered joint activity	“Synchrony of [joint] action requires coordination on the entry and exit times to each phase.” (Clark, 1996, p. 84)	Agents should negotiate at and about coordination points
Uncertainty about the world and other agents cannot be “designed out” of the problem	Pre-activity projections are bound to be wrong	Agents should attempt problems pragmatically, not analytically

Table 2.1: Philosophical considerations for a multi-agent framework.

Restricting common knowledge makes designing a general method of communication and coordination difficult. The solution adopted in this work is that agents communicate about the specific point in the activity when they need help. These

requests are called *coordination points* (Alterman and Garland, 2001) since agents must coordinate their efforts in order to move forward from this point in the activity. This framework allows for exchanging the essential information to solving cooperative activities without having to assume much commonality. The only commonality assumed is a (predicate) language to describe the domain, a small set of communication primitives, the ability to make unambiguous external references (i.e., all agents know which object PR12-MBOX refers to) and error-free transmission (i.e., no mis-hearing of requests).

Colloquially, the term “coordination point” will also be used in this paper to describe the request itself; the context should make it clear which meaning is appropriate. If all actions are reversible and the goals of the agents do not conflict, conversing about individual coordination points at the time when they arise during the course of activity is sufficient to guarantee that the activity will be completed.

In Clark’s view (Clark, 1996), joint activities have participants who assume public roles and share joint public goals. This differs slightly from the implementation of joint activity in this work. The public role of a participant is a label that describes to others the participant’s abilities; in this model, that information is learned through interaction experience. And, although there are no joint public goals for the problem as a whole, coordination points that are agreed-to are mutually held and mutually known by the participants and are used as a basis for learning.

The third major philosophical consideration that shaped the framework is the view of agents as pragmatic actors. The dynamic and uncertain nature of the world means that any plan is bound to fail on some occasions, regardless of how much information is exchanged by the agents beforehand. In fact, it is even possible that opportunities could be missed by delaying action until negotiations have taken place. Plans orient the individual agent as to how to proceed (q.v. Suchman, 1987) but,

because of uncertainty in acting, plans are continuously being revised. Agents are assumed to be adaptive planners (Alterman, 1988), so if the activity does not unfold as anticipated, the agent's plan will be revised to reflect the ongoing interaction between the agent and her environment.

The agents can be considered pragmatic if they take this philosophy one step further. Instead of viewing the environment as merely an up-to-date reflection of the current activity, it can be viewed as an aid to activity. For example, FLOABN (Alterman *et al.*, 1998b) is a model for agents that seek out contextual clues in the environment as a means of adapting plans when they break down. Clues are present because the design of the environment reflects the development of a common procedure by prior generations of agents. Therefore, as a result of interactions with the environment, an agent's internal representation of a procedure starts to approximate the knowledge acquired by the community during past similar activities. In other words, in many situations the environment is specifically designed to indirectly transfer the knowledge of other agents' previous activities to the current agent situated in an ongoing activity.

2.4 Performance considerations

The basic (i.e., without learning) framework is general, but does not necessarily lead to high performance solutions by the community. By assuming that learning is a core component of the agent architecture, a reasonable assumption for complex domains, performance does reach high levels. In this sense, learning could be considered a design decision motivated by performance considerations.

This section provides arguments in support of two other implementation decisions that are motivated by performance, rather than philosophical, considerations. These

decisions are: agents should only plan for one top-level goal at a time when using the baseline planner, and agents should delay communication as long as possible (i.e., until the agent can make no more progress on her own). Related to the latter point is the decision that communication should occur when acting instead of when planning. Table 2.2 summarizes the performance considerations.

Consideration	Implication	Consequence
Domain regularities exist	Performance will improve by remembering past experiences	Learn
Actions and plans will fail, regardless of how much agents communicate to each other	Contemplating long-range plans will often be wasted effort	Plan for one goal at a time; delay communication as much as possible
Communication costs cannot be overlooked	Communication as a sub-process to planning is too slow	Communicate at run-time

Table 2.2: Performance considerations for a multi-agent framework.

Incremental planning. Computational effort can be saved by an incremental approach to planning when using a linear planner if the goals are independent. In other words, there is no loss of generality by only planning for the first unmet goal when calling the baseline planner and there is a computational savings. Incremental planning helps lessen the *conjunctive goal problem* (Chapman, 1987) and allows the agent to more easily tie their plan to their *situated activity* (Suchman, 1987).

For a more formal argument of the advantage of incremental planning, assume that the agent has somehow ordered their set of n unmet goals G ; call the ordered goals (G_1, G_2, \dots, G_n) . Let (P_1, P_2, \dots, P_n) be the plans produced by the planner if the planner were called n times, once for each G_i . Note that P_1 will also be the initial

steps in the plan to achieve G . Let (E_1, E_2, \dots, E_n) be a measure of the planner effort needed to generate the plans. There are three cases to consider:

1. All P_i exist; that is to say, the planner can find plans for all G_i . In this case, using \circ and \bigcirc to denote concatenation:

$$\begin{aligned} Plan &= P_1 \circ \bigcirc_{i=2}^n P_i \\ Effort &= E_1 + \sum_{i=2}^n E_i \end{aligned}$$

So the agent will expend at least $\sum_{i=2}^n E_i$ more effort to construct a plan that will lead to the same behavior until G_1 is accomplished. All of this extra effort will be wasted unless the agent successfully executes all the steps of P_1 . Even if the agent does execute P_1 successfully, nothing is gained since the planner can construct a plan to accomplish G_2 through G_n after G_1 is accomplished. Furthermore, a plan constructed at that time will match the current state of the world, which might not be true for the original P_i . In general, a plan to accomplish more goals will take more effort to monitor to see if it needs adaptation and will need adaptation more frequently. These arguments apply to non-linear planners to some degree as well.

2. P_j does not exist for some $j > 1$. The savings in this case depend on the implementation of the planner. The planner could fail to return a plan for G even though $P_1 \dots P_{j-1}$ (and possibly P_i , where $i > j$) exist. If so, the planner will waste at least $\sum_{i=1}^j E_i$ effort. A more forgiving planner might be guided by a heuristic that returns a plan to achieve as many G_i as possible. Ignoring the possibility that this might involve an expensive search in the space of possible goal orderings, the effort savings would be the same as in case 1.

3. P_1 does not exist. Once again, the ramifications depend on the implementation of the planner. If the planner does not return a partial plan for G , the amount of effort expended will be exactly the same, E_1 . If it returns a plan to achieve as many G_i as possible, it will expend more effort but will reduce the idle time of the agent.

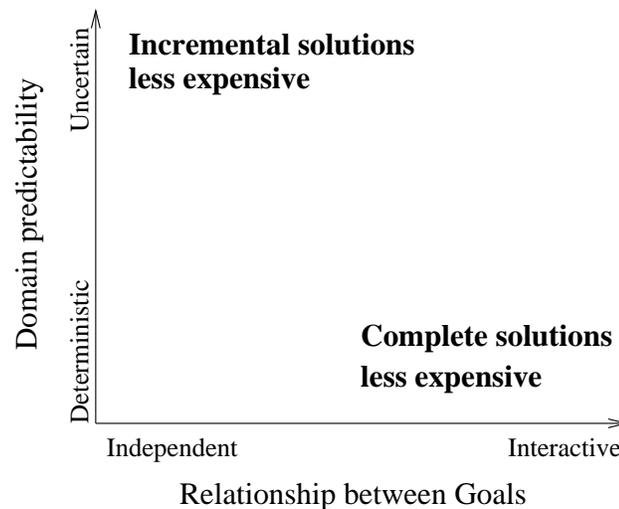


Figure 2.2: Graphical depiction of the relationship between computational expense and regions of domain uncertainty and goal interactions.

If the goals are not independent, the computational advantages to the alternate approaches is domain-dependent. A complete theoretical solution will require less effort when the goals interact and the actions that make up the plans rarely fail. Figure 2.2 gives a graphical depiction of the relationship between domain and goal characteristics and computational efficiency.² The figure is an implicit cost-benefit

²There are other factors which are relevant (for example, the extent to which previous planning information can be re-used when re-planning will depend on the planner design as well as characteristics of the domain), but the two axes shown are the most important.

analysis: the extra cost of developing complete solutions increases in more uncertain domains and the benefit of complete solutions lessens as goals interact less. Reactive architectures, such as those described by Firby (1987) and Brooks (1991), reflect the belief that the amount of uncertainty faced by an agent embodied in the real world is too great to even plan at all. An intermediate position is to select an action from a set of situation-specific choices based on a built-in notion of priority (Agre and Chapman, 1987).

Delay communication. The notion that one should not commit oneself too far ahead has a major impact on agent design. The agents must be designed to use communication as a primitive action to be executed at run-time. Planners that use communication during planning are therefore not appropriate. It is possible that if a single agent fails, these planners would involve the whole community in re-planning and re-communicating. And the more information that is exchanged, the higher the chance that the resulting plan will not match the circumstances that exist when the agent community starts acting due to the increased delay in starting to act.

The arguments in favor of delaying communication as much as possible are similar to the arguments above. In short, when domains are dynamic and uncertain, communication about future potential coordination points is more likely to end up having no value. And, when there are fewer goal interactions, there is less to be gained by discussing longer-range arrangements.

2.5 Formal specification

A general description of this architecture can be given by the triple of sets:

$(\mathcal{A}, \Sigma_D, \Sigma_C)$, where:

$\mathcal{A} = \{A_i \mid i \in [1, n]\}$ is the set of agents,

Σ_D is a common vocabulary to describe the domain D , and

Σ_C is a common set of communicative primitives.

The internal architecture that controls agent $A \in \mathcal{A}$ is private and can vary from agent to agent; for example, a multi-agent system could contain one member that was guided by case-based planner, one that was guided by a plan-space planner, and one agent with a decision-theoretic planner. Agents are not required to engage in communication, although in the domains of interest to this thesis, they need some help from other agents and communication is the only guaranteed way to get it. If they do engage in communication, they need to provide communication that is restricted to $\Sigma_D \times \Sigma_C$. In other words, considered as a “black-box,” agent A must have communication interface processes IN and OUT as depicted in Figure 2.3.

An individual agent A is assumed to have an internal representation for state, $REP_A(S)$, which is derived from the sensed state S . The agent may have other internal data structures for beliefs, desires, intentions, accumulated reward, interaction history, *et cetera* that help to guide the agent’s behavior. At this level of description, though, these structures are irrelevant. What is relevant is that A provides the following functionality:

A request-generation process $OUT_A : Range(REP_A) \rightarrow \Sigma_D \times \Sigma_C$. In other words, the agent must have a means of translating from their own internal representation of goal states into the common goal vocabulary and the com-

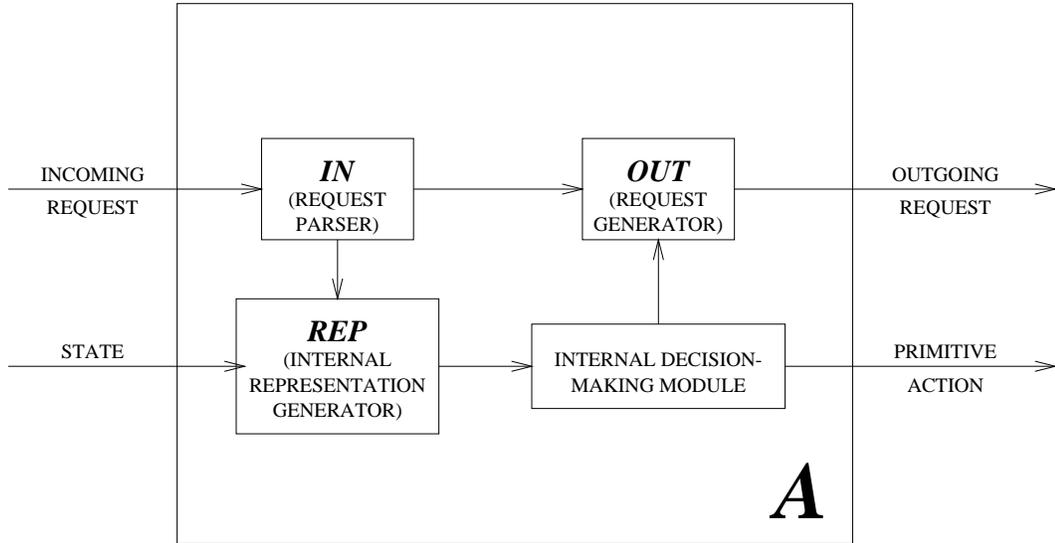


Figure 2.3: Agent communication interfaces.

mon communication language. For example, if A were a decision-theoretic planner-based agent in MOVERS-WORLD that needed assistance to move to `pre-defined-state-1210`, A would have to map that into something akin to `((HOLDING-TOGETHER A OTHER-AGENT BOX85),HELP)`.

A response-generation process $IN_A : \Sigma_D \times \Sigma_C \rightarrow \Sigma_D \cup \{\emptyset\} \times \Sigma_C$ that generates a response to an incoming request. For cooperative, rational agents this process would be broken into two phases, one that converts the request into an internal representation so that A can decide whether or not to agree to it, and a second that converts the decision into an appropriate response for the request.

A requirement to limit conversation duration is $Range(OUT_A) \cap Range(IN_A) = \emptyset$. This means that responses will not look like original requests, so that each agent makes at most one proposed course of action in a dialog. This does not preclude the agents from having another dialog immediately after the end of this one.

In addition to Σ_D and Σ_C , the semantics of all communicative primitives, i.e., $c \in \Sigma_C$, are commonly known. For example, the semantics of the minimal Σ_C needed to solve MOVERS-WORLD problems is given below³ (A is the requester, B the responder, and $R \in \Sigma_D$ is the request):

1. $c = \text{HELP}$ is defined to mean A desires B 's assistance to achieve R .
2. $c = \text{NO}$ is defined to mean B does not intend to help A at this time.
3. $c = \text{READY}$ is defined to mean that B is willing to immediately attempt the requested joint action associated with achieving R .
4. $c = \text{WAIT}$ is defined to mean B intends to help achieve R , but that A must wait for an unspecified length of time.

In principle, A or B might be deceptive, but in such cases it is not possible to guarantee that the community of agents will be able to solve the entire range of problems within the scope of their combined execution abilities. In practice, such situations have not been investigated in the deployed MOVERS-WORLD system.

Σ_D can provide either extensional or intensional state descriptions of the various objects $\in D$. Ideally, the choice of language should not influence the possible planning architectures available to the agents, but this may not be achievable in practice. For MOVERS-WORLD, Σ_D proscribes lists of predicates from a simple predicate language that can be described by the following formal grammar:

³Section 6.4 goes through the language in detail.

$$\begin{aligned}
\textit{request} &\longrightarrow \textit{open pred request close} \mid \lambda \\
\textit{pred} &\longrightarrow \textit{open 'ON' token token close} \mid \\
&\quad \textit{open 'READY-TO-LOAD' token close} \mid \\
&\quad \textit{open 'READY-TO-UNLOAD' token close} \mid \\
&\quad \textit{open 'TILTED' token close} \mid \\
&\quad \textit{open 'CLEAR' token close} \mid \\
&\quad \textit{open 'AT' token token close} \mid \\
&\quad \textit{open 'HANDEEMPTY' token close} \mid \\
&\quad \textit{open 'HOLDING' token token close} \mid \\
&\quad \textit{open 'HOLDING-TOGETHER' token token close} \\
\textit{token} &\longrightarrow [A-Z][A-Z0-9]^* \\
\textit{open} &\longrightarrow '(' \\
\textit{close} &\longrightarrow ')'
\end{aligned}$$

The above specifies that any alphanumeric string starting with a letter (treating all characters as uppercase) is a valid token. The presumption is that the token strings should refer unambiguously to items in the environment (e.g., TRUCK2), but in point of fact any token string is valid. If it is a nonsense string, then B will not be able to satisfy the request regardless of her abilities or attitude. Along the same lines, the predicate name and number of arguments do not have to be rigidly limited to those listed above. They are listed in that manner to portray what are realistic requests in MOVERS-WORLD. However, if A made a request of ((ON BOX1 BOX2 BOX3)) or ((UNKNOWN A B)), B would be able to parse the request, but once again would only be able to agree if B had built-in inference rules to process ON/3 or UNKNOWN/2.

The final piece to the architecture is a general characterization of the output of an agent. In other words, when should the agent attempt primitive action or communicative action. In theory, the community might include agents that do not follow these rules, but such situations have not been studied in practice. The following specifications state that communication should not occur until the agent has reached a coordination point, and that the corresponding request should be about the coordination point.

There are several general situations in which the action selection of the agent is not an issue. An agent might be in the middle of attempting a primitive action. In such a case, there is no action selection for the agent to make until the attempt is over. Or the agent might be in the middle of an ongoing dialog, in which case, the output of the agent is determined by their response-generation process IN_A until the end of the dialog. Finally, the agent might be idle, either because they have no more unmet goals and they were unable to construct a plan to accomplish one of the unmet goals.

Action selection is then an issue when the agent has a plan directing their behavior, and is not in the middle of a previously selected action. In that case, let S_A be A 's current sensed state, $step$ be the next primitive action in A 's active plan, and \mathcal{S} be the set of states (in A 's internal representation) that A believes are the possible states from which to execute $step$. If $S_A \in \mathcal{S}$, then $output(A) = attempt(step)$. If $S_A \notin \mathcal{S}$, then select an $s \in \mathcal{S}$ such that $d(s, S_A) = \min_{s \in \mathcal{S}} d(s, S_A)$, where d is a domain-dependent distance function that measures the difference between two states. Then $output(A) = OUT_A(s)$. This formalism states that if A does not know how to get into a state enabling further actions, only then will A make a request for assistance.

Also, in order to ensure that the community solves problems which their combined execution abilities warrant, agents are assumed not to be lazy in the following sense.

If A is idle, then $IN_A : \Sigma_D \times \Sigma_C \rightarrow \emptyset \times (\Sigma_C \setminus \{\text{NOT-WILLING}\})$. $\{\text{NOT-WILLING}\}$ is a place-holder, representing all negative responses to requests A is capable of. In English, this means that an agent agrees to help whenever possible if she is idle.

2.6 Related research

The AI learning model of this thesis bears a relationship to a cognitive model of convention in joint activity (Alterman and Garland, 2001). It parallels the emergence of convention in circumstances where no ruling body of knowledge developed by prior generations of agents within the community exists to draw on as potential designs for cooperative and coordinated behaviors. This theory of convention in joint activity is part of a model of everyday reasoning and behavior (Alterman *et al.*, 1998b; Alterman, 1999). Convention helps to coordinate behavior among agents and is defined as a design for the activity that develops from practice within a community of agents. In the everyday world, there are few (if any) actions that are not steeped in convention.

Agents must constantly be internalizing, reasoning about, and muddling through on the basis of convention. During the give-and-take of activity, participants reason with a bias that at one point or another their expectations about points of coordination will be met. There is also the understanding that because of uncertainty, interruptions, and numerous other opportunities to get off-track and out-of-synch, the participants must work continuously and jointly to achieve conventional coordination. The expected points of coordination collectively form a *design for the activity* — the convention — but not a complete specification of it. Different agents bring to bear different knowledge and experience. Gaps in behavior are filled by the historically conditioned constraints, affordances, and patterns of coordination available to the participants as they proceed with their joint activities.

With practice, individuals within the community begin to converge on a set of conventionalized behaviors that best match the regularly occurring problems of coordination in the domain of activity. One feature of the model is that the community improves its performance despite the fact that individual agents reason independently about their experiences. Because the agents use prior experiences to guide future behaviors, they can, however, indirectly adjust their expectations about the best way to coordinate behavior by making adjustments for the success and failure of this or that act. Another important feature of the model is that the mechanisms for improving behavior are tied to the memory function of individual agents.

There are other models for describing interacting groups of agents besides Clark's model of joint activity. In general, these models specify additional mental attitudes that all individuals must have. These attitudes are belief, desire, and intentions and represent, respectively, the information, motivation, and deliberative states of the agent (Rao and Georgeff, 1995). Three prominent models specify the attitudes needed for the group of agents to be considered to be engaged in a *shared cooperative activity* (Bratman, 1999) or a *collaboration* (Grosz and Sidner, 1990) or to have *joint intentions* (Levesque *et al.*, 1990). These works thus emphasize the distinction between plans as recipes and plans as complex mental attitudes (Pollack, 1990).

Shared cooperative activities are, in essence, joint activities with a trio of characteristic features:

1. Participants are to committed to the joint activity.
2. Each actor attempts to be responsive to the intentions and actions of the others.
3. Each actor is committed to supporting the efforts of the others to play their role in the joint activity.

Bratman (1999) specifies that the actor's sub-plans must mesh (i.e., there is a way to accomplish a shared cooperative activity without violating the goals of the sub-plans) and the actors intend to have meshing sub-plans. This requirement is not as stringent as it first appears. First, actors may neither know nor care about the details of other actors' sub-plans, as long as the actor knows the intended goal of the sub-plan. Also, actors may negotiate over the best way to achieve meshing sub-plans even while they are undertaking the activity.

From the perspective of collaboration, each occasion of group interaction is managed through the ongoing construction of a shared plan (Grosz and Kraus, 1996, 1999). Shared plans may be partial and can be revised as the activity continues. For a group of collaborators to have a full shared plan there must be a mutual belief that they are committed to the success of the action and a mutual belief of the need to accomplish each of the sub-actions. A key idea is that even when they only have a partial shared plan to achieve a given sub-action — collaborators only partially know what they are going to do — they always have a full shared plan to work out the details of the partial shared plan. A unique shared plan is constructed on each occasion of joint activity.

Since the plans of individual MOVERS-WORLD agents are partial and are revised as the activity continues, in part due to beliefs about other agents, the community activity in MOVERS-WORLD bears some relation to the development of a shared plan. The critical difference between a shared plan and a coordinated procedure is that a shared plan is a construct that is tied to a single episode of joint activity and a coordinated procedure is a construct that is tied to the history of related joint activities. In other words, if the unit of measure is a single episode of joint activity, the shared plan is a synchronic analysis of the behavior of the participants and learned coordinated procedures are a diachronic one. The utility of the coordinated

procedures is measured by a reduction in the amount of work needed by participants in a joint activity to develop a shared plan to achieve a common goal during the course of actually achieving the goal. Over time, through practice at working at a recurring problem of coordination, certain points of coordination become expected, and a shared plan that reflects learned regularities in the domain is developed more easily. A shared plan realizes domain regularities only after a community of agents have worked together, over many episodes of joint activity, to achieve similar goals in similar contexts.

Joint Intentions (Levesque *et al.*, 1990) is a formal description, formulated in a modal language, that specifies the necessary commitments and intentions required for joint action by a team. In their view, joint action involves more than just the union of individual actions, even when those actions are coordinated. Instead, each agent must be aware of, and care about, the status of the group effort as a whole in order to guarantee the success of the collective effort. As part of their joint commitment to a goal, agents ensure that mutual belief is maintained at all times by communicating when they discover privately that a goal is true or unachievable. Also, communication is used by all agents who have a joint intention to do an action throughout execution of that action in order to make sure that initial mutual belief does not dissipate over time.

STEAM (Tambe, 1997) is an alternate implementation of joint activity. The theory underlying the system is principally joint intention theory, but some aspects of the system borrow from SharedPlans. The communication load placed on the system by these theories, however, proved to be burdensome. In order to get reasonable system performance, the agent architecture was modified to do selective communication based on the agent's estimate of the utility of communicating each message. Two other notable differences from MOVERS-WORLD is that STEAM agents, as

proscribed by joint intentions theory, participate in joint activities solely through explicit team operators, which all agents know, and that STEAM is not a learning system.

Another approach to selecting information to communicate is given in Turner (1994). In this work, the problem-solving knowledge of the agent is analyzed to heuristically determine a *usefulness rating*. The suggested heuristics are derived from knowledge that is available and easily accessible during problem solving; agents are thus able to compute usefulness ratings with little additional computational effort. Information is communicated if the rating exceeds “some threshold.” An interesting byproduct of this approach is that it supports volunteering information even in the absence of knowledge about the other agents’ plans and reasoning.

Chapter 3

Overview of the architecture

The last chapter presented arguments, both philosophical and practical, for how a model to support joint activity should be designed. This chapter presents a framework that meets those design criteria: agents have minimal common knowledge designed into them, communication at and about coordination points forms the basis for coordinating activity at run-time, and learning techniques are integrated into the agents so that performance will improve over time. The last point is important enough to be treated separately in the next chapter.

This chapter begins with a (re-)introduction of MOVERS-WORLD. Next, the cycle of activity for learning agents is described, both from a theoretical standpoint and in terms of the implemented architecture. The third section describes the baseline planner, which is guided by probabilities independently estimated by the agents. The following two sections then discuss the essentials of how communication and coordination is implemented. The chapter closes with a section that shows the impossibility of optimal performance in MOVERS-WORLD and the difficulty of achieving near-optimality.

3.1 MOVERS-WORLD

The overall control structure for the community of MOVERS-WORLD is not hierarchical nor is planning centralized. In other words, agents do not share an overarching plan (Georgeff, 1983; Ephrati *et al.*, 1995) or reason explicitly about group concepts (Tambe, 1997), rather cooperation emerges from local interactions. The capacities of other agents are not given (Genesereth *et al.*, 1986), but this information is acquired through experience, as will be discussed in the next chapter.

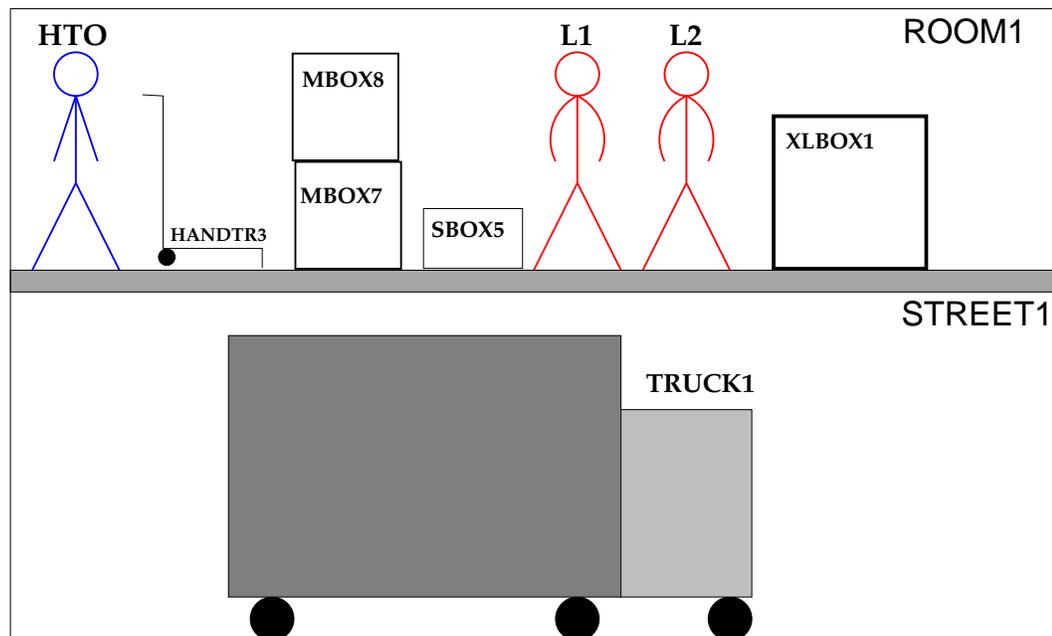


Figure 3.1: Sample MOVERS-WORLD problem.

MOVERS-WORLD has two types of agents: hand-truck operators (such as HTO in Figure 3.1) and lifters (L1 and L2).¹ Depending on their type, different agents

¹For the examples and experiments in this work, there are always two lifters, labelled L1 and L2, and one hand-truck operator, HTO. However, this is not an inherent part of MOVERS-WORLD problems; the system solves problems involving more lifters and/or more hand-truck operators.

have different actions that they can perform. The type-specific actions for the agents in MOVERS-WORLD are:

lifter LIFT, LIFT-TOGETHER, CARRY, CARRY-TOGETHER, LOAD, LOAD-TOGETHER, UNLOAD, UNLOAD-TOGETHER, PUT-DOWN, and PUT-DOWN-TOGETHER.

hand-truck operator PUSH-HANDTR, TILT-HANDTR, and STAND-HANDTR.

all agents MOVE, WAIT, and SIGN.

Agents themselves do not know and cannot perceive the types of other agents or even of themselves; in fact agents have no representation for the concept of type! Instead, an agent merely knows the set of operators she is capable of and her set of attributes. The duration of actions varies from 20 to 40 ticks (of a simulated clock); conversation durations vary depending upon the content of the dialog. Conversations do not interrupt primitive actions. There is no common clock, so agents cannot make requests like "In 75 ticks, please do X. or respond" "I will do that for you; it will take 112 ticks."

MOVERS-WORLD contains three types of objects: boxes, hand-trucks, and a single moving truck. Boxes come in four sizes: extra-large (XLBOX), large (LBOX), medium (MBOX) and small (SBOX). Large and extra-large boxes require two lifters to lift, and extra-large boxes are too unwieldy to be carried. Hand-trucks can hold one large or extra-large box or any combination of two smaller boxes. Hand-truck operators are not capable of handling boxes directly so loading and unloading a hand-truck requires the cooperation of at least one lifter.

The domain features that determine whether actions are possible are object size, object weight, agent strength, and item capacities. None of these are observable features of the environment. The static observable box features are height, width,

depth and material. For boxes, the size is derived from the volume of the box and the weight is a linear function of the size and material. The label associated with a box includes a S, M, L or XL solely in order to make it easier for the human reader to parse the system output. Item capacities specify how many other items and how much total weight the item can hold.

3.2 Cycle of activity

The learning techniques of this work are built on top of an underlying architecture that supports joint activity with or without learning. When the agents are learning, improvement in the performance of the community occurs both during the course of a single problem and over several generations of problems. For each activity considered in isolation, the problem-solving process of the community can be broken down into the four stages shown in Figure 3.2.

-
1. A community of agents is given a set of top-level goals to achieve.
 2. The community engages in pre-activity analysis as time allows.
 3. Activity *per se*:
 - Commences when any agent interacts with the world.
 - Asynchronously, each agent:
 - (a) when active, creates or adapts a plan as needed.
 - (b) when has a plan, attempts the first action in the plan until the action has ended, possibly unsuccessfully.
 - Terminates when all agents believe their goals have been achieved.
 4. The community engages in post-activity analysis as time allows.

Figure 3.2: Stages of a joint activity.

The term “analysis” in Figure 3.2 includes many possible behaviors by the agent. The rest of this section addresses both the range of possibilities and the choices made in the implemented system.

Pre-Activity analysis occurs before action commences (stage 2). In general, agents may exchange planning information, task decompositions, or resource requirements. Early distributed planners (e.g., Corkill, 1979) did all planning / communicating during this stage; this reflects a (then-standard) plan-then-execute model of behavior that did not integrate planning with execution. If there are irreversible actions in the domain, it is essential that pre-activity analysis ensure they are only undertaken in the proper contexts (e.g., Durfee *et al.*, 1998).

In MOVERS-WORLD, the only pre-activity process undertaken does not involve any communication; each individual uses the current problem setting to partially instantiate memories of past experiences in order to make them more readily deployable during the activity proper.

Post-activity analysis occurs after all agents believe their goals have been achieved. In general, agents can exchange local perspectives about how the activity transpired in order to form a global perspective. Also, agents can take advantage of the time between activities to reorganize knowledge, for example recluster- ing classification trees. Or agents can analyze their actions in the context of a completed activity in order to identify mistakes to be avoided or new ways to achieve goals.

In MOVERS-WORLD, the only post-activity process undertaken does not involve any communication; each agent independently stores coordinated procedures into memory. Coordinated procedures reflect new ways to solve goals and

are derived from portions of run-time behavior that were essential to past solutions. In future episodes, these procedures are recalled, and further improvements can occur. Thus, regularities in the coordination of behavior in joint activities develop over time, and the behavior of the community will evolve and improve.

3.3 Planning

MOVERS-WORLD presents a challenging domain. The challenge is not in handling typical planning hurdles, such as deep search trees with large binding factors; rather, MOVERS-WORLD agents must coordinate their actions in an environment in which they have limited information about, and control over, other agents and the world.

An agent can create a plan from scratch using a given set of STRIPS-like operators in a hierarchical planner. Each agent maintains a representation of the probability of success for different actions in different contexts. When planning from scratch, the agent uses these probabilities to guide it through the search space. This information represents some of the expectations that the individual agent has about her own capabilities and those of her co-participants. The probabilities are incrementally updated, so the agent develops more realistic expectations during the course of action.

No explanations accompany failing actions. For example, when L2 fails to lift a large box, she is not informed that this is because the box is too heavy for her to lift alone. So, there are no causal structures available for future planning sessions to prevent L2 from attempting to lift that box by herself again. The failure is manifested only in L2's estimates of the likelihood of successfully executing the lift action again in the future. Section 6.1 explains how the baseline planner uses probabilities, and Section 6.2 recites how operator probability trees are used to generalize this experience and make L2 less likely to lift any large boxes in the future.

An agent can determine whether or not an action succeeded or failed by seeing if the expected effects of the operator hold. In a more general approach along the same lines, an agent could learn both a set of execution-time effects for each operator and learn the associated probability of the set being the outcome of an attempt. Goodman (1993) explains failures by inducing features sets in another straight-forward application of the information available by observation. By contrast, the process of generating a predicate-based explanation (e.g., Hammond, 1990; Mitchell *et al.*, 1986; DeJong and Mooney, 1986) for a failure seems complex. Perhaps an agent that got wet unexpectedly did so because they left their umbrella at home. Or perhaps the umbrella failed to open. Or the wind inverted the umbrella. Or the agent stepped in a puddle. Or was hit by a water balloon. The generation process appears as difficult and open-ended as the planning process it is meant to improve.

The independence of MOVERS-WORLD agents yields a rich variety of run-time behaviors. At any given point in time, an agent's knowledge of the external world is her perceivable environment and a map of the world that she constructs as she goes along. In general, activity may not go as expected for a variety of reasons:

1. An agent may attempt a primitive action which fails because of the agent's lack of omniscience about the domain.
2. An agent may be interrupted by a request from another agent. In this case, the agents have a conversation. During the conversation, agents may choose to suspend their current goals and plan to accommodate a request. Alternatively, an agent may have to abandon her current goals and plan because the other agent cannot or will not provide the needed assistance.
3. There may be a resource conflict among agents, i.e., more than one agent tries to manipulate the same object.

4. An agent's attempt to communicate with another agent may be delayed since the other agent is already engaged in communication.

For these reasons, agents will frequently have to adapt, replan, or suspend the current activity. When a plan is completed or abandoned, the agent selects one or more of the unmet top-level goals to actively work on. A plan is created for a set of goals by either selecting an old plan from memory or creating a plan from scratch. Old plans are stored somewhat abstractly so they require some refinement before they are deployed. The plans in memory are extracts of execution traces, which are the result of multiple planning sessions occurring at various times during the activity. This technique differs from standard procedural learning techniques based upon the output of a single planning session (Carbonell, 1983; Veloso and Carbonell, 1993; Laird *et al.*, 1986).

3.4 Communication

Solutions to MOVERS-WORLD problems, and the techniques used to learn coordinated procedures by MOVERS-WORLD agents, are shaped by the fact that communication is the central mechanism for cooperation and coordination. Two agents are said to “cooperate” if they work together to achieve some common purpose and they are “coordinated” when their individual actions are appropriately ordered to support cooperative activity. In this framework, communication is the only mechanism whereby agents can determine if they are cooperating. In other words, there are no global structures, such as blackboards, for agents to use to determine if they happen to be working on the same goal. Each MOVERS-WORLD problem-solving episode includes some goal(s) that can only be solved by cooperating agents, so communication is an essential part of the community activity.

Communication occurs at and about coordination points. Communication is used to attempt to establish cooperation when an agent is unable to achieve a coordination point on her own. Cooperation is not guaranteed during communication since agents have their own decision-making strategies; moreover, even if an agent is willing to cooperate, she may be incapable of helping. An agent who is unwilling or unable to assist can propose an alternative that the original requester now contemplates adopting. That is the extent of negotiations that can occur during a single dialog. Each agent presently uses the same strategy for deciding whether to cooperate: preference is given to the plan that achieves her top-level goals in the most time-efficient manner.² Despite their common strategy, agents will make different decisions because of differences in their experiences.

Communication between two agents in this system is a peer-to-peer connection, similar to a telephone conversation, rather than a broad- or multi-cast. Agents do not have to be in the same location to engage in communication. One agent calls another and either establishes a connection, gets a busy signal or the other agent does not respond. Once a connection is established, communication is handled via request and response frames. After agreeing to cooperate, an agent can opt-out at any time, without obligation to notify other agents. This differs from the commitment level required in collaborations as specified by Grosz and Sidner (1990) and Levesque *et al.* (1990).

Communication is considered an action in its own right, so it occurs at run-time and not as a sub-process to the internal planning process unlike many distributed, continual planning systems. Agents limit themselves to single requests; transmitting single requests lowers both communication costs and plan merging costs, which can be considerable. The communication language is simple enough that it could be

²Appendix B.1 discusses cooperation strategies in detail.

mapped to KQML (Finin *et al.*, 1993; Labrou and Finin, 1997) or comparable other agent languages, so agents could be constructed using development tools such as JATLite (Jeon *et al.*, 2000).

There are three initial request types and eighteen responses, twelve of which make an alternate request to the original caller. In practice, not all of the potential sequences of request and response types are manifested; for the 1,800 trials that form the basis of the learning curves presented in Section 5.2, there were 300,923 conversations comprising 94 different sequences. 285,669 conversations, in 47 different sequences, contained a single request and 15,254 conversations, in 47 sequences, contained more than one request. 47% of all conversations were requests for joint action that the responder was ready and willing to attempt. See Table 6.3 for full details.

3.5 Coordinating joint activity

Cooperation is not guaranteed during communication since agents have their own decision-making strategies. In other words, agents might refuse to assist when they should, or might agree to help when they should not. Even if an agent is willing to cooperate, she may be unable to construct a plan to do so; an agent who is unwilling or unable to assist can propose an alternative that the original requester may now contemplate adopting. Even when they use the same decision-making policy, agents will make different decisions because of differences in their experiences, their assessment of the situation, and their current plan and goals.

Communication and meta-planner wait operators jointly form the mechanism whereby joint activities are coordinated. The agents are not assumed to possess sufficiently detailed knowledge of each other to coordinate without communication (Genesereth *et al.*, 1986); thus, agents cannot coordinate exclusively by observation

(Huber and Durfee, 1995) or plan recognition (Huber and Hadley, 1997). When cooperation is first established during communication, the requester will be told if she has to wait. If so, a wait operator is pushed onto her plan. The agent will continue to wait until she is told otherwise or the subgoal becomes true or she has waited too long (details are given in Chapter 6).

An obvious alternative to this method of coordinating is for a wait response to include information indicating how long the requestee thinks the requesters will have to wait. Then the requester might be able to attempt actions in the interim. However, this requires that agents have compatible measures of time, which is not assumed. In practice, there is a global measure of time that all agents and the system use. In theory, there is no dependence on global time in the techniques presented to support activity, with or without learning. One agent might internally record time in ticks elapsed since the start of the problem and another record time in seconds elapsed since 1972.

Figure 3.3 shows an example of two MOVERS-WORLD agents successfully coordinating their actions. Such a situation is unusual when agents are novices; in past experiments, more than half of the requests are refused and, further, agents fail to accomplish agreed-to requests on the first try more than half of the time. As shown in the figure, time is measured in ticks. The duration of actions varies from 20 to 40 ticks; conversation durations vary based upon the content of the dialog.

Communication and a meta-planner operator jointly form the mechanism whereby joint activities are coordinated in MOVERS-WORLD. When cooperation is first established during communication, the agents must determine how they will coordinate the cooperative activity. Sometimes, nothing need be done — for example, if two lifters are both adjacent to a ready-to-be-lifted box and both are ready to lift it. More often, though, the requester will idle for one or several time steps — for ex-

```

Ticks    1 to   15: HTO and L1 converse
          "L1, would you help me achieve (ON PR00-MBOX7 HANDTR3)?"
          "HTO, I'll help, but you'll have to wait a bit."

Ticks    16 to   45: <LIFT PR00-MBOX7> by L1 successful

Ticks    46 to   80: <LOAD PR00-MBOX7 HANDTR3> by L1 successful

Ticks    81 to  100: <TILT-HANDTR HANDTR3> by HTO successful

```

Figure 3.3: Simple example of coordinated agents

Lifter L1 agrees to load medium-sized box MBOX7 on a hand-truck for hand-truck operator HTO. This will allow HTO to move the box to the street, where HTO will then ask another agent to load MBOX7 onto the truck. Since L1 does not know HTO's plan or abilities, L1 is unaware that this is what HTO intends.

ample, if the requestee lifter is not currently ready to lift the box. Another common situation is when a hand-truck operator idles after a lifter agrees to load a box onto a hand-truck. Idling is presently implemented by adding a WAIT to the beginning of a plan. While the WAIT is at the beginning of her plan, an agent is waiting for one of two events to occur: communication indicating that joint action can occur (e.g., the other lifter now indicates she is ready to act) or the completion of her request (e.g., the box appears on the hand-truck). If an agent is idle too long, she will become frustrated, as defined in Section 6.3.1, and inquire about the status of her request, possibly discovering that the other agent has opted out.

The following two examples show trace snippets involving cooperation in unloading a box from a hand-truck and loading it onto the truck. The examples are actual output of the system (with unrelated actions trimmed), but they are illustrative, not prototypical; empirically, agents agree to cooperate less than half of the time when

they have different goals. In Figure 3.4, L1 happens to have the same goals as HTO before conversing; in Figure 3.5, she does not. In both cases, HTO will add a WAIT operator to her plan as a result of the conversation with L1.

```

Ticks 152 to 176: <MOVE PR24-STREET1> by L1 successful
Ticks 177 to 189: HTO and L1 converse
    "L1, would you help me achieve (ON PR24-MBOX0 TRUCK3)?"
    "HTO, I'm already working on it!"
Ticks 190 to 224: <UNLOAD PR24-MBOX0 HANDTR3> by L1 successful
Ticks 225 to 259: <LOAD PR24-MBOX0 TRUCK3> by L1 successful

```

Figure 3.4: Coordination based on first-principles: same goals.

As Figure 3.4 shows, when L1 and HTO have the same goals, their behavior is fairly efficient because L1 moves to the street before HTO makes her request. However, this kind of timing is exceptional. In this case, HTO's request is superfluous, but there is no way for the hand-truck operator to know this beforehand. A more common case, when the agents have different goals, is shown in Figure 3.5. As mentioned above, there is no guarantee L1 will agree to help in this situation.

```

Ticks 131 to 150: <STAND-HANDTR HANDTR3 PR24-STREET1> by HTO successful
Ticks 151 to 163: HTO and L1 converse
    "L1, would you help me achieve (ON PR24-MBOX0 TRUCK3)?"
    "HTO, I'll help, but you'll have to wait a bit."
Ticks 164 to 188: <MOVE PR24-STREET1> by L1 successful
Ticks 189 to 223: <UNLOAD PR24-MBOX0 HANDTR3> by L1 successful
Ticks 224 to 258: <LOAD PR24-MBOX0 TRUCK3> by L1 successful

```

Figure 3.5: Coordination based on first-principles: different goals.

3.6 Impossibility of optimal performance

The distribution of problem-solving knowledge among a group of independent agents makes achieving optimal run-time behavior extremely unlikely. This section will demonstrate this fact by addressing a one-box problem and discussing the difference between plans a lifter can generate and plans a hand-truck operator can generate. By expanding the problem to include more boxes, the difference in planning ability makes achieving optimal behavior without learning impossible and achieving near-optimal behavior unlikely.

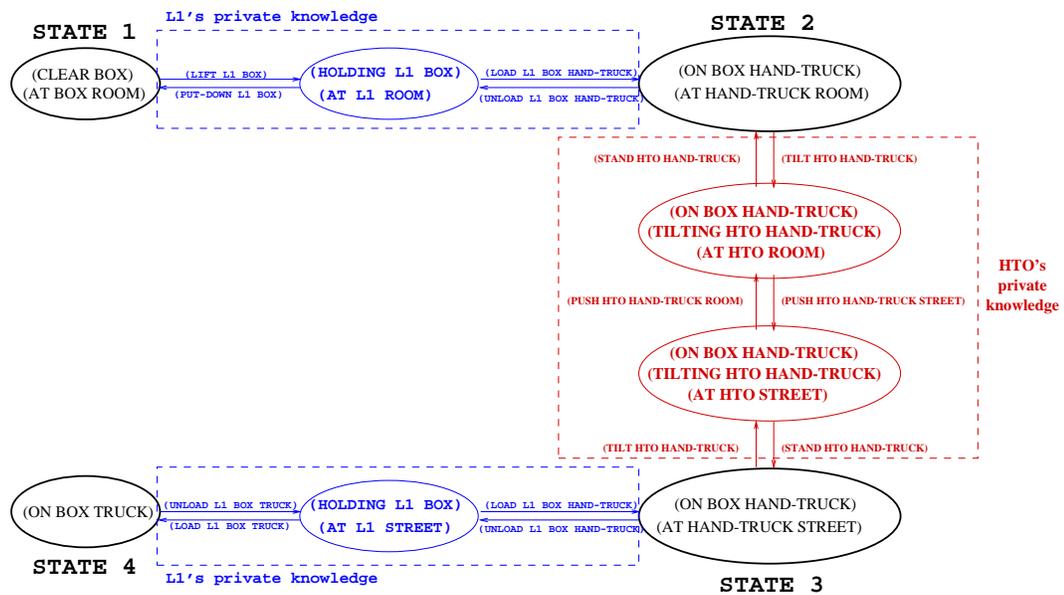


Figure 3.6: Distribution of problem-solving knowledge in MOVERS-WORLD.

A solution path for the simplest situation where a lifter and a hand-truck operator work together to get a box onto the truck is shown in Figure 3.6. Over the course of the solution, the box “moves” through eight different states (configurations), each of which is marked in an oval. Only L1 knows how to transform the box from state 1 to state 2. Only HTO knows how to get from state 2 to state 3. And only L1 knows how to get the box from state 3 to state 4 (the goal state).

L1 cannot generate a plan to match this solution path on her own since L1 has no built in knowledge of the hand-truck or of HTO's capabilities. Backward chaining can identify state 3 as a precursor to state 4 and forward-chaining can identify state 2 as a successor to state 1. However, nothing logically connects them for L1's point of view. In other words, L1 cannot distinguish the pair of states (state 2, state 3) that are relevant to the solution path from the many other pairs of states that are not relevant.

The plan that L1 can generate is to bypass states 2 and 3 altogether and carry the box to the street directly. (HTO cannot construct this plan.) This is, in fact, more time-efficient than working with the hand-truck operator if the entire problem is to move just that one box. However, optimal performance can rely on working with the hand-truck operator if more boxes are involved. Consider the ramifications of this distribution of knowledge in terms of the two agents getting two identical boxes onto the truck in as little time as possible. Optimal performance is unachievable since it involves no communication whatsoever; it would consist of:

L1	HTO
<LIFT BOX1>	
<LOAD BOX1 HANDTR>	
<LIFT BOX2>	<TILT HANDTR>
<CARRY BOX2 STREET>	<PUSH HANDTR STREET>
<LOAD BOX2 TRUCK>	<STAND HANDTR>
<UNLOAD BOX1 HANDTR>	
<LOAD BOX1 TRUCK>	

If possible, this process would take 244 ticks. However, without communication, L1 would not construct a plan to load BOX1 onto the hand-truck! L1's expectation at the outset would be to move both boxes to the truck by carrying them; HTO's expectation is that both boxes would be moved via the hand-truck. Furthermore,

neither knows that the other is working on the same two goals. The closest to optimal that is possible is 272 ticks as shown in Figure 3.7. And achieving that requires that L1 and HTO make the same goal ordering choice to start and that L1 agrees to help HTO on both occasions and that L1 correctly adapts her plan. While none of these is unlikely independently, the chance of them all occurring is less likely. And, as the complexity of the problems increases by including other agents and increasing the number of boxes, there is an exponential increase in the number of decisions that must all be made “correctly” for the community to perform even this close (90%) to optimal.

```

Ticks    1 to   15: HTO and L1 converse
    "L1, would you help me achieve (ON BOX1 HANDTR)?"
    "HTO, I'll help, but you'll have to wait a bit."
Ticks   16 to   45: <LIFT BOX1> by L1 successful
Ticks   46 to   80: <LOAD BOX1 HANDTR> by L1 successful
Ticks   81 to  100: <TILT-HANDTR HANDTR> by HTO successful
Ticks   81 to  112: <LIFT BOX2> by L1 successful
Ticks  101 to  125: <PUSH-HANDTR HANDTR STREET> by HTO successful
Ticks  113 to  152: <CARRY BOX2 STREET> by L1 successful
Ticks  126 to  145: <STAND-HANDTR HANDTR> by HTO successful
Tick 146: HTO got no response when trying to contact L1. Letting ring ...
Ticks  153 to  167: HTO and L1 converse
    "L1, would you help me achieve (ON BOX1 TRUCK)?"
    "HTO, I'll help, but you'll have to wait a bit."
Ticks  168 to  202: <LOAD BOX2 TRUCK> by L1 successful
Ticks  203 to  237: <UNLOAD BOX1 HANDTR> by L1 successful
Ticks  238 to  272: <LOAD BOX1 TRUCK> by L1 successful

```

Figure 3.7: Near optimal coordination in simple MOVERS-WORLD problem.

Chapter 4

Overview of learning

Learning plays a key role in a framework to support joint activity among independent heterogeneous agents. Without learning, the lack of initial knowledge would make the quality of community performance too low for the framework to be practical. This chapter will describe the techniques that individual agents use to learn. The emphasis of this thesis is on learning new solutions, called *coordinated procedures*, from past successful joint endeavors. A coordinated procedure is defined as a sequence of individual actions separated by mechanisms (i.e., SIGNs and WAITs) that support the achievement of coordination points. However, this is not the only learning that occurs. Agents also use and maintain trees structures, called operator probability trees, as a means of learning which individual actions are likely to succeed. All learning is completely distributed; the agents learn on their own without communicating to each other.

The two different learning structures are used and updated at different times as shown in Figure 4.1. Coordinated procedures are identified only at the end of a problem, when the agent has time, and a better perspective, to reflect on the actions that contributed to her success. Once added to her memory, the coordinated

procedures are available when creating a plan from scratch in the future. By contrast, operator probability trees are updated incrementally during the activity after each action (both primitive and communicative) is attempted. Operator probabilities are used by the baseline planner and when adapting coordinated procedures to fit the current setting.

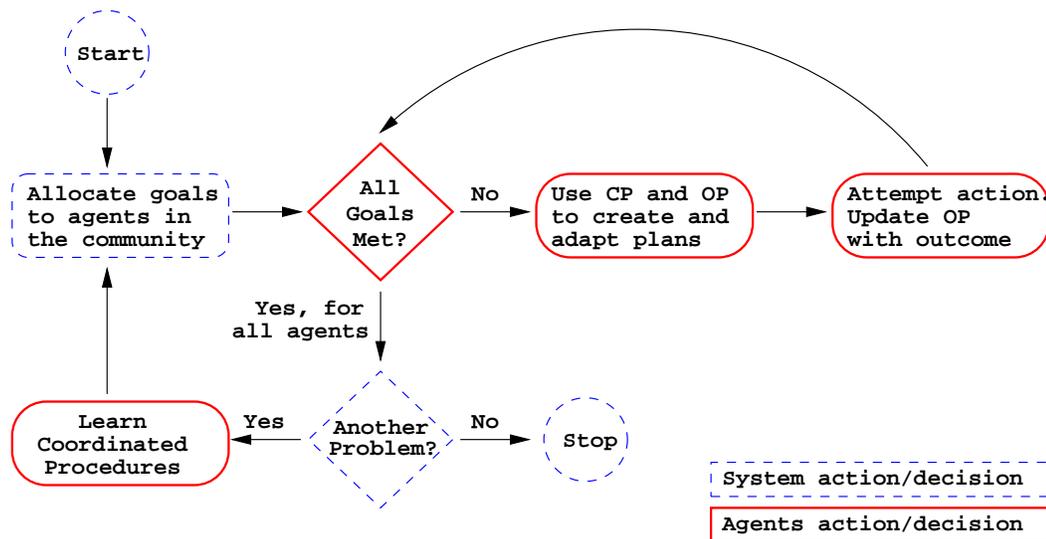


Figure 4.1: Flowchart of the community routine.

Coordinated procedures and operator probability trees enable the agents to learn essential coordination knowledge. Specifically, agents learn under what circumstances another agent is likely to ask for help, what is the nature of the help the other agent will need, who is capable of providing help, and how likely is the other agent to agree to provide that help.

Agents undertake a seven-step process to convert their successful run-time behavior into coordinated procedures. This chapter begins with a simple *MOVERS-WORLD* example that demonstrates how a lifter learns to expect to interact with the hand-truck and the hand-truck operator. The next section provides an overview

of the seven steps involved in learning coordinated procedures, each of which is described in detail in Chapter 7. Then, a more complicated example demonstrates how an agent learns a plan that efficiently interleaves goals. The subsequent section shows how the coordination mechanisms introduced into coordinated procedures effect communication and coordination. The chapter closes with a discussion of the learning model in this work and relates it to other work.

4.1 A new procedure learned from experience

The simplest plan that lifters learn that their first-principles planner does not construct is to load a box onto a hand-truck and later unload it and load it onto a truck at the behest of a hand-truck operator. This is not generated from first-principles since the lifters have no initial knowledge of the capabilities of, or actions available to, the hand-truck operator. This plan can get created from the following snippet of activity involving a medium-sized box.

First, a high-level description:

1. Hand-truck operator HTO asks lifter L1 to get box MBOX3 onto hand-truck HANDTR2. L1 agrees and does so via lifting and loading the box. L1 next fails in an attempt to lift large box LBOX2 by herself, does nothing for a tick since she has no plan, and lifts small box SBOX5 while HTO tilts, pushes to the street, and stands up HANDTR2.
2. HTO asks L1 to get MBOX3 onto truck TRUCK. L1 agrees, puts SBOX5 back down, moves to the street, unloads the box from the hand-truck and then loads it onto the truck.

L1 records her behavior internally each tick, including information about that tick's active goals, cooperation agreements, state of the world, attempted action, reason for attempting the action, and result of the attempt. These internal structures contain too much data to show fully, but the gist of them is clear in the following listing of the relevant portion of L1's execution trace:

```
(<agreed to achieve (ON MBOX3 HANDTR2) for HTO>
<executed (LIFT MBOX3)>
<executed (LOAD MBOX3 HANDTR2)>
<failed to (LIFT LBOX2)>
<executed (NO-OP)>
<executed (LIFT SBOX5)>
<agreed to achieve (ON MBOX3 TRUCK1) for HTO>
<executed (PUT-DOWN SBOX5)>
<executed (MOVE STREET)>
<executed (UNLOAD MBOX3 HANDTR2)>
<executed (LOAD MBOX3 TRUCK1)>)
```

This trace is “cleaned,” by removing the failed primitive action and the idle step (NO-OP). Then the lift and load of the small box are removed since they are part of a solution to solve a different top-level goal. Also, agreements are converted into WAIT-FOR-REQUEST actions. At this point, the trace is:

```
(<WAIT-FOR-REQUEST HTO (ON MBOX3 HANDTR2)>
<executed (LIFT MBOX3)>
<executed (LOAD MBOX3 HANDTR2)>
<WAIT-FOR-REQUEST HTO (ON MBOX3 TRUCK1)>
<executed (MOVE STREET)>
<executed (UNLOAD MBOX3 HANDTR2)>
<executed (LOAD MBOX3 TRUCK1)>)
```

Summarization of this grouping removes any action which is planner-reconstructible, leaving:

```
(<WAIT-FOR-REQUEST HTO (ON MBOX3 HANDTR2)>
<executed (LOAD MBOX3 HANDTR2)>
<WAIT-FOR-REQUEST HTO (ON MBOX3 TRUCK1)>
<executed (LOAD MBOX3 TRUCK1)>)
```

Finally, to prepare this group of actions for retrieval in the future, literals such as MBOX3 must be replaced by newly generated variables. Normally, a variable name would be something like ?T1524; the variable names shown are for readability. The procedure stored in memory, based on indices derived from the state of the world as of WAIT-FOR-REQUEST, is:

```
(<WAIT-FOR-REQUEST ?AGENT (ON ?BOX ?HANDTR)>  
<LOAD ?BOX ?HANDTR>  
<WAIT-FOR-REQUEST ?AGENT (ON ?BOX ?TRUCK)>  
<LOAD ?BOX ?TRUCK>)
```

4.2 Learning coordinated procedures

This research ties learning from past success to the functioning of the memory of the individual agents. Learning is needed to overcome the initial lack of knowledge that agents have about the domain and the other agents they work with. Coordinated procedures enable the individual agents to improve their behavior by planning for new activities by borrowing from prior ones. For example, when an agent receives a familiar request, she can retrieve a plan which anticipates future requests rather than merely creating a plan to satisfy that single request. Also, remembering past successes leads to the agents independently adopting compatible viewpoints on the best order to achieve goals.

In this learning model, the coordinated procedures stored in procedural memory are culled primarily from execution traces. Others, such as Carbonell (1983) and Veloso and Carbonell (1993) and Laird *et al.* (1986), have argued that an agent should store planning histories in memory. However, reusing plan derivations will not, in general, produce a sequence of actions to better solve a similar problem in the future. On the other hand, execution traces encapsulate the history of both planned and

unplanned agent interactions with the domain. Consequently, coordinated procedures can be learned through the machinery of procedural memory that were not developed in a single (or multiple) planning histories.

Coordination devices, which represent past successful joint achievement of coordination points, form the skeleton of future plans. This skeleton is fleshed out by individual actions that support the achievement of coordination between agents. Coordinated plans can be effective since remembering points of coordination predisposes the individual agent to anticipate the actions and requests of other agents. Coordinated procedures will be effective if agents retrieve them in the right circumstances; in this dissertation, storing procedures indexed by the surface features of the environment propagates “compatible viewpoints” on how best to achieve a solution.

When different agents anticipate the same points of coordination, they can communicate more effectively for three reasons:

1. They will not waste time discussing alternatives that will prove to be unproductive.
2. They will not waste time negotiating over two workable alternatives.
3. In some cases, they can eliminate communication entirely.

Unfortunately, agents will sometimes assess the same situation in different manners. This reflects both differences in experience between agents and the open-endedness of interpretation in general. If this occurs, the agents can discover, and possibly redress, the situation during communication.

There are seven steps agents undertake in their post-activity analysis in order to add to their memories. There is no communication between the agents after the activity ends; the memories are created and maintained by each agent independently.

Cleaning the execution trace. Traces of execution-time activities are cleaned to remove unsuccessful behavior, which simplifies further analysis and prevents reifying past mistakes. Failed primitive actions, refused requests and idle time constitute the bulk of the trace entries removed during cleaning. This stage is a summarization process only in the broadest sense: it removes events that could not possibly have directly contributed to successfully achieving any goal, but does not distinguish between ones that might have.

Segmenting the execution trace. The cleaned trace is next reorganized into groups of actions that are related by the goals they achieve. There are two kinds of groupings are implemented: goal-groupings and time-groupings. For each top-level goal the agent achieved, the associated goal-grouping is identified by seeing if the goal literals intersect with the operator role fillers for each action in the cleaned trace. Time-groupings are found by looking at the actions in a time interval to see if their combined effect accomplishes a set of top-level goals; time-groupings allow the agents to learn to interleave top-level goals.

Removing inefficiencies. The agent's run-time behavior will doubtlessly contain mistakes of one sort or another. To prevent reifying sub-optimal behavior, it is desirable to identify and remove such inefficiencies. One could look at this step as either being a continuation of the cleaning process or as the commencement of the summarization process. The criteria for identifying inefficiencies are simple to state: an agent removes primitive actions whose effects are undone (usually by herself) and communicative acts associated with such primitive actions; also, all but one out of possibly several communications about a coordination point are discarded.

Optimizing coordinated procedures. As an optional optimization, a coordinated procedure may undergo two modifications related to coordinating past requests for service. SIGNs are changed into WAIT-IMPLICITs, reflecting an (optimistic) expectation that the request for service will be satisfied without a direct request and WAIT-FOR-REQUESTs are dropped, reflecting an (optimistic) expectation that the agent knows the right time to accomplish the request without being specifically asked.

Summarizing coordinated procedures. Actions which are planner-reconstructible are removed from the segment during this step. Two important consequences of removing reconstructible actions are to improve plan quality and reduce plan-merging effort at communication time.

Preparing summarized procedures. The low-level representation for the actions in the summarized trace segment is not suitable for re-use, so the procedure is prepared for future problem-solving episodes before storage. A straightforward change is to replace goal literals with variables. A more complicated preparation is to augment action descriptions with role-binding information that is lost when actions are removed during summarization or optimization.

Storing procedures in memory. Finally, the agent compares the prepared, summarized procedure to current case-base entries to determine if it should be added to the case-base or if a current entry should be generalized. Remembering is dependent on the agent's assessment of the situation at both storage and retrieval time, so agents can retrieve incompatible plans. When this occurs, communication provides an opportunity for the participants to get in synch.

Learning coordination knowledge in multi-agent systems has been studied under different settings. In a work with a common philosophical bias but a differing implementational outlook, Sugawara and Lesser (1998) present explanation-based learning techniques to learn coordination rules from failures in order to select and prioritize future diagnostic activities in a homogeneous multi-agent network monitoring system. Ho and Kamel (1998) use multi-agent probabilistic hill-climbing to learn reactive action selection to match environmentally provided feedback in communication-free settings.

Learning operator probabilities has similarities with reinforcement learning techniques (Sutton, 1988; Kaelbling *et al.*, 1996). Although reinforcement learning also allows an agent to act more capably by learning from the outcome of execution-time actions, there are major differences in representation and the type of action the learning supports. Reinforcement learning allows an agent to learn the best single next action to take given that she is in a particular (pre-defined) state based upon feedback provided by the environment. It is an unsolved research problem how an open world containing multiple agents can pre-define all of the possible states. It is also unclear how the learned policy is tied to the goal of the agent and therefore allow it to transfer its knowledge to different problems. Nonetheless, reinforcement learning has been successfully applied in communication-free settings (Mataric, 1992; Sen *et al.*, 1994; Sen and Sekeran, 1998) and has even been used to learn a very simple communication protocol (Yanco and Stein, 1993). A non-reinforcement learning approach to learning to control reactive behaviors can be found in Stone and Veloso (1998).

Case-based reasoning (Kolodner, 1993) has been used to change run-time behavior in other systems with multiple agents. In particular, Haynes and Sen (1998) keep a record of past execution-time conflicts in a communication-free domain to supplement control strategies in order to reduce conflicts in the future. Also, Ohko *et al.* (1996)

learn to better allocate tasks among a community communicating via a contract net by storing the outcome of past bid and task announcements. In general, global difficulties can arise when individuals use local criteria to determine the best case to retrieve, as discussed in NagendraPrasad *et al.* (1995). A critical difference between this work and prior multi-agent work is that agents are learning new pieces of procedural knowledge.

4.3 Example: learning to interleave goals

Using execution traces as a basis for future coordinated plans allows the agents to learn plans beyond the scope of the first-principles planner (or a traditional second-order planner based on it). Another example shows how lifter L2 learns to interleave two goals. In this example, HTO and L1 will act as in the first example, with the box in question now extra large box XLBOX1 rather than MBOX3. The following are L2's plans and actions, starting four steps before HTO makes her first request to L1.

1. L2 creates a typical plan to get XLBOX1 onto the truck: to lift, carry and load the box jointly with L1. L1 agrees to lift the box together and they do so. L1 then agrees to carry the box to the street. However, XLBOX1 is too large to carry, even jointly, and the action (and hence rest of the plan) fails.
2. L2 creates a plan to get SBOX4 onto the truck. The plan consists of putting down XLBOX1 with L1's help and then lifting, carrying and loading SBOX4 onto the truck by herself. L2 is delayed in asking for L1's assistance because HTO calls L1 first with a request to put XLBOX1 onto HANDTR2. L1 agrees to help HTO.

3. When L2 does ask L1 to help achieve HANDEEMPTY via putting XLBOX1 down together, L1 replies that she would rather load the box together onto the hand-truck. L2's planner adapts her current plan by replacing the PUT-DOWN-TOGETHER with the appropriate LOAD-TOGETHER. The agents then load XLBOX1 onto the hand-truck.

L2 continues on with her plan and loads SBOX4 onto the truck. Meanwhile, HTO has pushed the hand-truck to the street and L1 has agreed to get XLBOX1 onto the truck.

4. L2 constructs a plan to get large box LBOX2 onto the truck and moves back to ROOM1.

5. L2 is interrupted before attempting to lift LBOX2 by a request from L1 to help unload XLBOX1 from the hand-truck. L2 constructs the plan of moving to the street and then unloading XLBOX1. The agents do so.

6. L1 asks L2 to load XLBOX1 onto the truck and they do.

The sequence of actions L2 undertakes corresponds to six different calls to the planner. Nonetheless, L2 can extract a single coordinated procedure with only three actions in it (showing the original literals instead of new variables for clarity):

```
((LOAD-TOGETHER L1 XLBOX1 HANDTR2)
 (LOAD SBOX4 TRUCK1)
 (WAIT-FOR-REQUEST L1 (LOAD-TOGETHER L1 XLBOX1 TRUCK1)))
```

4.4 Coordinating from experience

There are two kinds of waits introduced by coordinated procedures: WAIT-FOR-REQUEST and WAIT-IMPLICIT. This section describes how they differ from the

first-principles WAIT, particularly in regard to how they influence communication and coordination. The responses that agents give during communication depend, in part, on how their current plan relates to the incoming request. Since the wait operators in coordinated procedures represent past and potential agreements (important relationships between plans and requests), responses are strongly influenced by their presence.

A WAIT-FOR-REQUEST is introduced whenever an agent agreed to a request during a previous activity. It acts as a place-holder to represent when the agent *expects a request* to be made. During communication, this must be treated differently than WAIT, which is only present when the agent has an explicit agreement.

```

Ticks 299 to 318: <STAND-HANDTR HANDTR3 PR36-STREET1> by HTO successful
Tick 317: L1 became frustrated since WAIT-FOR-REQUEST not satisfied
Ticks 319 to 375: HTO and L1 converse
    "HTO, I was expecting a call about (ON PR36-MBOX7 TRUCK3)."
```

```

    "L1, would you help me achieve (ON PR36-MBOX7 TRUCK3)?"
    "HTO, I've lost track of those items."
    "L1, here is that information."
    "HTO, I'll help, but you'll have to wait a bit."
Ticks 376 to 400: <MOVE PR36-STREET1> by L1 successful
Ticks 401 to 435: <UNLOAD PR36-MBOX7 HANDTR3> by L1 successful
Ticks 436 to 470: <LOAD PR36-MBOX7 TRUCK3> by L1 successful
```

Figure 4.2: Coordination based on explicit expectations: same goals.

In Figure 4.2, the agents are working from plans which are derived from past joint activity and the lifter's plan contains a WAIT-FOR-REQUEST operator. Although the two agents are working from the same past interactions, L1 becomes frustrated waiting for HTO to make her expected explicit request. (Chapter 6 describes frustration and communication in detail.) As it turns out, L1's inquiry came in the same tick that HTO would have made the request, so the conversation proceeds smoothly.

Note that, from an efficiency stand-point, this is no better than a first-principles solution. However, in this case, the outcome is more robust: it does not depend on serendipitous timing or agreeable agents.

```

Ticks 77 to 111: <LOAD-TOGETHER PR34-XLBOX1 HANDTR3> by L1 and L2 successful
Ticks 162 to 181: <STAND-HANDTR HANDTR3 PR34-STREET1> by HTO successful
Ticks 217 to 237: L2 and L1 converse
    "L1, would you help me achieve (HOLDING-TOGETHER PR34-XLBOX1)
    via (UNLOAD-TOGETHER PR34-XLBOX1 HANDTR3)?
    This is part of a plan involving you to achieve (ON PR34-XLBOX1 TRUCK3)."
```

"L2, sure! Lets get to it."

```

Ticks 238 to 272: <UNLOAD-TOGETHER PR34-XLBOX1 HANDTR3> by L1 and L2 successful
Ticks 273 to 290: L1 and L2 converse
    "L2, would you help me achieve (ON PR34-XLBOX1 TRUCK3)
    via (LOAD-TOGETHER PR34-XLBOX1 TRUCK3)?"
```

"L1, sure! Lets get to it."

```

Ticks 291 to 325: <LOAD-TOGETHER PR34-XLBOX1 TRUCK3> by L1 and L2 successful
```

Figure 4.3: Coordination based on implicit expectations: same goals.

As an optimization, a WAIT-IMPLICIT replaces a SIGN operator which had requested a service to be performed. A WAIT-IMPLICIT is an optimistic plan modification: the agent *expects a service* by another agent without having to ask for it. Figures 4.3 and 4.4 show scenarios when agents are acting from optimized coordinated procedures. In both of them, the box requires two lifters to handle. As the dialogues starting in ticks 217 and 273 of Figure 4.3 show, not all communication has been removed; namely, communication about joint actions (which require more fine-grained coordination) has been kept. Despite this, the optimizations have led to an efficient solution by the community because the implicit expectations prevented some superfluous dialogs. In Figure 4.4, the risk of the optimizations is revealed. In this case, 160 ticks went by after tick 632 without any progress being made because the lifters did not see the situation in the same way that the hand-truck operator did. Except for the delay, everything proceeds as it would have by first principles

(the joint actions by L1 and L2 starting in ticks 879, 1073, and 1126 are preceded by conversations that are not shown).

```

Ticks 507 to 541: <LOAD-TOGETHER PR41-LBOX3 HANDTR3> by L1 and L2 successful
Ticks 613 to 632: <STAND-HANDTR HANDTR3 PR41-STREET1> by HTO successful
Tick 793: HTO became frustrated since WAIT-IMPLICIT not satisfied
Ticks 794 to 859: HTO and L2 converse
    "L2, would you help me achieve (ON PR41-LBOX3 TRUCK3)?"
    "HTO, I'll help, but you'll have to wait a bit."
Ticks 879 to 908: <PUT-DOWN-TOGETHER PR41-XLBOX2 PR41-ROOM1> <snip> successful
Ticks 909 to 933: <MOVE PR41-STREET1> by L2 successful
Ticks 948 to 1026: L2 and L1 converse
    "L1, would you help me achieve (HOLDING-TOGETHER PR41-LBOX3)
    via (UNLOAD-TOGETHER PR41-LBOX3 HANDTR3)?
    This is part of a plan involving you to achieve (ON PR41-LBOX3 TRUCK3)."

```

Figure 4.4: Coordination based on implicit expectations: different goals.

4.5 Learning operator probabilities

Probabilities influence many decisions in *MOVERS-WORLD*. Agents use them when planning from scratch, when deciding whether to cooperate, and when adapting coordinated procedures to match the current problem setting. Having accurate probability estimates, therefore, is essential for agents to act efficiently. Operator probability trees are the generalization mechanism that agents use to make accurate probability estimates based upon past experience interacting with the domain and other agents.

To get a feel for how estimating probabilities can help an agent behave efficiently, consider a common decision a lifter must make. A lifter can lift some boxes alone, but

not all of them. If a lifter can lift the box alone, it is more sensible to do so because they do not have to spend time asking for assistance (and there is no guarantee the assistance will be given). On the other hand, if there is little chance that the lifter can handle the box on her own, it is a waste of time (and energy) to make the attempt. So, individuals should be able to recognize which of these two possibilities is more likely and act accordingly.

In MOVERS-WORLD, successfully executing an action means that after the attempt has been completed, all of the expected effects have occurred. The system determines whether actions succeed or not based upon a set of execution-time conditions unknown to the agent. The domain features that determine whether actions are possible are object size, object weight, agent strength, and item capacities. None of these are observable features of the environment. Item capacities specify how many other items and how much total weight the item can hold. The observable box features are height, width, depth and material - each measured in integers between 0 and 3. For boxes, the size is derived from the volume of the box and the weight is a linear function of the size and material. The label associated with a box includes a S, M, L or XL solely in order to make it easier for the human reader to parse the system output.

Initially, probabilities are set to 50%. This steers an inexperienced lifter to try to lift boxes alone because a plan to do so will be shorter (because there is no communication needed) and thus more likely to succeed. So, at the beginning of a problem, lifter L1 would try to pick up a large box LBOX1 by herself. L1 would fail; the next time she wants to lift LBOX1, L1 will decide to ask for the help of another agent. This decision is based upon L1's experience interacting with that particular box.

L1's experience interacting with a particular box will not prevent her from attempting (and failing) to lift other large boxes alone. Tree structures provide a mech-

anism by which agents generalize past run-time interaction experiences, saving time and effort. The successes and failures of attempted actions are stored in a COBWEB (Fisher, 1987) tree associated with all of the observable features of the various role fillers for the action. Actions might fail for reasons not explicitly considered (q.v. the *qualification problem*: McCarthy, 1977) and COBWEB can handle this noisy data. Also, COBWEB trees can be updated incrementally, which allows the agents to learn during the course of their activity. If the observable characteristics (e.g., height, width, depth, texture) of another box LBOX2 exactly match LBOX1, L1 will not attempt to lift LBOX2 alone. If the features do not match exactly and there are other experiences stored in the tree, L1's behavior depends on which experience the COBWEB classification algorithm considers the best match with the current action.

4.6 Discussion and related research

The learning techniques in this chapter are more than just a vital component of a multi-agent framework. They are also examples of techniques that apply to many other complex domains, whether they contain many actors or one. These techniques fit into a framework for learning that emphasizes learning as a means of acting more ably in her environment.

In order to act more ably, an agent must be able to revise her model of the world in order to reflect the results of past actions. For example, on the basis of prior experience, the agent can revise previous estimates modeling the domain, such as how likely an action is to succeed or how likely another agent is to agree to help. Or, the agent may add to the plans she knows if the activity led to a solution in a way that was previously unknown. Or, based on past execution problems, the agent can learn when not to use a known plan.

This emphasis on learning in support of acting more effectively is orthogonal to the traditional Artificial Intelligence view of learning as a means to plan more quickly. In other words, this work is interested in agents learning to make better decisions, as opposed to making those decisions more quickly. Such speed-up techniques, e.g., learning search control knowledge (Minton, 1988) or remembering how previously-generated plans were constructed (Kambhampati and Hendler, 1992; Veloso and Carbonell, 1993), would make worthwhile additions to the architecture presented herein. They would increase the computational power of an implementation of these ideas, but they would not change the range of execution-time behaviors demonstrated by the agents. For this reason, speed-up is not a focus of this research even though its importance cannot be ignored.

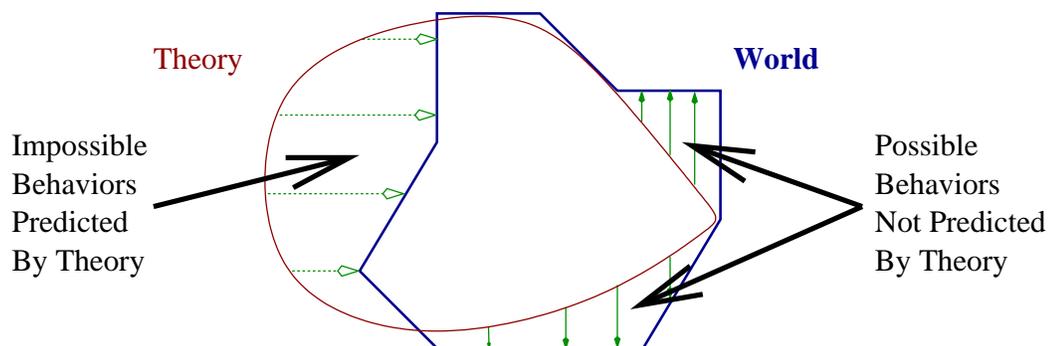


Figure 4.5: Differences between theory and practice.

Figure 4.5 abstractly depicts how an agent's domain model should be changed by comparing the scope of the model, defined to be the range of possible behaviors generated by the model, to what can actually be done. The rounded region depicts the scope of the model; it is rounded to emphasize the fact that most theoretical models predict easily-defined sets of behaviors. The polygon represents the less structured dynamics of the world itself. The horizontal and vertical arrows lie in the gaps between

the figures and the direction of the arrows indicate whether the scope of the model needs to be narrowed (horizontal arrows) or widened (vertical arrows).

Figure 4.5 is not meant to imply that it will be possible to shrink and expand the model into a completely accurate model of the world. For one thing, the world itself may be changing, so an agent can never be completely confident that what worked in the past will work in the future. Also, it is impossible to represent the infinite amount of information about the world that could be relevant. In other words, there is no guarantee that an accurate model of the world falls within the finite scope associated with an implemented agent architecture, regardless of how large that scope is. Furthermore, it may not be practical to build an agent that can perceive and/or process the amount of information proscribed by the theory.

In this work, learning operator probabilities are an example of how an agent can learn from past experience without changing the scope of her model of the world. Revising probabilities revises the likelihood of returning this or that plan, but does not rule out old plans or add new ones. In fact, most learning techniques fall into this category; the presumption is that the model is sufficiently general so that the convergence of the pre-defined parameters insures the agent will behave efficiently. In reinforcement learning (Kaelbling *et al.*, 1996), agents are simultaneously learning the environmental rewards and transition functions that control the agent's reactive behavior. In neural networks, connection weights are revised based upon the feedback associated with a set of inputs. As an example of both at once, Tesauro (1992) shows how temporal difference learning, a particular kind of reinforcement learning, can train a neural net backgammon program to an expert level through self-play. Adjusting domain estimates is not the only way to change behavior based upon experience. An alternative way to change behavior without changing the scope of the model is the operationalizing of generalized knowledge done by explanation-based

learning systems (Mitchell *et al.*, 1986; DeJong and Mooney, 1986).

In general, though, a complete learning framework should be able to make more substantial changes to the model of the world than the above techniques. Another way to state the issue at hand is put forth in Hammond (1990, page 386):

Any problem that arises out of a disparity between what the planner knows and how the world is actually structured indicates that the model of the world ... should be changed.

If the model of the world needs to be changed, the question is how. One possibility is for an agent to be guided by a “meta-theory” that is able to somehow generalize the old theory to fit new facts. However, a convincing argument against this is given by Goethe (1988, p. 307):

The ultimate goal would be to grasp that everything in the realm of fact is already theory ... Let us not seek for something beyond the phenomena — they themselves are the theory.

The AI translation of this statement by Goethe is that case-based reasoning (Kolodner, 1993) provides a more promising approach than a meta-theory. In CBR, isolated instances of success or failure can stand on their own, without a supporting theory, and provide the agent a means to change the scope of her original model.

Previous approaches have emphasized the use of a case-base when learning from failures. Hammond (1990) focused on learning to anticipate and avoid problems through the explanation of past execution failures in a framework that is exclusively case-based. In this model, run-time failures are not repaired at the time; instead, the knowledge is stored so that future plans will execute without failure. Planning failures can also be considered *expectation failures* (Schank, 1982), which Schank

argues is what drives an understander to learn. Another memory-based technique to anticipate failures is *projective visualization* (Goodman, 1993). An agent using projective visualization evaluates a course of action by running the situation forward into the future, retrieving past cases to estimate the likely outcome. When execution-time failure occurs, the agent induces the set of observable features that caused the failure and uses them to index the episode. In this way, experience at the task prevents the agent from repeating past mistakes. Using a case-base to block an agent from repeating undesirable past behaviors is an instance of narrowing the scope of an agent's model.

The emphasis of this work is on learning from past successes. No domain theory will predict all possible behaviors in complex domains, so a learning framework should add overlooked behaviors to the theory when possible. If an agent's activity follows an unplanned but successful path, the agent should remember and re-use that experience. This will widen the scope of the model.

Figure 4.5 implies that learning from failures can only restrict the initial domain model, but this is not always the case. A single-agent system that incrementally builds a bigger model through execution breakdowns is FLOABN (Alterman *et al.*, 1998b). The activity is modeled by a representation called a multi-case (Zito-Wolf, 1993) that organizes episodic memory so that it can be used to guide future behavior. The adaptive planning agent begins activity with a simple starting representation and a set of execution-time adaptation strategies. In some situations, the only way for FLOABN to solve a problem is to acquire new information by seeking out and reading instructions (Carpenter, 1998).

Adding new behaviors requires that knowledge is available in the environment to be acquired by the agent. For many Artificial Intelligence domains, such knowledge is not a “natural” part of the task. New knowledge can be said to be naturally arising if it is acquired through agent behavior whose sole motivation is to solve the current problem. For example, FLOABN acquires information naturally through reading and MOVERS-WORLD agents acquire knowledge naturally through conversation. In such situations, learning procedures that will be useful in future problems is secondary — a beneficial byproduct of actions the agent would make anyway.

In summary, this thesis advocates that a complete learning framework should include a means of narrowing and widening the scope of an agent’s model of the world, in addition to standard methods that change the contents of a model without changing its scope. This thesis presents techniques to widen the scope of an agent’s model by learning procedures that includes information about other agents that was acquired naturally in the course of the activity. This learning framework is general enough to apply to single- or multi-agent systems and should be considered a core component of a model to support activity among heterogeneous, independent agents.

4.6.1 PRODIGY

There are implementational similarities between this work and Prodigy/Analogy (Veloso, 1994). However, this thesis is motivated by quite different research goals than those underlying Prodigy/Analogy, leading to substantial differences too. Prodigy/Analogy is not an integrated planning and acting system, it is a single agent system, and is primarily focused on speed-up. In sum, the differences are far more substantive and foundational than the similarities.

Prodigy/Analogy is a single agent planning system that improves its performance by learning cases, called *derivational histories*, directly from the output of the first-order planner. Prodigy/Analogy, as with most planning systems, considers a constructed plan a solution. If prompted, it can produce many solutions, but it does not provide a mechanism for comparing their quality other than plan length. Furthermore, since there is no evident way for Prodigy/Analogy to generalize execution-time failures that do not come with predicate-based explanations, it is not clear how to extend Prodigy/Analogy to an integrated planning and acting architecture.

Comparing Prodigy/Analogy to MOVERS-WORLD requires a reductionist view; examining how each agent plans and learns in isolation. However, planning, acting and learning in a multi-agent environment is more complicated than merely giving each agent a good single-agent planner. The central multi-agent issues of cooperation, coordination, and communication are left unaddressed by a reductionist approach.

Prodigy/Analogy and MOVERS-WORLD must address a wide range of common issues relating to integrating a case-based reasoning component on top of a baseline planning system. Differing research motivations directly influenced how the two systems addressed the following questions:

1. Exactly what past experience should be remembered?
2. How should past experience be represented so that the planner can use it in the future?
3. How should the case be indexed into memory?
4. When and how should the case library be searched for similar past experiences?
5. How should past experience be adapted when the current circumstances do not allow for straightforward retrieval?

In Prodigy/Analogy, complete planning histories are stored in justification structures that provide the underlying planner with all information needed to replay the previous solution. The storage and retrieval processes are designed to incorporate the derivational replay mechanism within the baseline planner with minimal additional CPU overhead. Multiple past experiences are merged via *ad hoc* meta-planner strategies, and adaptations are handled through simple insertion/deletion/substitution transformations on the derivational histories. In summary, Prodigy/Analogy uses past experiences to learn to produce a solution more quickly.

In MOVERS-WORLD, the portions of past activity that are remembered are those actions that were relevant to establishing cooperation or maintaining coordination between agents during successful problem-solving. Partial execution histories are converted into sequences of actions that have the same representation as first-principles actions. The storage and retrieval processes are designed to allow the use of common past experience in lieu of first-principles planning. The stored plan constitutes a starting point that the agent is constantly revising during the course of activity as events unfold. In this regard, a stored plan is treated identically to a first-principles plan. In summary, MOVERS-WORLD agents use past experiences to learn to produce a plan that is more likely to succeed.

Among Prodigy/Analogy's research contributions, a primary one was to extend the power of time-bounded problem solving. In other words, using learning to overcome a limited amount of planning time. On the other hand, this dissertation is primarily concerned with using learning to overcome a limited amount of information about, and control over, other agents and the world.

Chapter 5

Empirical results and analysis

This dissertation asserts that learning techniques allow a community of agents to overcome a dearth of built-in common knowledge. This chapter supports that assertion by presenting rigorous empirical studies for an implemented test-bed. Careful experimental methodology accounts for the influence of initial random seeds, sampling bias and ordering effects.

After delineating the experimental methodology in the rest of this introduction, the first section presents the central results showing the significant benefits of learning. The second section contains learning curves for several key statistics to show how learning is improving behavior during the performance of the activity itself. The next section digs slightly deeper to reveal qualitatively why the behavior is improving. Finally, results are included whereby the functionality of the system is incrementally changed in order to quantify the contributions of four major learning components.

The test-bed system is written in over 30,000 lines of object-oriented Common Lisp that produces 3MB of byte-compiled code. The system solves both individual MOVERS-WORLD problems and sequences of them. Individual problems are constructed by randomly selecting subsets from the pool of permanent MOVERS-

WORLD objects (agents, hand-trucks, and trucks) that will be active for that problem. Then a random group of boxes and locations is constructed and a list of goals involving them is generated. For these experiments, a database was created of 60 problems; the goal of each problem was to move all boxes to the truck. The number of boxes was uniformly distributed between 3 and 5.

The experiments are designed to control as many sources of randomness as possible in order to make comparisons meaningful. In particular, community performance can be strongly effected by the outcome of random decisions made by either the agent (e.g., selecting unmet goals to work on) or the system (e.g., when determining the outcome of resource conflicts). The starting random seed value for each problem can be stored and reused, but this is not sufficient. Since the order in which decisions are made varies, each randomly-decided decision point is stored together with a sequence of outcomes. The first time the decision needs to be made, the first item in the list gives the outcome. The second time the decision needs to be made (during the course of the same activity), the second item gives the outcome, *et cetera*.

In addition to concerns about the influence of random decisions on the results of individual problems, there are concerns about the the influence of problem ordering on the shapes of learning curves. It is not feasible to determine learning curves by running the system on all possible permutations of the database problems, so the system is run on ten predetermined groups of sequences. Each group of sequences is balanced in the following way: each of the database problems occurs once as the first problem-solving episode of some sequence in the group, once as the second of a different sequence in the group, *et cetera*. So, each point on the graph is the result of solving each of the 60 database problems ten times, using different sets of seeds and decisions each time; these 600 runs were repeated for both the baseline system and when the agents were learning.

As specified in Figure 3.2, the termination condition for the group of MOVERS-WORLD agents is that all agents believe the cooperative goals have been achieved. However, this adds a small but uninteresting component to the results, corresponding to the time needed for the uninformed to realize that the task is over. (Agents become informed by wandering to the street to get the correct state of world assessment and/or through conversation with an agent with the correct state of world.) For this reason, the statistics presented are as of the time when the cooperative goals have been achieved regardless of whether all agents are aware of it yet.

To get a feel for the impact, the following chart gives a preview of some statistics about the amount of time needed, measured in clock ticks, by the community to solve MOVERS-WORLD problems. The standard error is computed by $SE(X_1 - X_2) = \sqrt{(S_1^2 + S_2^2)/n}$, where n is the number of data points (600 in this case).

	Baseline System		Learned System		$X_1 - X_2$ Difference	$SE(X_1 - X_2)$ Std. Error
	X_1 Average	S_1 Deviation	X_2 Average	S_2 Deviation		
Uninformed	1130.033	493.234	644.692	215.985	485.341	21.982
Informed	1128.798	493.294	623.513	213.776	505.285	21.948

5.1 Cumulative impact of learning

There are many ways to measure the performance of the agent community, such as the number of primitive actions attempted and the number conversations that occur. The best overall measure of community effort, however, is the number of ticks of activity that transpire during the course of the community solving the problem. This measure of simulated time includes both action and communication effort, in addition to time when the agents are idle for one reason or another.

	Statistic	Baseline Behavior		Learned Behavior	
Events	Time Steps	1128.8 ticks		623.5 ticks	
	Total actions duration	1485.9 ticks	43.9%	940.0 ticks	50.3%
	Total conversations duration	965.0 ticks	28.5%	405.4 ticks	21.7%
	Waiting for a request	429.1 ticks	12.7%	370.3 ticks	19.8%
	Idle	506.3 ticks	15.0%	151.7 ticks	8.1%
Acting	Attempted actions	46.4		24.2	
	Successful actions	31.2		23.6	
	Wanderings	3.9		1.2	
	Attempted joint actions	10.7	23.1%	6.4	26.4%
	Successful joint actions	7.3		6.3	
Talking	Conversations	24.6		10.0	
	Agreements	16.3	66.3%	7.7	77.0%
	Agreements to change goals	6.6		0.7	
	Refusals	9.8		2.9	
Planning	Calls to the planner	73.8		34.9	
	Planning nodes expanded	2865	38.8	1060	30.4
	Role-binding nodes expanded	3406		1858	
	No plan generated	4.2		1.9	
	Current plan adapted	5.3		7.1	
	CPU time	6.96 seconds	162.2	5.13 seconds	121.5

Table 5.1: Community statistics: baseline versus learned behavior

Table 5.1 shows a battery of statistics characterizing the community activity, both for the baseline system and after agents have learned better models of their domain through practice. Some rows include both a first-order (i.e., directly measured) statistic and a second-order statistic derived from it. In those cases, the text will describe how the second-order statistic is calculated. To be precise, “Baseline Behavior” includes learning specific interaction knowledge (as described in Section 6.1); the column headed “Learned Behavior” additionally includes learning operator probabilities and coordinated procedures. The learned behavior measures the community performance for the tenth (out of ten) problem-solving episode that the community faced so the statistics in the chart show the cumulative effect of learning. This is not meant to imply that the agents should be “trained” before measuring the impact of learning;

in the next section, learning curves will fill in the gap between the baseline and the cumulative learned behavior.

The statistics in Table 5.1 are separated into four groups. The first group, labelled Events, shows that agents solve the problems in 623.5 ticks when learning as opposed to 1128.8 ticks when they are not, a reduction of 44.8%. The standard deviations for these data are 213.8 and 493.3 and there are 600 data points, so the $(100 - \alpha)\%$ confidence interval for the difference between these means is $505.3 \pm 21.9Z_{\alpha/2}$. This allows us to reject the null hypothesis that the two population means are equal with over 99% confidence.

The last four statistics in the first group of statistics decompose the activity of the community into disjoint event classes. The total actions duration includes the total time spent by all agents on all actions, whether the actions were successful or not and whether the actions are individual or joint. Likewise, the total conversations duration double counts the length of each dialog, since two agents are spending their time talking. The time spent waiting for a request includes all ticks that any agent executed a wait operator, be it a WAIT, a WAIT-IMPLICIT, or a WAIT-FOR-REQUEST. In other words, the agent may be waiting for a joint action to commence or a service to be performed or an expected request to be made. Section 6.3 outlines the circumstances in which an agent is idle; two common examples are when her action attempt is delayed by a resource conflict or she is waiting for a communication connection to be established. The second-order statistics are derived by dividing by $N * Ticks$, where N is the number of active agents (3 for these experiments), which is the total number of agent-ticks expended by the community. These statistics show that, when learning, the agents spend a larger percentage of their time acting and waiting for requests to be satisfied and a smaller percentage of their time conversing and idle.

The second group of statistics, labelled Acting, show that agents are solving problems with less action effort. Learned agents attempt 21.8 fewer actions than their baseline counterparts. Also, the problems are solved using 23.6 successful actions, compared to 31.2 in the baseline system, although around a third of this improvement is attributable to less wanderings. The number of successful joint actions drops from 7.3 to 6.3 and the number of attempts from from 10.7 to 6.4. However, the number of attempted joint actions as a percentages of total attempted actions rose from 23.1% to 26.4%. So joint actions account for a larger portion of the community's action effort when learning.

The focus of the third group of statistics is to track the efforts of the agents when they are conversing. All first-order statistics show a dramatic drop. The number of dialogs plummets by 59.3%, going from 24.6 in the baseline system to 10.0 when the agents are learning. Similarly extreme reductions are shown in the number of agreements, both overall and when it involves changing goals, and in the number of refusals. It should be pointed out that agents all share the same joint goals in these experiments, so changing goals in this context means switching the order in which the goals are worked on. The alert reader will notice that the sum of agreements and refusals exceeds the number of conversations. This is due to alternative request responses, which were portrayed in Section 6.4.3. Finally, a second-order statistic, the percentage of conversations that ended in agreement, climbed from 66.3% to 77.0% when the community is learning.

Planning statistics give an idea of the computational efficacy of these techniques. The focus of this research is not primarily upon computational efficiency, but the statistics are included for completeness. Learning significantly reduces the number of planning calls and the number of nodes expanded during search, for both planning and role-binding nodes. Furthermore, the average number of planning nodes expanded

per planner call dropped from 38.8 to 30.4 and the number of planning calls for which no plan could be generated went from 4.2 to 1.9.

The number of times an agent's current plan had to be adapted increased due to learning, climbing from 5.3 to 7.1 times per problem-solving episode. However, this should not come as a surprise. When agents are learning coordinated procedures, those procedures do not include all of the actions that a first-principles plan does. So, even if activity is proceeding as expected, there will be occasions when the plan must be adapted. For example, a common start to a hand-truck operator's coordinated procedure is the actions:

```
(WAIT-IMPLICIT L1 (ON BOX7 HANDTR2))  
(STAND-HANDTR HANDTR2 STREET)
```

In this case, the hand-truck operator does not fill in the missing TILT-HANDTR and PUSH-HANDTR actions until execution-time, after (ON BOX7 HANDTR2) becomes true.

The last statistic shown is CPU time. It is an inadequate measure of system performance since the implementation is a simulation of an activity, but CPU time is traditionally included in empirical AI results. The CPU time to simulate an agent attempting an action or exchanging information is dwarfed by the amount of CPU time required to plan. So, the significant drop in CPU time is of little note. Slightly more interesting is that the number of community ticks processed per CPU second of the system, is less when agents are learning. Instead of being able to process 162.2 ticks per second as it does when the agents are not learning, the test-bed processes 121.5 ticks per second. This gives a rough estimate of the overhead of the learning techniques, namely the time spent building and searching COBWEB trees and the time spent storing plans to, and retrieving plans from, memory.

5.2 Learning curves

The previous section was an exhaustive examination of the impact of learning better domain models after the agents have solved nine prior problems. The techniques are not meant to be used as a training tool, though, so results will now be presented showing how the learning incrementally improves performance while the community is immersed in the activity. The presentation of the framework in Chapter 2 emphasized the desire for a system to act at or near optimality. Figure 5.1 shows the percentage of optimal performance of the community, calculated in three different ways. The most natural way of computing optimality is to divide the theoretically best performance (in ticks) of the system by the measured performance of the system. The curve labelled 'Impossible' tracks the community performance measured in this way. To be specific, for these MOVERS-WORLD experiments, the average optimal number of ticks required to move all boxes onto the truck was 390.05 ticks. So, if a data point on Figure 5.1 reads 50%, it means the measured performance was roughly 780 ticks.

The optimal performance for these problems was determined assuming that the problems could be achieved using no communication whatsoever. However, as pointed out in the example of Section 3.6, the distribution of planning knowledge in MOVERS-WORLD makes optimal performance on any problem impossible since communication is always required. So achieving optimality under this measure is impossible.

The curves labelled '1 Tick dialogs' and 'Best Observed' are alternate measures of optimality that are designed to better isolate how efficiently the MOVERS-WORLD actors are coordinating their actions at run-time. The curve labelled '1 Tick dialogs' are the result of re-running the simulation with *CommOverheadRatio* and *CommRatio* set to 0, so that the length of all conversations is set to a single tick.¹

¹See section 6.4 for details of these parameters.

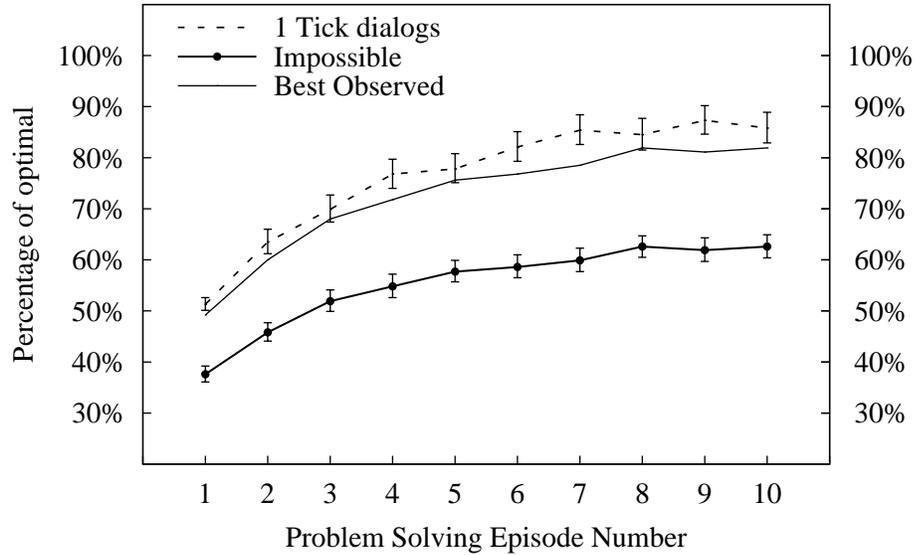


Figure 5.1: The community approaches optimal overall performance.

This curve exceeds 84% after the sixth problem-solving episode, peaking at 87.3% for the ninth problem-solving episode. This demonstrates that the learning techniques of this thesis allow the community of agents to start with very limited common knowledge and reach highly efficient solutions.

The curve labelled 'Best Observed' compares the standard (i.e., when the length of dialogs vary) results to an alternate measure of optimality. This measure is the average best *observed* performance by the community out of the 600 runs. For example, problem number fifty-one in the database can theoretically be solved in 506 ticks. However, the best measured performance was 667 ticks (the average was 930.3 ticks with $s = 376$). For these ten balanced runs, the average best observed was 510.7, so this curve parallels the 'Impossible' curve but is roughly 31% higher. Error bars are not included for this curve because of the overlap with the '1 Tick dialog' curves.

For the rest of this chapter, interest will be shifted away from optimality. Instead, the focus will be on the impact the two learning structures have on community performance. So the figures will presented the statistics directly, rather than as a percentage of optimal. Learning curves will be labelled “CP+OP”, “CP”, or “OP”. CP is shorthand denoting the fact that agents are learning coordinated procedures and OP indicates that agents are using operator probability trees. The baseline (no learning) behavior will be plotted in the same style as on this graph on most figures but is usually unlabelled to keep figures from being overly cluttered.

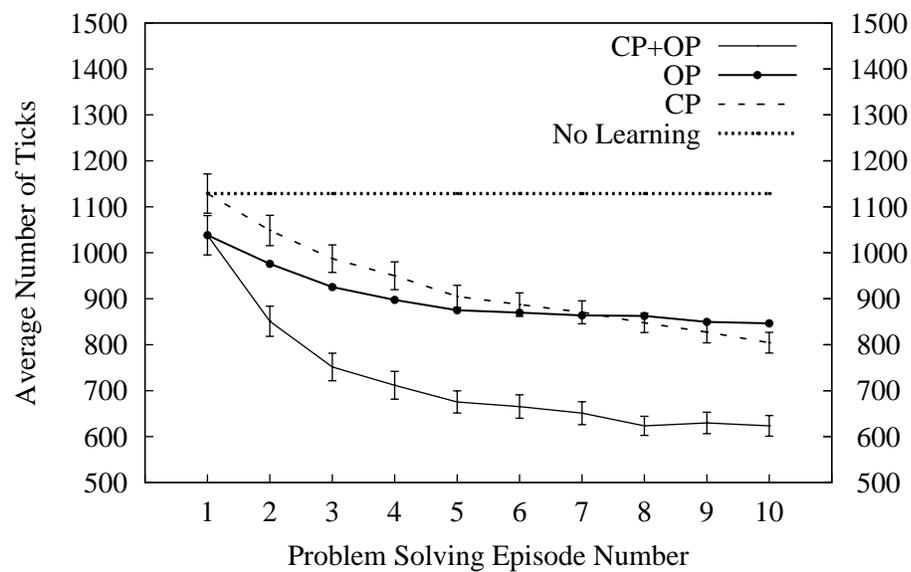


Figure 5.2: Overall community performance improves.

Figure 5.2 shows how the two learning techniques reduce the number of ticks of activity the community takes to solve problems. The data for runs where the agents learned coordinated procedures are graphed along with their 99% confidence intervals. Both OP and CP lead to statistically significant improvements on their own. In addition, their learning curves, including error bars, are so similar that the

confidence intervals for OP could not be included without obfuscating the figure. OP shows significant intra-problem learning, reducing the average number of ticks for solving the first problem of a sequence from 1128.8 to 1038.1. This number continues to go down to until it levels off at 846.3, but the learning after the fifth problem is not significant with 99% confidence. The CP curve falls more slowly, but eventually undercuts the OP curve beginning with the eighth episode. Although unshown, the OP confidence intervals are almost the same width as the CP intervals.

The combined effect of learning coordinated procedures and operator probabilities has substantially more impact than either alone. After two problem problem-solving episodes, the number of ticks expended by the community is 751.6, which is less than the best performance at any point in the sequence for just CP or OP. Improvement is significant until the seventh episode.

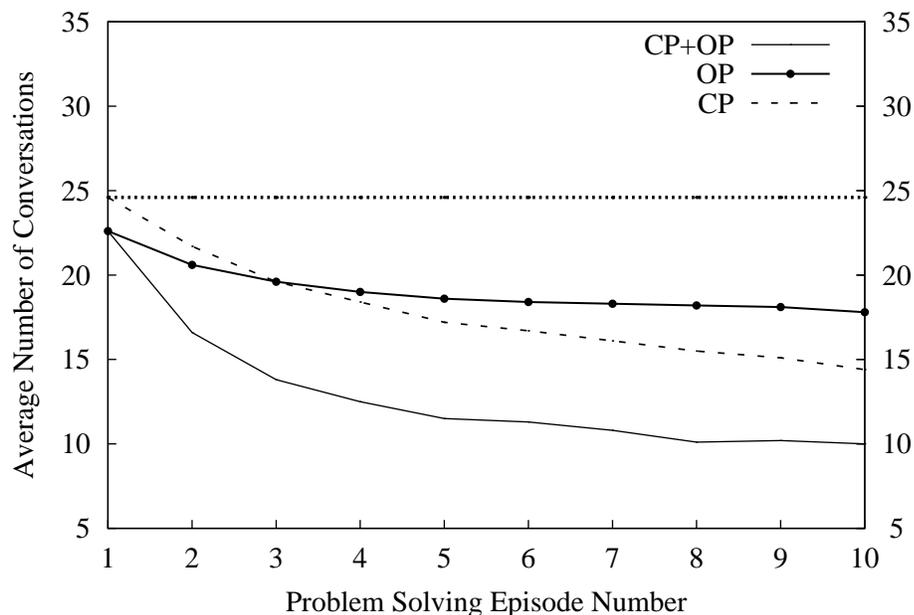


Figure 5.3: Community communication decreases.

Figure 5.3 shows that acting from past experience in MOVERS-WORLD reduces communication during activity. The average number of conversations required to solve these problems without learning was 24.6. Learning coordinated procedures leads to statistically significant improvement; the average number of conversations drops fairly steadily, ending at 14.4, a 41.5% improvement over the baseline system. OP is not as effective in reducing communication, only dipping below 18 talks per problem for the tenth problem. As with the number of ticks, the number of dialogs is much lower when the agents are learning both new behaviors to use and old behaviors to avoid. By the third problem, the number of conversations (13.8) is less than at any time in the sequence for OP or CP. The final value of 10.0 is a reduction of 14.6 conversations, slightly less than the combined reductions of OP (6.8) and CP (10.2).

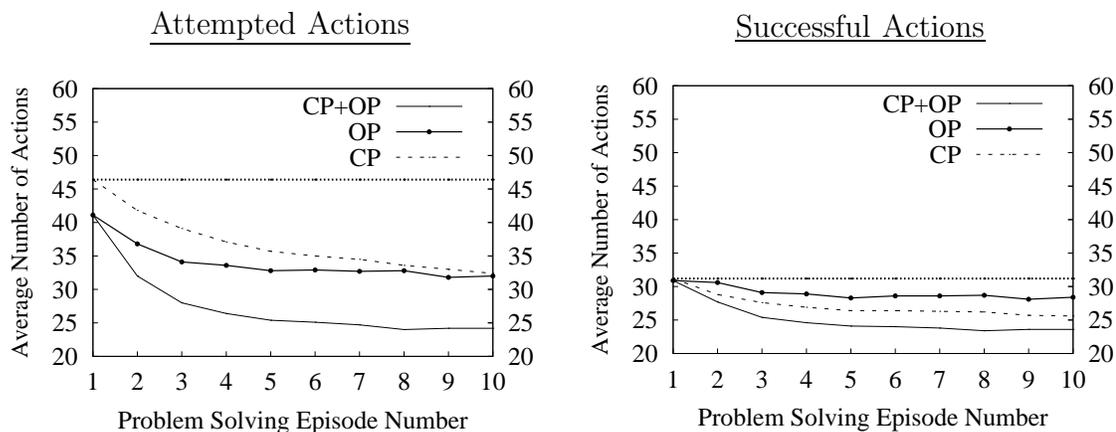


Figure 5.4: Community action effort decreases.

Figure 5.4 shows that the amount of effort agents expend attempting and successfully executing primitive actions improves despite the decrease in communication. On their own, OP and CP lower the number of attempted actions from 46.4 to around 32. Combined, the number is lower still, down to 24.2 attempts per problem. The

reduction in successful actions is not as dramatic. OP manages to reduce the average from 31.2 to 28.4, whereas CP leads to a level of 25.6 for the tenth problem-solving episode. Combined learning improves this a small amount — the community executes only 23.6 successful actions. As with the number of conversations, the sum of the reductions of OP (2.8) and CP (5.6) barely exceeds the reduction when both techniques are used (7.6).

The improved runtime performance of the community is a direct result of the fact that the planner produces plans that are either more efficient (e.g., interleaves goals) or more likely to be successful or both. The question must be asked whether this improvement in plan quality comes at a high price in increased planner effort due to increased match costs and/or increased branching factors. Using a memory of coordinated procedures in MOVERS-WORLD, there is no high price to pay. See Figure 5.5. Learning operator probabilities leads to more search in order to instantiate local role-binding variables (the left-hand plot of the figure) and little improvement in the number of planning search nodes expanded during calls to the planner. Learning coordinated procedures offsets this, so that the cumulative effect is to reduce planning time search effort.

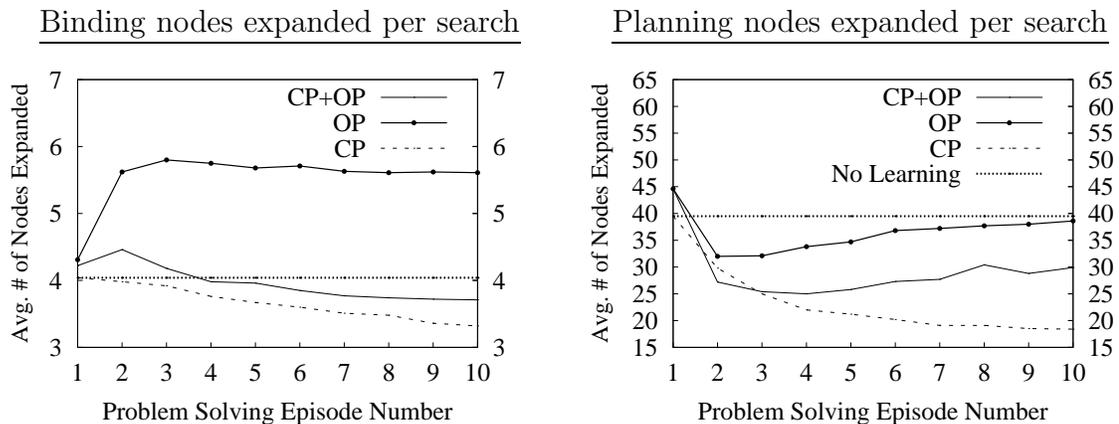


Figure 5.5: Planning effort decreases when agents learn coordinated procedures.

5.3 Why is learning improving performance?

This section will identify why learning a better model of the domain is dramatically improving performance. It will focus on exactly how the agents' improved domain models are improving the quality of their execution-time reasonings that cumulatively determine community performance for the entire problem. It will show that the primary causes for improvement are the development of compatible viewpoints, a reduction in misguided or unnecessary communication, learning procedures that efficiently interleave subgoals, and more frequently working on individual subgoals in the most efficient manner. The statistics presented in this section will be suggestive, not conclusive. The next section will more directly attribute what contributes to the improvement of the community by systematically altering and re-running the system.

There are three kinds of execution-time reasoning decisions that agents make:

Goal selection. Goal selection in the architecture presented in this work means picking a top-level goal to be actively pursued. In general, it can be considered a selection for goal ordering. The quality of an agent's goal selection is judged by the execution-time impact it has, so it can only be evaluated in the context of other agents' goal selections. In other words, each agent could select an ordering which is locally optimal but the net effect is that too many agents are working on some goals and too few are working on others.

Plan Determination. A plan can be determined by retrieving one from memory or generating one from first-principles. The quality of plans that an agent determines can improve over time for one or more of the following reasons:

- The plans are more likely to succeed.
- The plans represent new ways to solve problems.

- The plans are more efficient, in a domain-appropriate measure. Usually, this would be related to the length of the plan, but in this model it is the cumulative duration of the plan. It is also related to the number of goals the plan achieves or the number of unsatisfactory side effects it avoids.

Cooperation Decisions. As with goal selection, the quality of a decision whether or not to cooperate can only be determined from the global impact it has and so depends on the actions and decisions of other agents. One obvious way to improve the quality of decisions about cooperation is for the agent to become more informed and so in a better position to make the decision. A less obvious way is to make the decisions easier for the agent to make. For example, asking an agent to do something she was going to do anyway presents a very easy decision for a rational agent.

This analysis begins with a discussion of what it means to have compatible viewpoints and how that improves the quality of goal selection and cooperative decisions (i.e., deciding whether or not to cooperate). Part of having compatible viewpoints is for agents to be actively working on the same goals at the same time since it is evidence that the agents see the situation in the same way. As pointed out above, it is not necessarily a good thing if it means that more agents than needed are simultaneously working on a goal. As Figure 5.6 shows, storing past successes leads to better goal selection. The left-hand side of the figure shows that during conversations, agents are working on the same goals at the same time more frequently when learning coordinated procedures. The baseline level of 65.3% is quite high since the general pattern in MOVERS-WORLD problems is for the extra-large box to be accomplished last (and with great difficulty), so a large percentage of the communication revolves around it. The right-hand side of the figure refutes the premise that too many agents

might be working on the same goal at the same time. It measures the number of ticks that agents are idle because there is a resource conflict or because they cannot determine a plan. (Not being able to determine a plan to get a box onto the truck is usually because another agent has control of it.) Notice that operator probabilities cause an immediate and sizeable reduction. This is because lifters stop “fighting over” who will attempt to pick up large and extra-large boxes and attempt to lift them together. In the long run, though, coordinated procedures are more effective.

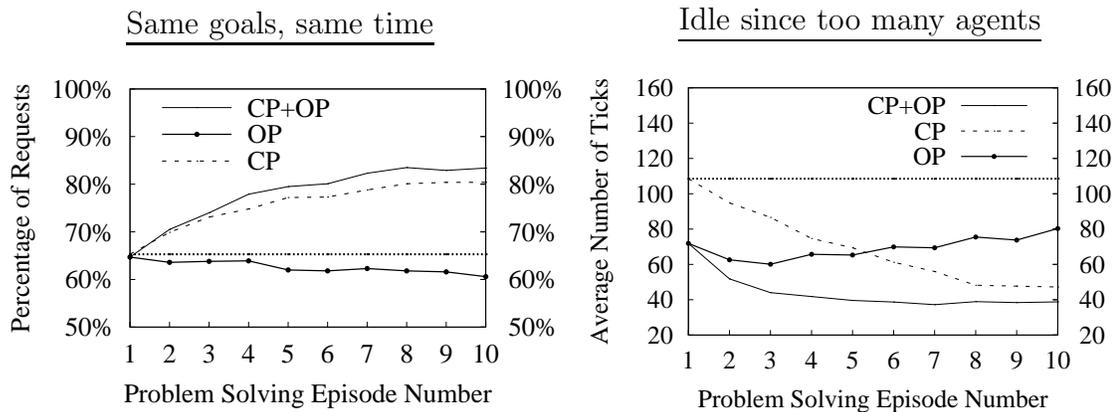


Figure 5.6: Measuring how learning coordinated procedures improves goal selection.

Besides working on the same goals at the same time, another part of having compatible viewpoints is to approach goals in related ways. When the agents have compatible plans, it makes their cooperation decision very easy. Many times it is trivial, since the listener was planning on helping anyway. In most other cases, the response of the listener is still a logical consequence of the relationship between the request and her plan. Conversations are tracked in order to measure when the listener’s plan already includes a coordination point corresponding to the request. Technically, the listener’s plan is considered related to the incoming request if any of the following are true:

1. The incoming request is a request for a service to be provided and the plan contains either:
 - (a) communication about the same request, or
 - (b) a wait operator for the same request.

2. The incoming request is a request for a joint action and the plan contains either:
 - (a) communication about the same action, or
 - (b) a wait operator for the same action, or
 - (c) the same action.

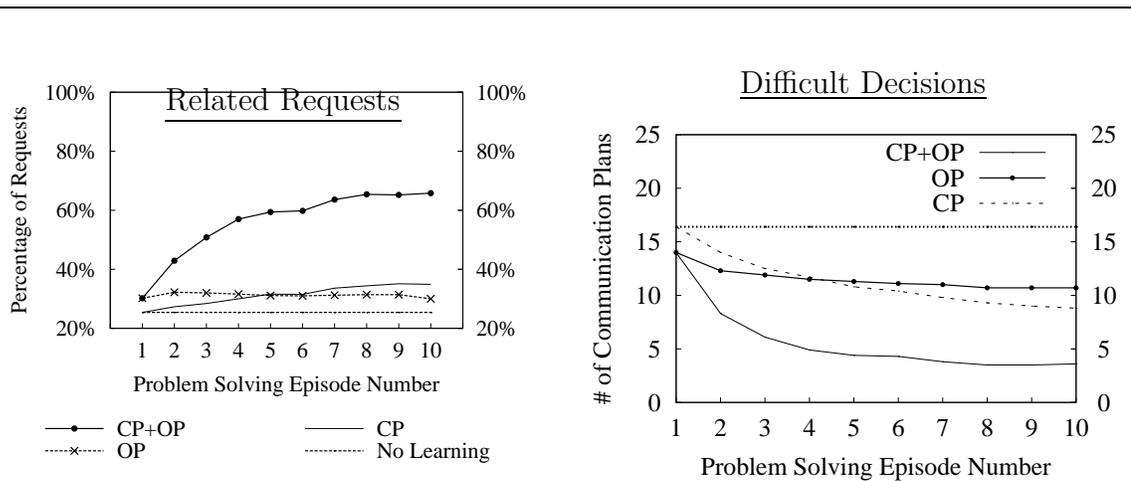


Figure 5.7: Showing how better domain models lead to easier decisions.

The left-hand side of Figure 5.7 measures how frequently two conversing agents have compatible plans. Initially, the listener has a related plan just 25.4% of the time even though (as shown in Figure 5.6) the listener is working on the same goal 65.3% of the time. Over time, neither learning operator probabilities or coordinated procedures makes a huge difference. However, the synergy between the two types

of learning is clearly evident here as the percentage of time that the listener has a compatible plan rises to 65.8%. Another way to see the same phenomena is to count the number of “difficult decisions” an agent faces. The graph in the right-hand side estimates this by the number of times agents construct plans during dialogs since an agent only constructs a plan if there is no logical response based on the relationship of her plan to the request. So, learning better models of the domain allow agents to make better cooperative decisions by approaching goals in compatible ways, leading to fewer difficult decisions to make.

Agents also make better cooperative decisions as they learn operator probabilities because the accuracy of their plan evaluations increases. An agent is a better plan evaluator as her probability estimates approach the “correct values.” If these values are close to 0 or 1, the agent is more likely to have disparate evaluations of the quality of alternate plans. The agent will thus be more likely to be working from a plan with a high evaluation since the baseline planner returns the plan with the highest evaluation. Since the agents are making decisions rationally (see Appendix B.1 for details of this and other cooperation strategies), the agent will switch goals if the alternate plan has a higher evaluation. This is less and less likely as the agent continues to successfully execute actions from her current plan.

Figure 5.8 shows how frequently agents refuse to agree to help when they receive requests unrelated to their current plan. It shows that agents become more attached to their current plan when they are learning operator probabilities, especially so when they are also learning coordinated procedures. Presumably this reflects the fact that the agents highly rate the plans they retrieve from their case-base. These data can be read that agents are making higher quality decisions, but it could also be read in other ways. Finding a better way to track the quality of cooperative decisions is difficult in a dynamic system such as MOVERS-WORLD.

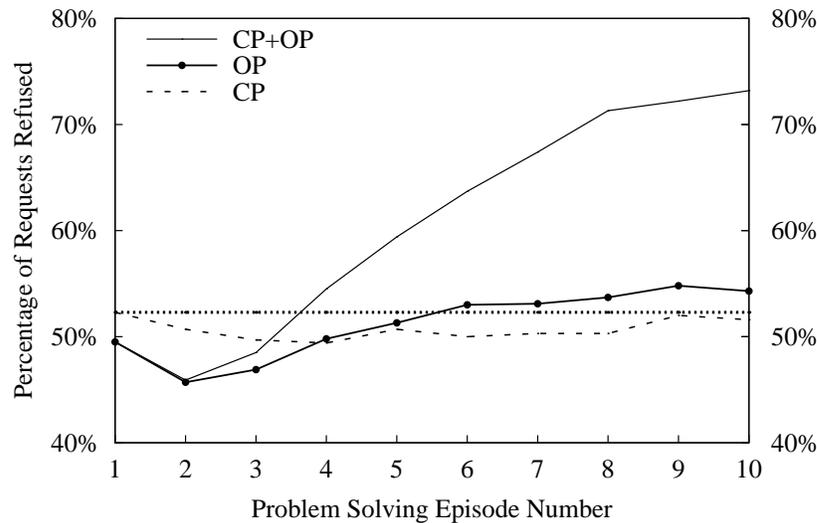


Figure 5.8: CP+OP agents become attached to their plans.

The notion that agents are becoming uncooperative (i.e., more likely to refuse to switch to other, unrelated goals) is supported by the fact that agents are solving problems using the minimal number of successful actions more frequently. Recall that Figure 5.4 gave a macroscopic view, measuring the action effort of the community to solve the entire set of top-level goals. One can also define an alternate, myopic measure of how efficiently the agents are solving individual top-level goals in isolation. For a given goal, the number of actions can be computed that the agents undertook to achieve the goal. This can be compared to the fewest possible given the initial configuration of the problem. For example, a small, clear (i.e., nothing is on top of it) box inside the house requires a minimum of three actions to get it onto the truck: LIFT, CARRY and LOAD. Then, for all problems and all top-level goals, the frequency with which the community undertakes the minimal number of actions can be computed.² The results are given in Figure 5.9.

²Since the bulk of the actions that manipulate the boxes are attempted by lifters, this part of the analysis primarily relates to the quality of their plans rather than those of hand-truck operators.

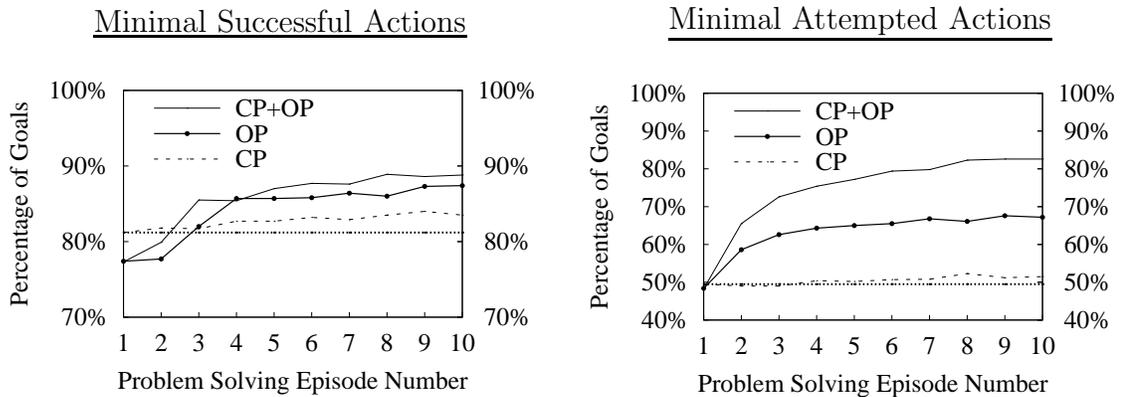


Figure 5.9: Approaching minimal action effort.

Approaching minimal successful action effort is difficult, since it is not enough to merely reproduce past successful behavior. One way that agents can execute fewer actions is to be uncooperative in the right circumstances. For example, if L1 has already executed `<LIFT SBOX1>` and plans to execute (`<CARRY SBOX1 STREET>` `<LOAD SBOX1 TRUCK>`), then the number of successful actions required to get SBOX1 on the truck will be fewer (and minimal) if L1 sticks to that plan rather than agreeing to load SBOX1 onto the hand-truck. In general, a goal will be solved using fewer successful actions if an agent who is in the middle of sequence of actions that will successfully achieve that goal refuses to switch to an unrelated goal when asked for help.³ This argument is supported by the fact that the community achieves minimal successful action effort on 11.5% more of the goals when learning operator probabilities alone (and so are not learning new procedures).

³This is not certain since the yet-to-be-executed portion of the plan may be very inefficient. For MOVERS-WORLD, this is extremely unlikely, so this possibility is ignored. Furthermore, it is should be emphasized that being uncooperative in such situations is beneficial as far as this local measurement is concerned; globally it may not be for the best.

The additional plans available to the agents when they learn coordinated procedures also results in more frequently solving goals while executing the minimal number of actions. For example, if the lifters have a plan available that uses the hand-truck they are not forced to try to carry an extra-large box jointly to the street. Besides reducing the number of failing actions by preventing the attempt to CARRY-TOGETHER, it reduces the number of successful actions by preventing the PUT-DOWN-TOGETHER that frequently follows the failed CARRY-TOGETHER as well as the extra LIFT-TOGETHER needed later in order to load the hand-truck at the hand-truck operators request. When the agents learn coordinated procedures but are not learning operator probabilities, the benefit is slight because the agents are not opinionated enough at the outset of each problem-solving episode to realize how much better using the hand-truck is than carrying the extra-large box.

The right-hand side of Figure 5.9 measures how frequently agents attempt the minimal number of actions to accomplish isolated goals. Without learning, the agents in MOVERS-WORLD are attempting the minimal number of actions for 49.5% of the goals. When learning only operator probabilities, this jumps to 58.6% for the second problem-solving episode and then inches up to 67.2% at the end of the learning curve. This upward trend indicates that agents are learning plans that are more likely to succeed over time. In MOVERS-WORLD, this is most commonly manifested as lifters not bothering to try to lift large boxes alone. However, they do not know what to do about extra-large boxes, so they are forced to attempt (and fail) to lift them alone or carry them together since these are the options when planning from scratch. The lifters escape this rut when the hand-truck operator intervenes with a request to load the hand-truck. On the other hand, the coordinated procedures case-base will contain past successful experiences involving extra-large boxes and the hand-truck so the lifters have a third alternative at planner time that does not need the guidance

of a request from the hand-truck operator. When combined with learning operator probabilities, learning coordinated procedures leads to a large improvement (from 67.2% for OP only to 82.6% for CP+OP). This large improvement implies that the agents are using coordinated procedures that have information outside of the scope of the initial model of the domain, namely using the hand-truck.

Another way to see how the agents are learning plans that are more likely to succeed is to measure how well agents know other agents' capabilities. The left-hand side of Figure 5.10 shows how frequently a listener responds by saying she is not capable of accommodating a request. Initially, it was impossible for the requestee to satisfy over 13% of the requests she received; learning operator probabilities quickly reduced this to around 4% and combined learning reduced it to just over 2%. Agents mismatch their requests more frequently when they use coordinated procedures without the guidance of operator probabilities, since they are pre-disposed to joint actions without having learned whom should be asked for help.

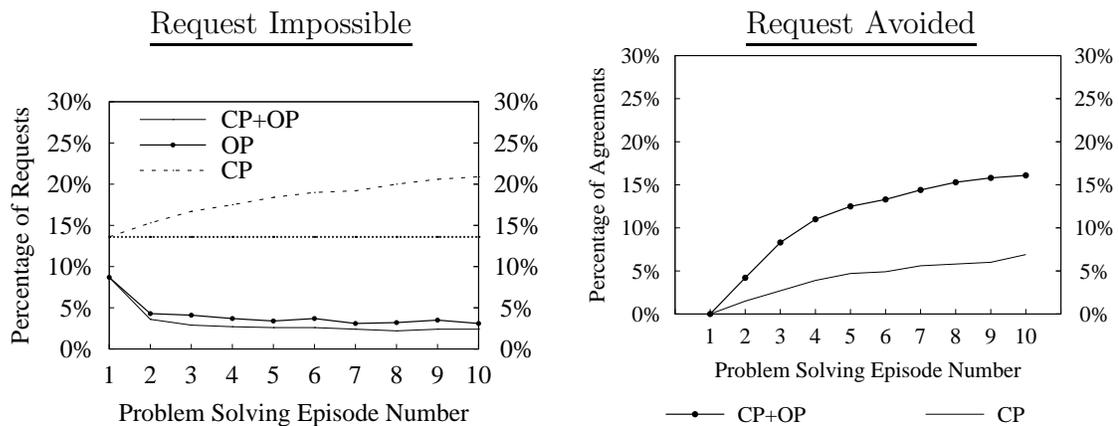


Figure 5.10: Learning who to ask and when not to ask

It was mentioned earlier that coordinated procedures contain new information, particularly in the form of lifters' plans involving the hand-truck. Another piece of information in coordinated procedures that is not present in first-principles plans are implicit waits. These implicit waits represent places where an agent would normally explicitly make a request, but, based on past experience, will instead hope to avoid communication altogether. Measuring when communication is avoided requires comparing satisfied implicit waits (*SIW*) and explicit agreements made during communication (*EA*). A formula deriving the percentage of requests avoided is $SIW/(SIW + EA)$. This reflects the fact that each time an implicit wait is satisfied without any communication, the agents have, in effect, had a conversation in which the request was made and agreed-to. The right-hand side of Figure 5.10 shows that the percentage of explicit conversations that were avoided rises steadily, exceeding 16% by the last problem-solving episode. The flip side, which is not shown in a figure, is that sometimes the implicit wait will not be satisfied and the agent has to make an explicit request anyway. In this case, the period of time spent waiting was completely for naught. In *MOVERS-WORLD*, this never averaged more than 0.8 times per problem.

Section 4.3 showed how unplanned, but efficient, goal interleavings can be stored as coordinated procedures. Figure 5.11 shows that the case-base of past successful behavior contains plans that interleave goals and that the agents use them over 12% of the time. The graph shows how frequently agents are actively working on two different top-level goals, measured when agents converse. Although possible to include plans that accomplish more than one goal at a time sequentially, they are not stored away for future re-use in the current version of the system. The agent can instead determine a plan to accomplish the first goal and then later, if still appropriate, determine a plan for the second goal. In other words, a sequential plan does not constitute an

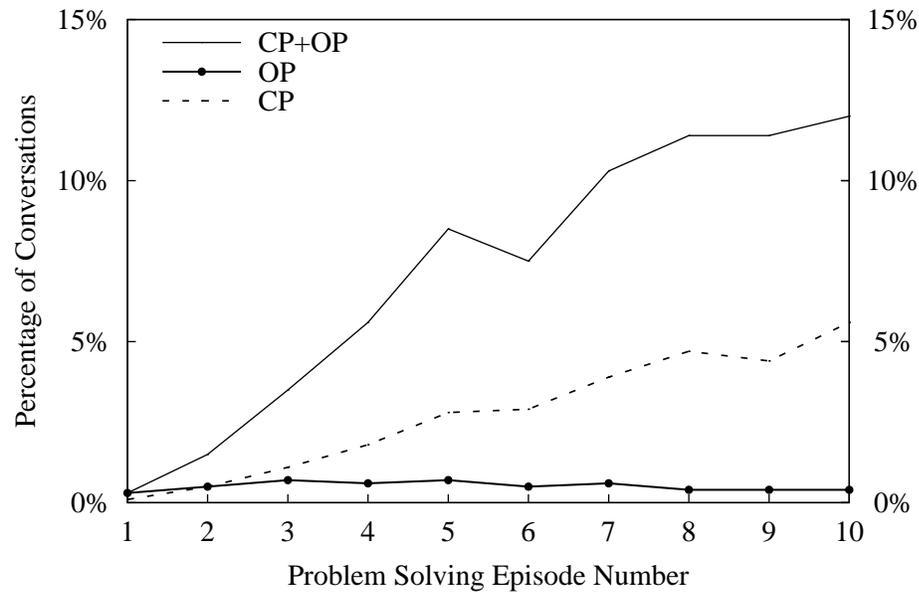


Figure 5.11: Coordinated procedures lead agents to simultaneously work on two top-level goals more frequently.

addition to the domain model of the agent. Note that on rare occasions agents work on two goals at once without the benefit of coordinated procedures. This happens when a lifter is holding one box and agrees to work on another box. As a subgoal of working on the second box, the lifter must free her hands, and does so by loading the first box on the truck.

To recap the arguments advanced in this section:

Goal selection. The quality of an agent’s goal selection is improved by the development of compatible viewpoints that results from using a case-base of past successes. Another contributing factor is that learning operator probabilities prevents agents from individually attempting goals that require the help of others.

Plan Determination. The quality of plans that an agent determines improves over time because:

- Learning better probability estimates via operator probability trees leads to first-principles plans that are more likely to succeed.
- Coordinated procedures represent ways to solve problems outside of the scope of the baseline planner. Agents learn when they can quietly assume help will be given (e.g., hand-truck operators learn implicit waits) and contain new ways to combine operators (e.g., the lifters learn about the hand-truck).
- The coordinated procedures case-base contains past, successful ways to interleave subgoals.

Cooperation Decisions. The development of compatible viewpoints means that agents are faced with fewer hard decisions. Agents are better at making hard decisions because operator probabilities allow them to learn more accurate plan assessments.

5.4 The benefits of coordinated procedures

The main results of this chapter show that learning coordinated procedures significantly increases community performance, whether or not the community is learning operator probabilities. The last section listed several reasons why the case-base of procedures led to improved performance, but the support for those reasons was indirect. In other words, the reasons were inferred based on second-order statistics. To be thorough, a series of five experiments were run in which the system architecture was changed and then the problems were solved by the community anew. This provides

the first-order statistics needed to quantify the benefits of the plans and viewpoints stored in the case-base.

The emphasis of this section is on attributing the difference in performance between OP agents (i.e., agents using operator probability trees to learn) and CP+OP agents (i.e., agents using both learning structures), so in all experiments the agents will be learning operator probabilities. The first three experiments are lesion studies, in which the agents will be deprived of some of the functionality of the coordinated procedures case-base. In the fourth experiment, OP agents will be given additional knowledge to enable them to perform as well as agents using a restricted version of CP. In the final experiment, when CP+OP agents are also given this additional knowledge, they outperform OP agents and do so with less planner effort. In the graphs of this section, error bars (for 99% confidence intervals) will be shown for only the middle curve in order to reduce clutter. The unshown error bars are of the same width as the shown one.

Experiment 1. The coordinated procedures case-base is hypothesized to benefit the community by improving the goal selection of the agents and also by improving the quality of the plans at their disposal. In order to isolate the importance of each of these effects, the system can be re-run with the agents only having partial access to the case-base. Figure 5.12 shows the impact that limiting access to the case-base has under two scenarios. On the left, agents can use the case-base in order to make goal-selection choices, but the plans to accompany the selected goal must be generated by the first-principles planner. On the right, the agent has access to the plans in the case-base, but must select goals to plan for by the first-principles strategy. The benefit of the stored viewpoints is not statistically significant with even 95% confidence (the standard error is 13.22 and the difference in the means is 24.05). Acting from the stored plans leads to performance that significantly differs from both

OP and CP+OP, showing that the guidance of the viewpoints is an important part of the benefits of the coordinated procedures case-base.

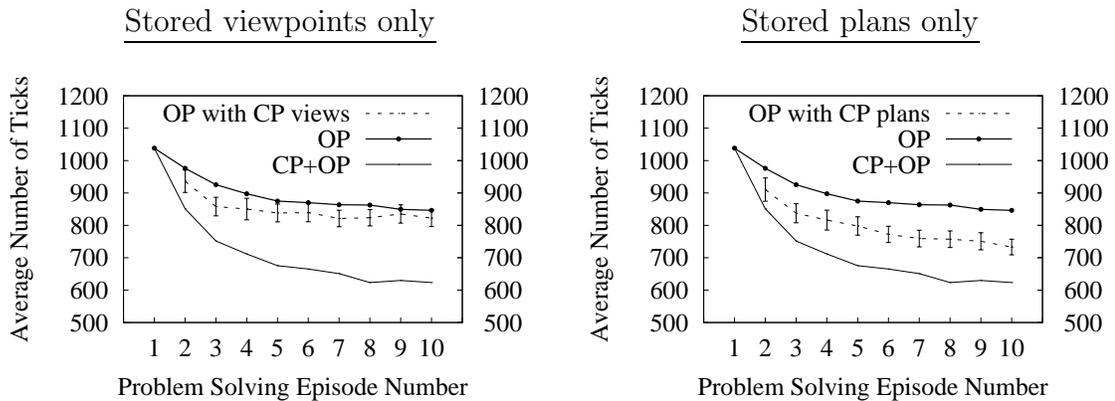


Figure 5.12: Contributions of pieces of the coordinated procedures case-base.

In order to ensure consistency between these graphs and previous results, each simulation generates one data point at a time (rather than the usual ten). For example, to generate a point for problem-solving episode six, a simulation is run such that the agents had full access to the case-base for the first five problems and limited access for the sixth. To generate a point for problem-solving episode seven, the simulation is run such that the agents had full access to the case-base for the first six problems and limited access for the seventh. So these two data points have to be generated by different simulations.

From the curves in Figure 5.12, it is clear that the benefit of the stored plans outweighs that of the stored viewpoints. Taken separately, stored viewpoints improve the performance in the tenth episode by 24.1 ticks and stored plans by 113.2 ticks. Combined, this gives 137.3, which is less than the difference between the OP curve and the CP+OP curve (222.8). In other words, the two pieces contribute independently and there is a synergy between them.

Experiments 2 and 3. The second experiment is designed to see how much of the benefit of the stored plans is due to the heuristic optimizations that MOVERS-WORLD agents perform. Figure 5.13 compares the number of ticks and the number of conversations when the system is run with or without optimizations; the OP curve is shown for comparison. The optimizations have a significant effect, but do not account for a large part of the disparity between the OP curve and the optimized curve.

Examining the two CP curves in Figure 5.13 shows learning is faster and performance better when the optimizations are performed. As would be expected, the optimizations significantly reduced the number of dialogs — even more than the 16% predicted by Figure 5.10. This is because the optimizations were twofold: not only were signs changed to implicit waits, but also WAIT-FOR-REQUESTs were dropped entirely. And dropping these waits reduces communication since when a WAIT-FOR-REQUEST remains in an agent’s plan too long, the agent will initiate a conversion.

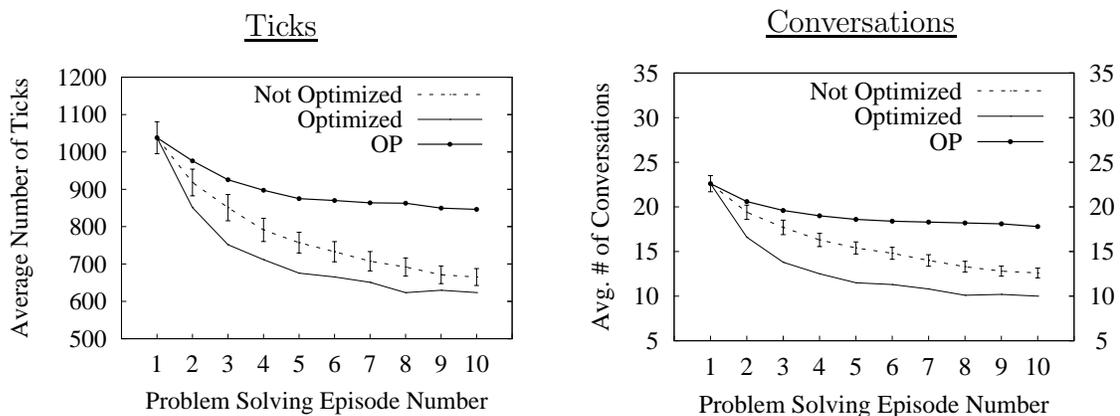


Figure 5.13: Contributions of the optimizations on ticks and conversations.

The third experiment measures the benefit of lifters learning plans to achieve multiple goals. For this experiment, the results of which are shown in Figure 5.14,

the system was changed so that agents do not remember coordinated procedures that accomplish more than one goal. In one case, for the curve titled 'Single Goal', this was the only change. In the other case, titled 'Restricted CP+OP', this change was in addition to not utilizing the heuristic optimizations. The standard coordinated procedures learning curve is shown, titled 'Full CP+OP'.

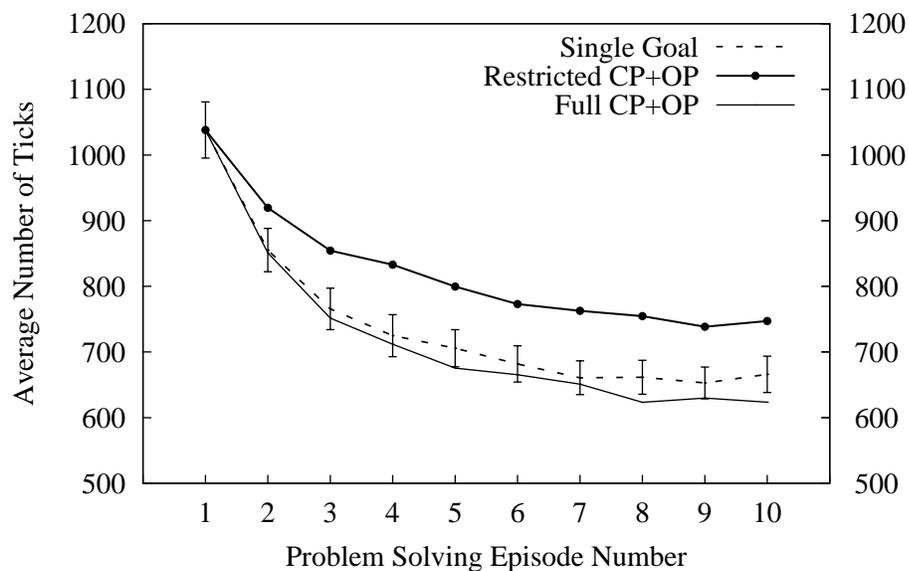


Figure 5.14: Contribution of plans to achieve multiple goals.

One can see that storing chunks with more than one goal did not have a significant impact. Two reasons for this are both related to the dynamic nature of the environment. One reason is that agents rarely are able to successfully execute an entire plan if the plan is long, such as one to achieve several goals. The other is that the environment can be a boon as well as a bane in the sense that the agents are able to interleave goals successfully at run-time even without having access to such plans in memory.

The curve for the restricted method of learning coordinated procedures (abbreviated as restricted CP+OP) is shown on the graph to give a realistic measure of what OP agents might be able to produce if given adequate additional knowledge. Since there is no principled way to incorporate the heuristic changes into first-principles plans, they are excluded. And without replacing the baseline planner to generate plans that efficiently interleave goals, which is outside the scope of this thesis, plans to accomplish multiple goals must be excluded as well.

Experiment 4. So far, the experiments have removed pieces of the coordinated-procedures case-base. The analysis now takes the opposite approach by examining the effect of building additional knowledge into OP agents. The goal of the rest of this section is to see what needs to be added to OP agents to allow them to solve problems as effectively as the restricted CP+OP curve in Figure 5.14. Adding common knowledge to the community is for experimental analysis only and not as a method for improving the performance of the system.

The most important piece of domain knowledge that OP agents lack is that lifters do not know how to work with the hand-truck. In order to measure the benefit of building this in as common knowledge, the system was re-run when the lifters had additional planning operators similar to those of the hand-truck operator. For example, L1 can generate the following plan:

```
<SIGN HTO (AT HANDTR3 PR11-ROOM1)>
<SIGN L2 (LOAD-TOGETHER L2 PR11-XLBOX2 HANDTR3)>
<LOAD-TOGETHER L2 PR11-XLBOX2 HANDTR3>
<MOVE PR11-STREET1>
<SIGN HTO (AT HANDTR3 PR11-STREET1)>
<SIGN L2 (UNLOAD-TOGETHER L2 PR11-XLBOX2 PR11-LBOX3)>
<UNLOAD-TOGETHER L2 PR11-XLBOX2 PR11-LBOX3>
<SIGN L2 (LOAD-TOGETHER L2 PR11-XLBOX2 TRUCK3)>
<LOAD-TOGETHER PR11-XLBOX2 TRUCK3>
```

The left-hand side of Figure 5.15 graphs the number of ticks required to solve problems when the lifters have this additional knowledge about the hand-truck. The graphs compare the performance of OP agents with common hand-truck knowledge to both restricted CP agents and to OP agents without common hand-truck knowledge. The benefit is immediate; the number of ticks needed to solve the first problem drops from 1038.1 to 955.8. However, common knowledge of the hand-truck accounts for just 35.0 ticks of the 99.1 tick difference between the performance of OP agents and restricted CP+OP for the last problem-solving episode. This is not surprising since much of the benefit of the additional plans in the coordinated procedures case-base was due to the synergy with the goal-selection strategy.

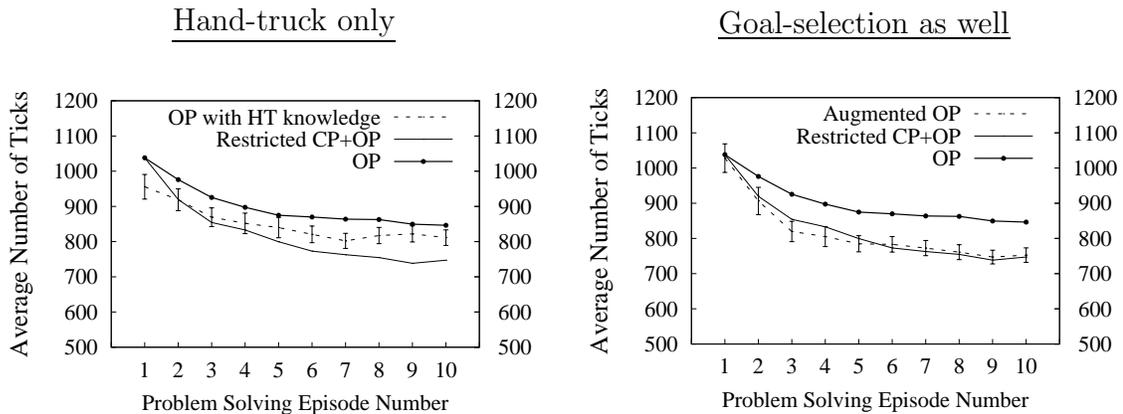


Figure 5.15: Matching coordinated procedures to common knowledge.

The right-hand side of Figure 5.15 shows that the combination of building in common knowledge of the hand-truck and a domain-specific goal-selection strategy (described below) make the performance of the OP agents and restricted CP+OP agents nearly identical. The domain-specific goal-selection knowledge is counter-productive at first. This is because agents are very poor at coordinating their efforts to move the extra-large box to the truck with their original probability estimates. The improve-

ment in the performance of OP agents is not without a cost; the addition of more planning operators leads to more planning search, as shown in Figure 5.16. This graph shows that the average number of planning nodes expanded during each call to the baseline planner rises from 38.6 to 63.4 (for the tenth problem-solving episode) when the lifters have hand-truck knowledge. When the community is forced to use the domain-specific goal-selection strategy, this average climbs to 74.2

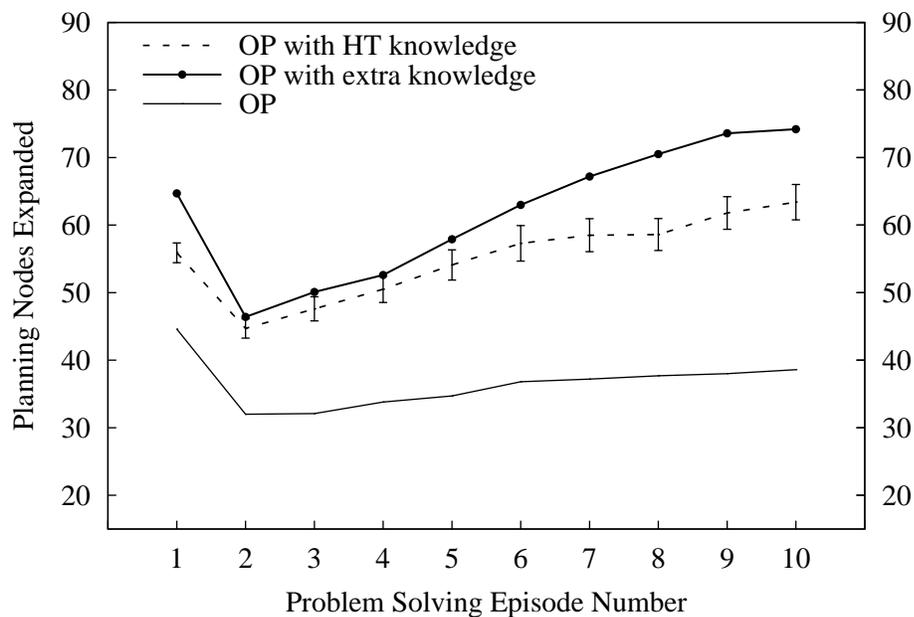


Figure 5.16: Common knowledge leads to more planner search.

The baseline goal-selection strategy is to first consider goals that the agent has not previously failed to achieve; if none, consider all unmet goals. From this set of potential goals, one is randomly selected to be active. Using a case-base improves on this by looking for a plan in memory that achieves a subset of the potential goals; if so, this supersedes the random selection. (See Appendix B.4 for an analysis of goal-selection strategies in general.) In practice, the case-base produces a bias for agents to work on extra-large boxes early in the problems, since many case-base entries involve

extra-large boxes. To mirror that bias for this experiment, OP agents select goals using a domain-specific strategy that supersedes random selection. After the extra-large box is on the truck or the agent has failed at the goal a fixed number of times (four) already, the agents revert to random selection. This strategy is effective (in the long-run) but requires that size (rather than object dimensions) is considered an observable characteristic, which is normally not the case.

Given this additional knowledge, the OP agents can be reasonably described as having complete and accurate domain knowledge after solving a sequence of problems. All agents start learning with the same general knowledge of the domain represented in a type-specific form. While this agents can not use this knowledge effectively at first, when all operator probabilities are 50%, they can by the last problem-solving episode. While they are not designed with a concept of type, they have learned which agents will be able to help lift a box or stand a hand-truck. Therefore, the performance of these agents is the basis for the 'Engineering' curve in Figure 1.2 of the introduction.

Experiment 5. A skeptic will look at the results in Figure 5.15 as evidence that learning coordinated procedures only has benefits since the lifters lack knowledge of the hand-truck and the baseline goal-selection strategy is so impoverished. This misses an important point. The methodology of learning coordinated procedures is domain-independent, as is the goal-selection strategy. Nonetheless, they combine to achieve the same level of performance that is possible by pre-engineering the key pieces of knowledge. And, as the fifth experiment will show, CP+OP agents significantly outperform OP agents when given the same knowledge to work with.

The previous comparison hardly did coordinated procedures justice. No optimizations were performed and plans to accomplish multiple goals were not put into the

case-base. Furthermore, the OP agents had knowledge available to them that the restricted CP agents did not, so that in unfamiliar settings (where the case-base provides no help), OP agents were at an advantage. To even the playing field, the system was re-run so that agents had the benefit of this knowledge when they were learning coordinated procedures as well. Since the forced goal-selection strategy for OP agents would not help agents when using their case-base, an equivalent mechanism guiding retrieval was added. To be specific, until the extra-large box was on the truck or the agent had failed at it four times, only plans in the case-base that involved the extra-large box were retrieved. This mechanism rarely changes the output of the basic retrieval mechanism, which does not require that size is an observable feature.

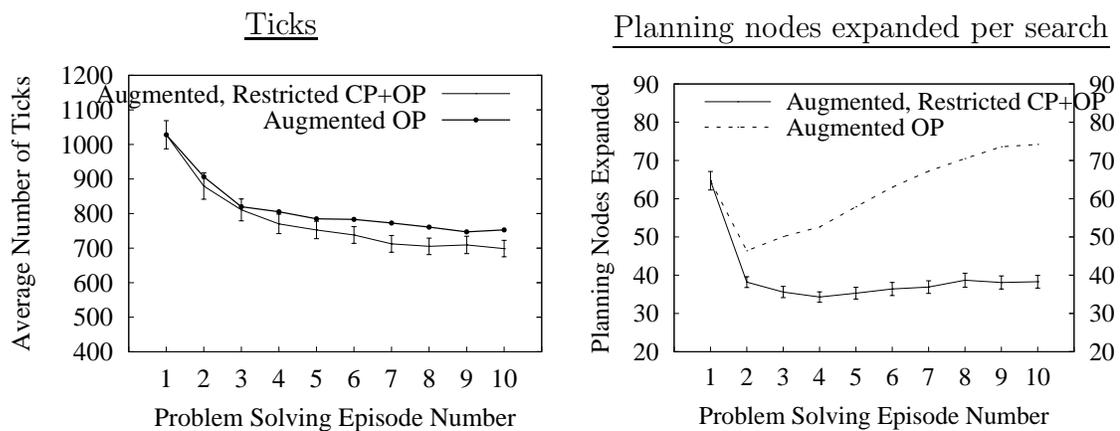


Figure 5.17: Coordinated procedures improve upon built-in knowledge.

Figure 5.17 shows that restricted CP improves performance when measured on a level playing field. The reduction in the number of ticks is much smaller than without the additional common knowledge, but it is statistically significant. For the tenth problem-solving episode, restricted CP drops from 747.2 ticks per problem without common knowledge to 698.6 with the knowledge; by contrast, the OP agents solve the tenth problem in an average of 752.6 ticks.

Figure 5.17 provides evidence that there is leverage in storing and retrieving plans based on the surface features of the environment, rather than having access to similar information that is accessed at separate times. Another advantage of learning coordinated procedures is in controlling the amount of planner effort, as demonstrated on the right-hand side of Figure 5.17. The addition of common knowledge led to a dramatic rise in the number of planning nodes expanded for the OP agents, but the number of nodes expanded remained nearly constant when learning coordinated procedures.

Summary of the experiments. The results of the previous section are summarized in Figure 5.18. In this graph, the best⁴ of the OP agents with common built-in knowledge from Figure 5.17 is labelled “Engineered Knowledge”. This line represents agents who have the requisite general knowledge and accurate probabilities to perform at high levels. The curve labelled “Learning with Engineered Knowledge” is the restricted CP+OP curve from the same figure. Finally, “Learning w/o Engineered Knowledge” is the first restricted CP+OP curve shown in this section (on Figure 5.14).

Figure 5.18 shows that learning coordinated procedures benefits agents who are engineered with common knowledge; this is because the agents learn to attempt problems in compatible ways. However, there will always be coordination problems that are not anticipated by the engineer and must be solved by the agents at runtime. Figure 5.18 also shows that even agents who do not have additional planner knowledge and goal-selection information can achieve high levels of performance.

⁴This corresponds to the ninth problem-solving episode.

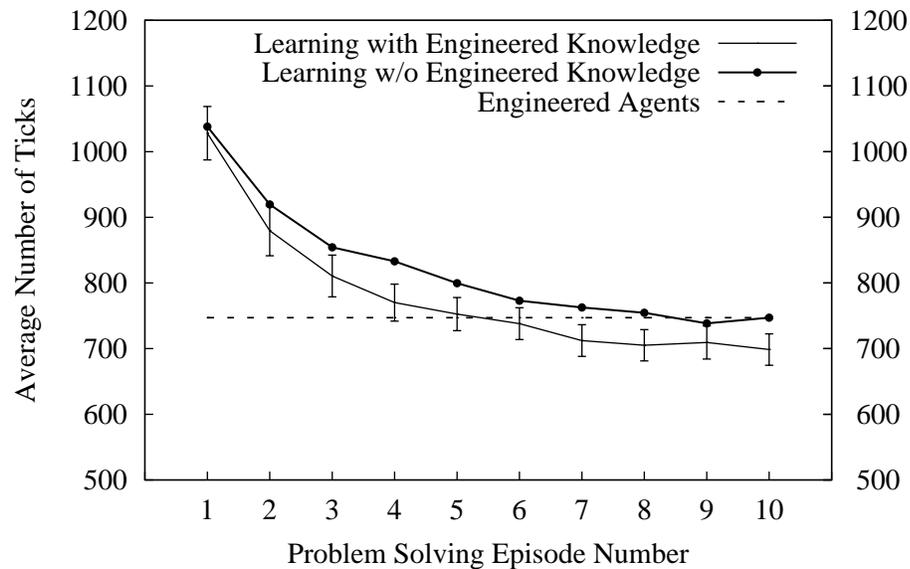


Figure 5.18: Summary: learning agents perform at high levels. (Reprint of Figure 1.2.)

Figure 5.18 is the graph presented in the introduction since the comparisons are fair. On the other hand, the addition of built-in common knowledge does not lessen the other benefits of coordinated procedures that are not part of restricted CP (plans to achieve multiple goals, optimizations). Figure 5.19 demonstrates this fact by comparing the engineered agents to two less restricted versions of learning coordinated procedures. (Restricted CP is also included for completeness.)

The curve labelled “Interleaving CP+OP” shows the performance of the community when agents learn plans with multiple goals, but do not use the heuristic optimizations. For “Full CP+OP,” agents optimize coordinated procedures as well. Acting effort and planning effort are much less for both of these runs than the engineered agents because of the introduction of additional knowledge. While this knowledge is outside of the scope of the baseline planner, it is acquired and used with minimal extra effort by the techniques of this thesis.

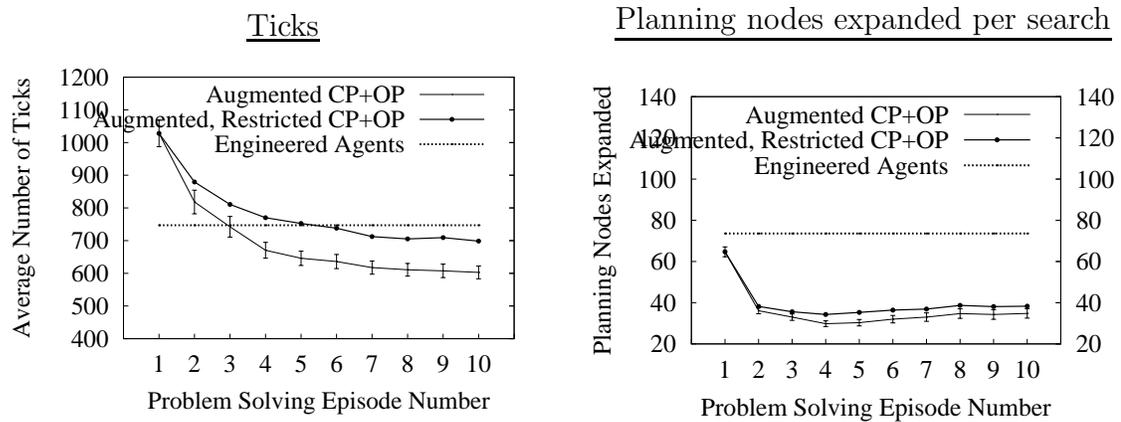


Figure 5.19: Revisiting optimizations and interleavings with built-in knowledge.

In conclusion, coordinated procedures contain the key pieces of domain knowledge needed by the community to improve performance. Furthermore, even if common knowledge is built into the agent architecture, coordinated procedures provide a statistically significant improvement in behavior. This is true with or without taking advantage of optimization techniques. When common knowledge is built into the system, planner search for agents who are not guided by coordinated procedures climb dramatically. Overall, these results indicate that agents automatically acquire and efficiently use essential domain knowledge with domain-independent techniques.

Chapter 6

Architecture details

This chapter contains information about several important pieces of the architecture that have yet to be described. The key pieces are the baseline planner of the agents and the implementation of communication. Another aspect of the system that is described is how the agents learn operator probabilities from their interactions with each other and the domain and how these probabilities effect the baseline planner. Also, several details related to how execution-time dynamics are handled are presented.

Figure 6.1 is a flowchart depicting the major components in the agent architecture. Agents loop through this flowchart until no agent has unmet goals left. At the beginning of each tick of activity, the agents who completed events (acting, talking, idling) at the end of the previous tick start working through the flowchart, starting at the top left-hand corner, labelled “Active Goals?”. At the outset of the activity, the agent has a set of top-level goals that are unmet but an agent does not start working on them until they become active. Simple decision points like this are in diamond shaped boxes. Two major decision points are labelled “Coordinated Procedure Found?” and “Baseline Plan Generated?”. These are more complex processes whose final result is tested. Finally, there are three nodes that have dashed outlines, labelled

“Idle”, “Converse”, and “Attempt Action”. The result of entering these nodes depends upon the dynamics of the world and on the other agents. Although unshown, these dashed boxes then return to the “Active Goals?” test.

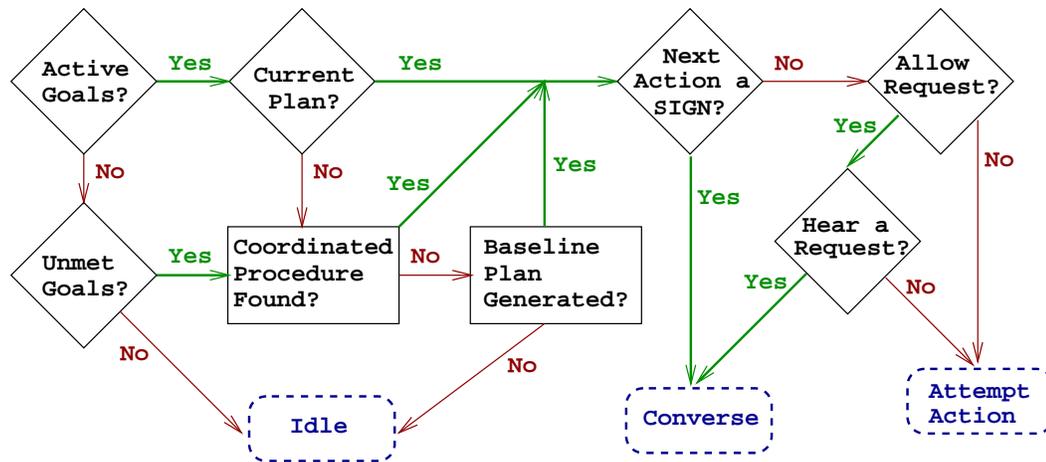


Figure 6.1: Flowchart of agent activity cycle.

6.1 The baseline planner

The agents’ first-principles planner is a STRIPS-based (Fikes and Nilsson, 1971), hierarchical planner which uses probabilities to guide search (cf. Kushmerick *et al.*, 1995). The heuristic used to guide planner search is based on expected reward per tick, where reward is computed by the agent, not provided as feedback from the environment. The expected reward per tick of a plan depends on how many goals it achieves as well as the probability of success and anticipated duration of the individual actions that make up the plan. The pseudo-code for `plan-rating` given in Appendix C computes the heuristic in general circumstances, including second-principles plans that include wait operators or plans that do not complete any goals.

Computing reward per tick requires knowing the duration of agent actions. Each action will last for a duration of several ticks, depending upon a user variable called *ATP* (Act To Plan ratio). For example, a MOVE action takes $0.625 * ATP$ ticks to complete. During the execution of a single primitive action, agents do not engage in planning. After the completion of the action, successful or not, the agent refreshes her state of world assessment and may adapt her plan or plan from scratch. Typically, *ATP* is set to 40 which yields the durations for successful actions (arbitrarily, failed actions take half as long as successful ones) shown in Table 6.1.

Action(s)	Duration
TILT-HANDTR, STAND-HANDTR	20 ticks
MOVE	25 ticks
PUSH-HANDTR	32 ticks
LIFT, LIFT-TOGETHER	32 ticks
PUT-DOWN, PUT-DOWN-TOGETHER	32 ticks
LOAD, LOAD-TOGETHER	35 ticks
UNLOAD, UNLOAD-TOGETHER	35 ticks
CARRY, CARRY-TOGETHER	40 ticks

Table 6.1: Typical MOVERS-WORLD action durations.

Unlike primitive actions, the durations of conversations is indeterminate at planning time because it is unknown how the other agent will respond. Instead, the duration is estimated from Formula 6.1 in Section 6.4 by assuming the dialog will contain a fixed number (20) of symbols. This leads to an estimate of 25 ticks for a conversation; in experiments, the average was around 20. An obvious extension to the learning framework is to treat action and conversation durations as learning issues. Agents could start with an uniform set of durations that are revised based upon execution-time feedback.

Computing expected reward per tick also requires estimates of the probability of successfully executing actions. The default probability for all primitive and commu-

nicative actions is set at 50%, but the agents would be able to solve problems as long as the default was greater than 0%. Given this, the formula used to determine $Prob(action)$ results in probabilities for actions that have never failed or succeeded that approach, without ever reaching, 1.0 or 0.0, respectively. This reflects a belief that actions are never sure things or hopeless. $Prob(action)$ is computed by the formula below, where S is the number of past successes and A is the number of past attempts.

$$Prob(action, S, A) = \begin{cases} S/A & \text{if } 0 < S < A \\ 1/(2 + 2^A) & \text{if } 0 = S \text{ and action is communicative} \\ 1 - 1/(2 + 2^A) & \text{if } S = A \text{ and action is communicative} \\ 1/(2 + A 2^A) & \text{if } 0 = S \text{ and action is primitive} \\ 1 - 1/(2 + A 2^A) & \text{if } S = A \text{ and action is primitive} \end{cases}$$

The asymptotic probabilities change more slowly for signs and waits out of practicality rather than theory. If they degraded as fast as for a primitive action then an unfortunate sequence of initial refusals (which happens, but rarely) would push the planner (guided by these probabilities) out of the portion of the search space that includes possible solutions. On the other hand, if primitive action probabilities decayed as slowly as communicative actions, learning will slower, but still effective. To make the above formula more concrete, Table 6.2 shows the asymptotes for small numbers of always unsuccessful attempts.

Hierarchical planners reduce planning costs by focusing search on the most important details first. In a traditional hierarchical planner such as ABSTRIPS (Sacerdoti, 1974), planner search starts at a level corresponding to highest criticality level assigned to a precondition (presently 4 for MOVERS-WORLD) and when a full plan

Attempts	Primitive Action Probabilities	Communicative Action Probabilities
1	25.0%	25.0%
2	10.0%	16.7%
3	3.8%	10.0%
4	1.5%	5.6%
5	0.6%	2.9%

Table 6.2: How probabilities change for failures.

is constructed, the level is decreased and the full plan is used as a skeleton to guide search at the lower level. Within a level, search is typically a variant of depth-first search. Unmet preconditions of operators are only planned to be achieved when the planning level is at or below the criticality level of the precondition. When a full plan is constructed at the lowest level assigned to a precondition (1 for *MOVERS-WORLD*), that plan is returned as the result. If no full plan can be constructed at a level using a particular skeleton plan, planning backtracks to the next higher level planning queue. *ALPINE* (Knoblock, 1990) improves on this by automatically determining the criticality levels; unfortunately, in the worst case, this process is as computationally difficult as planning.

The baseline planner deviates from this approach somewhat. The planner maintains a queue of planning objects, sorted by the heuristic function, and so is a variant of best-first search instead of depth-first search. Each planning object has its own planning level associated with it, in addition to a partial plan, a goal-stack, and a representation of the current state. As in traditional hierarchal planners, unmet preconditions of operators are planned to be achieved when the planning level of the planning object is at or below the criticality level of the precondition. In order to focus planner search on the most promising plans (i.e., those at lower planning levels), the heuristic presently used is the expected reward per tick for the estimated completion of the plan divided by the planning level of the plan object. This means

that actions in the goal-stack as well as unmet preconditions in either the plan or the goal-stack are taken into consideration during the computation.

The final place where criticality levels play a role in the baseline planner is in determining role-bindings. Role-binding selections are determined by a separate best-first search that is guided by a slightly different variant of expected reward per tick. For each set of ground values being considered as fillers for the set of variable role bindings of the operator, the set of post-substitution preconditions is checked to see which n of them are unmet in the current state of the world. Also, the post-substitution effects are determined to compute the reward of the state of the world after executing the (possibly incompletely instantiated) operator. The heuristic function then returns $prob * reward / dur$, where $prob = 0.5^n$ and $dur = ATP \sum_{i=0}^n level_i$.

6.2 Operator probabilities

Besides learning coordinated procedures, agents can improve their behavior by generalizing their experiences interacting with the world. This allows the planner to explore areas of the search space that are more likely to be productive. *Operator probability trees* are used to estimate the probability of success for actions an agent may attempt. Accurate estimates improve individual performance since the agents' planner search, including role-binding determination, is guided by probabilities. Operator probability trees are incrementally updated as the agent interacts with the domain; this enables the planner to produce higher-quality plans during the course of solving a problem.

To get a feel for why operator probabilities are needed, consider a common decision a lifter must make. A lifter can lift some boxes alone, but not all of them. If a lifter can lift the box alone, it is more efficient to do so because they do not have to spend time asking for assistance (and there is no guarantee the assistance will be given). On

the other hand, if there is little chance that the lifter can handle the box on its own, it is a waste of time (and energy) to make the attempt. So, individuals should be able to recognize which of these two possibilities is more likely and act accordingly.

Initially, MOVERS-WORLD agents use default operator probabilities of 50%. This means an inexperienced lifter will prefer to try to lift boxes alone because a plan to do so will have a shorter duration and thus will have a higher expected reward per tick. So, at the beginning of a problem, lifter L1 might try to pick up a large box LBOX1 by herself. L1 would fail; the next time she wants to lift LBOX1, L1 may decide to ask for the help of another agent. This decision is based upon L1's specific past experience interacting with that particular box.

Unless there is a mechanism for L1 to generalize its past experiences, L1 will repeatedly attempt (and fail) to lift other large boxes alone, wasting time and effort. Operator probability trees prevent this from occurring. If the observable characteristics (e.g., height, width, depth, material) of another box LBOX2 exactly match LBOX1, L1 will not attempt to lift LBOX2 alone since the general past experience in this case will include the failed attempt to lift LBOX1. If the features do not match exactly and there are other experiences stored in the tree, the general past experience may or may not be based upon LBOX1 depending on whether the COBWEB classification algorithm considers the feature set of that experience to be the best match with the current feature set.

To recapitulate, the agent estimates the overall likelihood of success for an action from probabilities based on both specifically related experiences (the same action with identical role fillers) as well as more abstractly related general experiences (actions and/or role fillers that are similar). The preceding section described how to compute the probability of an action based on specific experience. The following paragraphs will describe how to compute a probability based on general experience. If only one

of the two types of experience is available, the overall likelihood of success is set to it. If neither is available, a default probability (50%) is used. If both are available, the lower value is used if the specific probability is less than or equal to the default, else the higher value is returned.

The successes and failures of attempted actions are stored in a COBWEB (Fisher, 1987) tree associated with all of the observable features of the various role fillers for the action. We assume that actions might fail for reasons not explicitly considered (q.v. the *qualification problem*: McCarthy, 1977) and COBWEB can handle noisy data. Also, COBWEB trees can be updated incrementally, which allows the agents to learn during the course of their activity. Figure 6.2 gives an algorithm that computes a probability of success from the general past experience stored in these trees. The probability is determined by general-probability, which massages the data returned by COBWEB-classify. COBWEB-classify classifies action in the COBWEB tree by the observable features for the action's role fillers and returns all trees with maximal matching factors found via beam search.

Note that general-probability in Figure 6.2 uses a similarity measure that differs from the matching factor used by the COBWEB algorithms. COBWEB algorithms, both when classifying with or updating trees, divide the number of features that are common to both the tree being evaluated and the input data by the total number of features. However, the operator probability trees are constructed from merging different actions that have different types and numbers of role fillers. In order to keep the total number of features constant for COBWEB, the list of role fillers for an action is expanded to fit a standard template. For example, the template for lifters' primitive actions is: one other agent, two locations, four objects, and two vehicles. This results in numerous features that may take on the value *unknown-value*. If all values associated with a feature are unknown, both in the tree being evaluated

and in the input data, the feature is ignored in the calculation of similarity done by general-probability.

```

weighted-average prob default similarity
  return ( prob * similarity ) + ( default * ( 1 - similarity ) )

general-probability action default features
  set trees = COBWEB-classify ( features )
  if no trees return default

  foreach tree do
    excluding features that have the value *unknown-value*:
      similarity = # of matching features / total # of features
  set trees = trees with maximal similarity
  set similarity = maximal similarity
  if all trees contain only successes or failures
    set tree = tree with most data
    set prob = prob ( action, successes (tree), attempts (tree) )
  else set prob = prob (action, sum-of-successes (trees),
                        sum-of-attempts (trees) )

  set sim-weighted-prob = weighted-average ( prob, default, similarity )

OPTIONAL:
if trees has just one member AND the member is a leaf node with a parent
  set pa = parent
  set pa-sim = similarity (pa, features)
  set pa-prob = prob (action, successes (pa), attempts (pa) )
  if sim-weighted-prob <= default
    set sim-weighted-prob =
      min ( sim-weighted-prob,
            weighted-average ( pa-prob, default, pa-sim ) )
  else set sim-weighted-prob =
    max ( sim-weighted-prob,
          weighted-average ( pa-prob, default, pa-sim ) )

return sim-weighted-prob

```

Figure 6.2: Algorithm to calculate general operator probability.

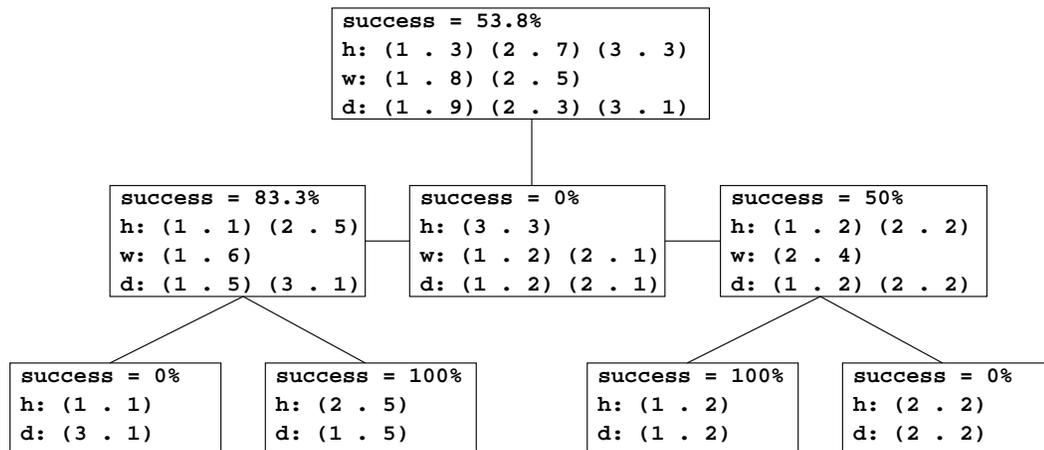


Figure 6.3: Operator probability tree.

See Figure 6.3 for a sample tree that can be used to predict the probability of success for an agent who is considering attempting to lift a 2 by 2 by 2 box by herself. The three features shown in the figure are object height (h), width (w) and depth (d). In a given node, each feature value is paired with the number of observations in the sub-tree rooted at that node that have the same feature value. For example, d: (1 . 5)(3 . 1) means that five observations had depth 1 and one observation had depth 3. For the data underlying this tree, the agent cannot lift a box alone if any dimension of the box is 3 or if all the dimensions are 2.

The success probability based up the general experience in the lower right hand node shown in Figure 6.3 is 16.7%. (For comparison, if the node has contained 1, 3, 4, or 5 unsuccessful attempts, the probability would have been 25.0%, 10.0%, 5.6%, or 2.9% respectively.) In the absence of specific experience, 16.7% would be the overall operator probability as well. In this particular case, 16.7% would also be the overall probability if there were a single specific failure, but not if there were more than one specific experience.

DAEDALUS (Langley and Allen, 1991) included a learning mechanism that also relied upon probabilities and COBWEB. However, there are several important differences between DAEDALUS' use of probabilities and learning operator probabilities. Given a state description, DAEDALUS selects the operator (and partial role bindings) that are most likely to be the correct choice based upon past planning experiences. In this way, DAEDALUS learns to produce a single plan in more situations; it does not produce more than one plan or provide a mechanism for comparing plans. By contrast, MOVERS-WORLD agents can compare alternate plans based upon their respective likelihood of runtime success. In MOVERS-WORLD, agents store past execution experiences in order to map actions to the probability of successfully executing them and can combine the operator probabilities into a measure of the probability of runtime success for an entire plan. Other efforts to use COBWEB to improve planner efficiency are Yang (1993) and Fisher and Yoo (1993).

6.3 Execution-time dynamics

This section is a hodge-podge. Principally it recounts the methods that the system uses to manage the execution-time dynamics of an activity. In the descriptions below, the phrase “an agent is delayed” or words to that effect appear several times. This should be interpreted that system forces the agent to be idle (execute a NO-OP) for the current tick; afterwards, the agent may decide to remain in an idle state (continue to execute NO-OPs).

At the beginning of each tick of activity, agents who completed events (acting, talking, idling, or waiting) at the end of the previous tick update their view of the world, including the status of their goals. Unless they have no unmet goals, they then either adapt their current plan or determine a new one. The outcome of the resulting

set of plans is then determined by the system. The functions the system performs, in order, are:

1. It manages the communication among agents. This involves randomly determining who calls whom, if there is more than one agent seeking to make a request. It also involves delaying callers if they cannot establish a connection, which happens for one of three reasons:
 - (a) The callee started conversing with another agent in this tick. In other words, the system had to randomly select among several attempted connections, and the caller was not selected.
 - (b) The callee had started an action attempt in a prior tick, and the attempt has not ended yet.
 - (c) The callee may have decided to temporarily stop taking requests. In *MOVERS-WORLD*, this occurs most frequently when agents agree to immediately undertake a joint action, but can also occur if an agent is interrupted by a request for a goal they are currently working on. For example, if a hand-truck operator is about to stand up a loaded hand-truck and receives a request to ready the hand-truck for unloading, she will refuse to answer any more communication until executing the *STAND-HANDTR*. With three agents and relatively fast communication, this is a minor point but with more agents and/or slower communication, an agent can be indefinitely delayed from making forward progress by a barrage of inquiries (after several agents ask initially, they then start calling again because it has not been achieved yet).

2. If an agent has a joint action as the next step in her plan, check that all other participating agents are attempting it at that time. If not, the agents intending to attempt it are delayed.
3. The system does not allow multiple agents to have control of the same physical resource, such as a box or a hand-truck, at the same time except via joint actions. The system determines if any attempted actions will conflict other actions to be attempted at that time. In this case, a winner is randomly selected from the set of conflicting actions. The losing agents are delayed, as is an agent who attempts an action that conflicts with an ongoing actions.
4. The system determines whether attempting actions succeed or not, based upon a set of execution-time conditions unknown to the agent. These attempts all begin as of the beginning of the current tick.

The domain features that determine whether actions are possible are object size, object weight, agent strength, and item capacities. None of these are observable features of the environment. The static observable box features are height, width, depth and material - each measured in integers between 0 and 3. Static observable vehicle features are just the vehicle name and its type (for MOVERS-WORLD, this is :HAND-TRUCK or :TRUCK). Static location features are just the type (for MOVERS-WORLD, this is :ROOM or :STREET) and agent features are just the agent name. There is just one dynamic feature per MOVERS-WORLD item: the number of items covering the item (if the item is involved in an ongoing action then the value is set as *NOT-FREE*). Since COBWEB works on categorical values, it is irrelevant that there may be a mixture of numeric and non-numeric values for this feature.

For boxes, the size is derived from the volume of the box and the weight is a linear function of the size and material. The label associated with a box includes a S, M, L or XL solely in order to make it easier for the human reader to parse the system output. Item capacities specify how many other items and how much total weight the item can hold.

5. The system updates its list of ongoing actions, adding all actions that start at the beginning of this tick and removing all actions that finish at the end of this tick.
6. The system updates the current state of world. Since actions have a duration longer than one tick, the standard STRIPS-style way of incorporating their effects does not apply. If the add effects are reflected in the state of the world immediately upon commencement, the hand-truck operator would think it reasonable to start tilting the hand-truck before the lifter has finished loading it. On the other hand, if the delete effects do not change the state of the world immediately, there will be no external indication that the action is being attempted. Therefore, delete effects are removed from the state of the world as soon as an action is started. When the attempt is finished, either the add or delete effects are added to the state, depending on whether the action succeeded or failed.

6.3.1 Frustration

Frustration is modeled via a linear progress estimator. The estimator starts at zero and an agent is deemed frustrated when it reaches a pre-set threshold (currently $4ATP$). The estimator is changed in the following circumstances:

- Frustration is increased by *ATP* when the agent fails at her current plan.
- Frustration is increased by *ATP* when the agent adapts her plan by adding actions to achieve preconditions that no longer hold.
- Frustration is increased by 1 for each tick the agent is idle.
- Frustration is increased by 1 for each tick the agent waits.
- Frustration is increased if the first step in her plan is a WAIT-FOR-REQUEST and she has not had previous conversations regarding the wait operator. If the wait is for a request for joint action, frustration is set to the maximum, otherwise it is increased by 1 in addition to the 1 mentioned above. Whether or not to incorporate these increases is optional, controlled by a system parameter.
- Frustration is increased by 1 for each tick the agent is involved in a conversation that the agent neither initiated nor agreed to help. In other words, the agent views this as wasted time.
- Frustration is decreased by *ATP* when the agent adapts her plan by dropping unnecessary actions since the sub-goal they achieve has become true.
- Frustration is decreased by *ATP* when the agent is told their request is true.
- Frustration is reset to 0 when the agent is told a top-level goal they are working on is already true.
- Frustration is reset to 0 when the agent is told to wait in a conversation.
- Frustration is reset to 0 when a request for service is completed.
- Frustration is reset to 0 when the agent determines a plan that begins with an IMPLICIT-WAIT.
- Frustration is reset to 0 when the agent starts wandering.

Wandering is an *ad hoc* solution to a problem the agents face since they have incomplete state of world knowledge. If an agent is already frustrated and fails to generate a plan to accomplish a goal, then the agent will instead generate a plan to go to a randomly selected (but different) location in the hope of discovering new information about the state of the world. Also, an agent will wander when she has lost track of the location of object literals (e.g., a box or the truck) in a top-level goal or if they fail to be able to find a plan for the goal, even if she is not frustrated. This second criteria is reasonable in MOVERS-WORLD, but perhaps not in other domains. In either event, it is a minor matter that can be included or not. Occasionally, one agent wanting to wander will leave another agent without an active goal temporarily:

```
Ticks 781 to 800: <CARRY-TOGETHER PR24-XLBOX1 PR24-STREET1> by L1 and L2 failed
Ticks 801 to 817: L2 and L1 converse
    "L1, would you help me achieve (HE)
      via (PUT-DOWN-TOGETHER PR24-XLBOX1 PR24-ROOM1)?
      This is to help me wander."
    "L2, sure! Lets get to it."
```

6.3.2 Idling

Agents are idle (execute a NO-OP) when they have a wait as the first action in their plan. Another situation in which agents idle is if they have no current plan. This occurs if they have no more unsatisfied goals or they cannot construct a plan to achieve the active top-level goal (Appendix B.4 talks more about selecting top-level goals to actively pursue). For example, a lifter cannot construct a plan for a box another lifter is holding or whose location is unknown to her. While agents idle, they will not act but they will respond to communication and so, possibly, stop idling by agreeing to help another agent. A full list of the situations in which an agent idles is given below.

Waiting for a connection. The agent was idle because she could not establish a communication connection. An agent's attempt to communicate with another agent may be unsuccessful since the other agent's communication channel may be busy or the other agent may be refusing to answer.

Resource conflict. The agent was idle because of a resource conflict. Currently, the outcome of resource conflicts is decided at random and the "losing" agents idle.

No current plan. The agent was idle because she had no plan for the currently active goals. Given accurate state knowledge, an agent is generally capable of generating a plan for any give top-level goal. However, there are a few circumstances in which when a lifter cannot generate a plan even with accurate state knowledge, such as when another lifter is already holding a box. Also, the agent often has incomplete or partial knowledge. As an example, if a lifter carries a box out of the current room, the remaining agents "lose track" of both the lifter and the box.

All goals true. The agent was idle since she believed all of her goals were true. In the current runs, this only happens at the end of a problem when an agent (invariably the hand-truck operator) is not at the street and does not observe the last box being loaded onto the truck. If the agents are given disjoint goal sets, however, agents could be idle for this reason much more frequently.

Extreme frustration. The agent was idle after she gave up on her current plan out of extreme frustration. (The idle steps after giving up are cleansed; any preceding WAITs are kept for now.) The typical case is when the waiting agent makes multiple abort request and is repeatedly told to wait (perhaps by an incompe-

tent agent, perhaps by a nefarious one). Currently, the maximum time an agent will wait after making an initial explicit request is $FrustrationThreshold * MaxWaitFactor$, where the default value for $MaxWaitFactor$ is 3. For these experiments, that corresponds to 480 ticks of waiting.

A pathological case when agents abandon a plan out of frustration is when an agent's frustration level reaches $FrustrationThreshold$ in the same tick that she adapts her plan to add a new (i.e., about a different coordination point) request to the top of the plan. The reason for this is that the request generator labels all requests made when the agent is frustrated as :ABORTs, which is not correct for the new request. The default way to circumvent this is to drop the plan rather than to reduce the agent's frustration level.

Delayed joint action. The agent was idle since not all agents that should be participating in a joint action are. Joint actions are an all-or-nothing proposition. Unless all relevant agents participate, those that attempt to participate are delayed. In the current experiments, lifters turn off their communication channel momentarily when agreeing to attempt a joint action now. This keeps the lifters synchronized in most situations by preventing a request from the hand-truck operator to delay the attempt. However, it is possible for the world to change during the conversation so that after the conversation ends, one of the agents adapts or abandons her plan.

No goals to pursue. The agent was idle because her goal-selection strategy did not specify any of the unmet goals to actively pursue. This does not occur for the default goal-selection strategy.

After being idle for one tick, an agent calls a function that determines whether the agent will either continue to delay action for a short period or will abandon the

current plan and goals. Currently, the agents all use the same function and choose to delay action. The length of time agents are idle depends on the reason they are idle:

- If an agent is idle because she has no plan, she will stop idling if she perceives a change in the environment. Also, she will stop idling if she has other unmet top-level goals, unless the last effort to achieve those goals resulted in not being able to generate a plan. These criteria are meant only to save a modest amount of CPU time. A change of state does not guarantee that a plan can be generated, but it has the advantage of not incorporating reasoning outside of the planning process.
- If an agent is idle since there is a resource conflict, they will idle until the conflicting item becomes available again. If the agent becomes frustrated, the agent will abandon their current goals and stop idling.
- If an agent is idle because a communication connection cannot be made, the agent idles until the connection is established.

6.4 Communication details

Communication between two agents in this system is similar to a telephone conversation. agents do not have to be in the same location to engage in communication. One agent can call another and either establishes a connection, gets a busy signal (only two agents can participate in a single negotiation — as opposed to *contract nets* (Davis and Smith, 1983)) or the other agent refuses to respond. The actual exchange of information is done by request and response frames (Minsky, 1975): the principle slots in a request frame are request type and the request itself, which is a coordination point that the requester needs assistance with. Section 2.3 discusses the choice of

coordination points as the communication language in more detail. The duration of a conversation is determined by the number of symbols exchanged between the two agents plus a small communicative overhead.

Communication only occurs at run-time, and consists of the exchange of triples in the form of (S, communication-type-code, optional-joint-action-name). There are 21 communication type codes (only 6 are essential); agents must be designed to process them all, and generate only codes within that alphabet for responses. Joint action names are passed along in cases where the sender needs to distinguish between several alternate ways to achieve a subgoal. If the recipient has no knowledge of the joint action name, she refuses to help.

At execution-time, the duration of a conversation is computed as follows:

$$ConvDuration = round(CommOverheadRatio * ATP) + \sum_{frame} round(CommRatio * FrameSymbols(frame)) \quad (6.1)$$

where *CommOverheadRatio* and *CommRatio* are user variables, with default values of 0.125 and 1.0 respectively. *CommRatio* models the transmission rate of the community; a higher ratio means that more ticks are needed to have a conversation. All conversations last a minimum of a tick, regardless of the values for *CommOverheadRatio* and *CommRatio*.

All frames contain a ‘type’ slot, which contains a symbol that determines the semantic meaning associated with the frame. The number of symbols that are on a given frame varies depending on the type of the frame. The following subsections cover different types of frames and describe the contents of the frames in detail.

6.4.1 Initial requests

Initial requests are of type :HELP, :PAGE, or :ABORT.

Help frames are used to request cooperation with another agent for a coordination point. Help frames are glossed as "OTHER-AGENT, would you help me achieve COORDINATION-POINT." Help frames are derived from a SIGN operator in the requester's plan.

Page frames are derived from a WAIT-FOR-REQUEST in a learned coordinated plan (see Section 7.4) after the requester has become frustrated while waiting for the request to be made. Page frames are glossed as "OTHER-AGENT, I was expecting you to ask me to achieve COORDINATION-POINT."

Abort frames are derived from a WAIT operator added to the requester's plan in a previous conversation after the requester has become frustrated while waiting for the request to be completed. Abort frames are glossed as "OTHER-AGENT, I became frustrated waiting for you to achieve COORDINATION-POINT."

Help and Page frames may not be the first attempt to achieve the coordination point, but if not, it is the first exchange with the other agent after a failure. On the other hand, Abort frames are always a follow-up to a previous agreement to a Help or Page frame.

For requests for joint action, the glosses shown above are followed by "via JOINT-ACTION-NAME." Unless COORDINATION-POINT is a top-level goal, the gloss is completed with "This is a subgoal to the top-level goals TOP-LEVEL-GOALS."

Besides slots for COORDINATION-POINT and TOP-LEVEL-GOALS lists, the frame contains a slot for the JOINT-ACTION-NAME symbol (which is nil for requests for service), and a 'sub-type' slot.

The sub-type is normally nil, but is set in certain cases to be either :NO or :CANT. This means the requester is either refusing or incapable of satisfying an

alternative request to accomplish COORDINATION-POINT and is re-stating their original request. In this case, the three slots mentioned above do not count towards the frame cost since they do not have to be re-transmitted.

6.4.2 Basic responses

Basic responses are of type :OK and :NO and signal the end of the conversation.

The basic response frames contain single symbols for the sub-type and an ‘agree’ slot. The sub-type is either :READY or :WAIT for OK frames and either :NO or :CANT for NO frames. The value of the agree slot is the requestee’s view of the status of the current agreement between the two agents to achieve COORDINATION-POINT. If the agent is still working on the past agreement, the value is T. If there has been no change in status since the last conversation about it, the value is nil. Other possible values are :COMPLETE, :DROPPED, :FAILED, and :NOHELP (failed since got no help from another agent).

6.4.3 Alternative request responses

Some responses are in the form of requests to accomplish a different coordination point or to accomplish COORDINATION-POINT in a different manner. Alternative requests are of type :ALTHELP, :ALTPAGE, or :ALTABORT.

These alternative requests can only be made in response to initial requests, so there are no circularity problems. They have all of the same slots as initial requests and as basic responses. In addition, there is an ‘another’ slot, which indicates that the agent will give another, different response if the initial request is re-stated. This is set in cases where the initial request and the alternative request both accomplish COORDINATION-POINT but via different JOINT-ACTION-NAMEs.

ALTPAGE and ALTABORT requests are made even if the agent is not yet frustrated since hearing an unexpected request is considered sufficient evidence that the agents are no longer coordinated. ALTABORT requests can directly follow :PAGE requests but much more frequently occur after an initial :HELP request followed by a :STATUS and a :STATUS-RESPONSE responses (described next).

6.4.4 Other request responses

These responses are all requests, but are not proposing a new alternative. The types can be :STATUS, :STATUS-RESPONSE, :LOST, :FOUND, :UNFOUND and :SUPPORT. The responses :STATUS-RESPONSE, :FOUND, :UNFOUND and :SUPPORT are implicit requests for the other agent to reconsider the original request, whereas :STATUS and :LOST request specific pieces of information.

:STATUS requests are made when an agent receives a request for help on one coordination point when the agent is waiting for the requester to accomplish another coordination point. In this way, the agent discovers whether the requester has opted out of the previous agreement. The slots used are the agree slot of basic responses (giving the agent's view about the new request) and the COORDINATION-POINT slot of requests (for asking about the coordination point of the previous agreement).

:STATUS-RESPONSE frames always follow :STATUS requests. It contains an agree slot and an active slot. The value of the active slot is either T (still working on the previous agreement), :BYP (will accomplish the request as a byproduct of my current plan), or nil.

Here is a sample conversation to demonstrate how :STATUS and ALTABORT requests differ. In this example, the frame types exchanged are :HELP, :STATUS, :STATUS-RESPONSE, :ALTABORT (subtype :CANT), and :OK (subtype :WAIT).

The (program-generated) gloss for the ALTABORT request should really read "continue working on" rather than "renew working on", but the same gloss is used if the :STATUS-RESPONSE frame indicated that L1 had opted out.

Ticks 92 to 129: HTO and L1 converse

"HTO, would you help me achieve (HOLDING-TOGETHER PR12-LBOX3)
via (LIFT-TOGETHER PR12-LBOX3)?

This is part of a plan involving you to achieve (ON PR12-LBOX3 TRUCK3)."

"L1, first I want to know about (ON PR12-LBOX3 HANDTR3)."

"HTO, I am still working on it."

"L1, I can't help, but I want you to renew working on (ON PR12-LBOX3 HANDTR3)."

"HTO, I'll help, but you'll have to wait a bit."

:LOST requests are made when there are one or more items in the previous request that the agent has either lost track of or the agent's current state of world indicates was in use. The frame contains an agree slot and a 'lost' slot. It is followed by either a :FOUND frame, with a single slot containing a list of predicates representing the agent's view of the location of the lost items, or an :UNFOUND frame, indicating that the agent has lost the items too. Currently, the glosses for these frames are content-free, solely due to programming oversight.

:SUPPORT requests are needed because learned coordinated procedures do not include planner-reconstructible actions and, rarely (44 times out of 371,246 conversations in a one experiment), the planner is not able to reconstruct them as hoped due to incomplete state of world knowledge. For MOVERS-WORLD, this usually is a failure to be able to expand an anticipated request to load the hand-truck, as in the following example. In this example, the frame types exchanged are :HELP, :ALTPAGE (subtype :NO), :SUPPORT, and :OK (subtype :WAIT). After the dialog, L1 moves to the location of the extra-large box (PR25-ROOM1) and gets an updated state of world.

Ticks 687 to 753: L2 and L1 converse

"L1, would you help me achieve (HOLDING-TOGETHER PR25-XLBOX2)
via (LIFT-TOGETHER PR25-XLBOX2)?

This is part of a plan involving you to achieve (ON PR25-XLBOX2 TRUCK3)."

```
"L2, no, but I was expecting a request of (ON PR25-XLBOX2 HANDTR3)
  via (LOAD-TOGETHER PR25-XLBOX2 HANDTR3).
  This is part of a plan involving you to achieve (ON PR25-XLBOX2 TRUCK3)."
```

```
"L1, Helping me supports your own goal!"
```

```
"L2, I'll help, but you'll have to wait a bit."
```

6.4.5 Terminal responses

Conversations are terminated by a :BYE frame, which has no slots and no gloss. The basic responses (:OK and :NO) are always followed by a :BYE frame. :TRUE and :UNTRUE frames are the only other type of frames that may be followed by a :BYE frame.

A :TRUE response indicates that the agent believes that there is no reason to work on the request since either COORDINATION-POINT is true or the TOP-LEVEL-GOALS are true. The subtype slot distinguishes between the two cases. The frame also may contain a 'time' slot and a 'location' slot. They are optionally included in order for the listener to possibly reply :UNTRUE. The agent first establishes at which time tick she last determined that the fact went from false to true. Then there are two cases. If the agent came to see the fact as true because of communication from another agent, the time is saved in the time slot and the location of the other agent at that time is saved in the location slot. On the other hand, if the agent believes the fact from direct observation, the location of the agent at that time is saved in the location slot and the time slot is set to be the last time the agent was in that location.

When an agent hears a :TRUE response that contains time and location slots, she can compare it to her view of the world to check that the other agent has reason to know better. However, the implementation depends upon agents having a uniform notion of time. The agent examines her history of the state of world to see if either

(1) she was at LOCATION after TIME or (2) TIME is the present and LOCATION is her current location. The latter case is to handle the situation where a :FOUND response has caused an agent to have an inaccurate state of the world assessment. This can happen because the present implementation only sends predicates that the listener adds to their state of world and does not include any that should be retracted.

```
Ticks 676 to 710: <LOAD-TOGETHER PR12-XLBOX1 HANDTR3> by L1 and L2 successful
  [ snip - removed 4 conflicts involving UNLOADing vs. TILting the hand-truck ]
Ticks 804 to 821: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 failed
Tick 807: <TILT-HANDTR HANDTR3> by HTO delayed
Ticks 807 to 831: <MOVE PR12-STREET1> by L2 successful
Ticks 822 to 841: <TILT-HANDTR HANDTR3> by HTO successful
Ticks 832 to 870: L2 and L1 converse
  "L2, would you help me achieve (READY-TO-LOAD HANDTR3)?
    This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."
```

Although not shown by the gloss, the :LOST frame specified that L2 asked for updated information about HANDTR3 because it was in use when she left the room. The :FOUND response includes several predicates, but does not include either a predicate giving the new status of the hand-truck (since the new predicate will not be true until after tick 841) or any indication to delete the old status. So, L2 replies :TRUE. L2 last identified the predicate as becoming true by direct observation after tick 710, when she was at PR12-ROOM1. And she was last at PR12-ROOM1 at time 806. Since L1 was at that location after that time, she replies :UNTRUE.

6.4.6 Neither agreements nor refusals

Agents can have dialogs that do not end in refusals and yet do not establish new agreements in many different circumstances. This happens most commonly when the listener was already working on the request. Other typical situations correspond to

ongoing dialogs about a previous agreement or to discussions where an agent indicates a willingness to establish an agreement, but not until a later time. An exhaustive list of the possible scenarios where a conversation between agents A and B ends in neither a refusal nor establishing a new agreement (from A 's perspective) is given below.

- A replied :ACTIVE.
- In response to an :ABORT or :ALTBORT, A was told that B was still working on it and to wait some more.
- A replies, or hears a reply, indicating that either the request or corresponding top-level goal is already true.
- B requested a joint action J when the first step of A 's plan is a WAIT for J . This only happens if A had already established an agreement and was waiting for B to indicate her readiness.
- B requested joint action J when the first step of A 's plan is J . This only happens if A already believed an agreement to attempt J was in force and has been delayed in attempting the joint action since B has not been assisting.
- A variant of the above: the first step of A 's plan is a joint action J but the request by B is not a request for J . As an alternative response, A requests B attempt J instead and B agrees. From A 's point of view, this is a continuation of the previous agreement, not the establishment of a new one.
- When A agrees to a joint action that is “roughly equivalent” to a joint action that is on the top of A 's plan. Roughly equivalent means that both achieve

the coordination point. For example, LOAD-TOGETHER and PUT-DOWN-TOGETHER are roughly equivalent if the coordination point is (HAND-EMPTY), but not if it is (ON BOX HANDTR). When this occurs, *A* adapts her plan to replace the intended joint action with *B*'s “roughly equivalent” one.

- *A* made a :PAGE and was told to wait. This communication does not establish the agreement since the :PAGE corresponds to a WAIT-FOR-REQUEST rather than a SIGN. In other words, *B* is indicating a willingness to establish an agreement, but not until a later time. If *B* does make the desired request later, that request establishes the agreement.
- *A* made a :PAGE about a joint action and *B* made the desired request. But, *A* could not adapt the WAIT-FOR-REQUEST into a workable plan when the request was heard because of incomplete or inaccurate state of world assessment.
- While waiting for *B* to be ready to attempt a joint action, *A* has seen the need adapt her plan by adding some new actions (e.g., unload a box and put it down). If *B* makes some unrelated request (because *B* has opted out) while *A* is in the middle of her actions (e.g., having unloaded the box but not having put it down yet), *A* will make an alternative request about the joint action even though she is not ready at the moment. From *B*'s standpoint, this is a new agreement — but not from *A*'s.
- Requests related to the interior of the agent's plan, when the top of the plan is not a coordination point involving the other agent. In these situations, the request is one the agent is “happy” to hear, but happened to hear too early. These include requests of :HELP for joint actions, :ABORT requests for service matching a WAIT-FOR-REQUEST, :PAGEs related to joint actions or WAIT-

IMPLICITs, and :PAGEs matching a WAIT-FOR-REQUEST that is not the first WAIT-FOR-REQUEST in the agent's plan.

The last case requires a bit of explaining. In MOVERS-WORLD, a necessary but insufficient condition for this to occur is for *B* to make a :PAGE for a LOAD-TOGETHER before the lifters have jointly lifted the box. (This happens because the state of world does not allow the lifter to successfully adapt the WAIT-FOR-REQUEST; e.g., the hand-truck is not ready to be loaded). Sufficiency requires that *A*'s plan is more complete; for example,

```
<WAIT HTO (READY-TO-LOAD HANDTR)>
<WAIT-FOR-REQUEST B (LIFT-TOGETHER BOX)>
<WAIT-FOR-REQUEST B (LOAD-TOGETHER BOX HANDTR)>
⋮
```

6.4.7 Distribution of conversations

The following table gives a summary of all of the conversations in the 1,800 runs that were made to determine the learning curves in Section 5.2. Each sequence is given as a list of frame types (with the leading colon omitted) and is followed by the number of occurrences for the sequence, split out by learning techniques. Also, the frame subtype is included for alternate responses and the basic responses; for example OK/READY means the frame type was :OK and the frame subtype was :READY. Also, :STATUS-RESPONSE is abbreviated S-R. Percentages shown are truncated after one decimal place, instead of being rounded.

Table 6.3: Distribution of conversation sequences.

Sequence of frame types	OP Only		CP Only		CP and OP	
	Pct.	Count	Pct.	Count	Pct.	Count
HELP OK/READY	42.6	48,973	46.0	49,695	56.0	43,793
HELP NO/NO	24.0	27,591	9.1	9,928	17.2	13,497
HELP OK/WAIT	10.8	12,462	5.2	5,701	6.4	5,000
HELP OK/ACTIVE	8.5	9,758	5.6	6,051	5.6	4,423
PAGE OK/READY	0.0	0	1.4	1,543	2.8	2,214

Table 6.3: Distribution of conversation sequences (continued).

Sequence of frame types	OP Only		CP Only		CP and OP	
	Pct.	Count	Pct.	Count	Pct.	Count
HELP NO/CANT	2.8	3,289	13.5	14,631	2.6	2,033
HELP ALT/NO OK/READY	3.1	3,656	1.1	1,217	1.5	1,177
ABORT OK/WAIT	1.4	1,693	1.7	1,887	0.7	597
HELP LOST FOUND NO/NO	0.6	701	0.3	380	0.6	529
HELP TRUE	0.2	246	0.3	409	0.6	522
HELP STATUS S-R ALTABORT/CANT OK/WAIT	0.6	787	1.4	1,562	0.6	519
PAGE NO/NO	0.0	0	2.6	2,831	0.5	463
HELP LOST FOUND OK/WAIT	1.1	1,270	0.6	744	0.5	420
ABORT NO/NO	0.8	1,025	1.0	1,171	0.5	392
ABORT TRUE	0.9	1,059	0.4	445	0.5	387
HELP ALT/CANT OK/WAIT	0.1	175	3.9	4,259	0.4	328
ABORT OK/ACTIVE	0.3	407	1.3	1,406	0.4	310
HELP LOST UNFOUND NO/CANT	0.3	352	0.4	464	0.3	294
HELP STATUS S-R NO/CANT	0.4	481	2.0	2,253	0.3	277
HELP LOST FOUND NO/CANT	0.0	81	0.0	73	0.3	239
HELP ALT/NO OK/WAIT	0.0	11	0.0	29	0.1	118
PAGE ALT/NO OK/READY	0.0	0	0.1	167	0.1	109
HELP ALT/NO NO/NO	0.0	8	0.0	0	0.1	87
PAGE OK/WAIT	0.0	0	0.0	33	0.0	67
ABORT LOST UNFOUND NO/CANT	0.2	253	0.0	89	0.0	49
HELP ALT/CANT LOST FOUND OK/WAIT	0.0	31	0.1	190	0.0	37
HELP ALT/CANT NO/CANT	0.0	61	0.0	93	0.0	36
PAGE ALT/NO NO/NO	0.0	0	0.0	10	0.0	26
HELP STATUS LOST UNFOUND S-R NO/CANT	0.0	65	0.0	33	0.0	25
HELP ALTPAGE/NO NO/NO	0.0	0	0.0	5	0.0	25
ABORT NO/CANT	0.0	85	0.0	70	0.0	15
HELP STATUS S-R OK/WAIT	0.0	0	0.0	23	0.0	13
PAGE ALT/NO OK/WAIT	0.0	0	0.0	5	0.0	12
HELP STATUS S-R ALTABORT/CANT NO/CANT	0.0	33	0.0	97	0.0	11
HELP STATUS S-R NO/NO	0.0	10	0.0	9	0.0	10
HELP STATUS TRUE NO/CANT	0.0	14	0.0	15	0.0	9
HELP ALT/NO LOST FOUND OK/WAIT	0.0	45	0.0	1	0.0	9
PAGE LOST UNFOUND NO/NO	0.0	0	0.0	7	0.0	8
HELP LOST FOUND TRUE	0.0	29	0.0	4	0.0	8
HELP STATUS S-R OK/READY	0.0	41	0.0	14	0.0	7
HELP TRUE UNTRUE	0.0	6	0.0	12	0.0	6
HELP ALT/CANT OK/READY	0.0	3	0.0	7	0.0	6
HELP STATUS LOST UNFOUND S-R ALTABORT/CANT LOST UNFOUND NO/CANT	0.0	0	0.1	115	0.0	4
HELP LOST FOUND STATUS S-R OK/WAIT	0.0	7	0.0	27	0.0	4
HELP LOST FOUND ALT/CANT NO/CANT	0.0	9	0.0	6	0.0	4
ABORT LOST FOUND NO/NO	0.0	8	0.0	20	0.0	3
HELP STATUS LOST FOUND S-R NO/CANT	0.0	3	0.0	17	0.0	3
HELP ALT/NO LOST FOUND NO/CANT	0.0	0	0.0	5	0.0	3
PAGE LOST FOUND NO/NO	0.0	0	0.0	4	0.0	3
HELP LOST UNFOUND ALT/CANT NO/CANT	0.0	1	0.0	3	0.0	3
HELP LOST FOUND ALT/NO OK/WAIT	0.0	0	0.0	0	0.0	3

Table 6.3: Distribution of conversation sequences (continued).

Sequence of frame types	OP Only		CP Only		CP and OP	
	Pct.	Count	Pct.	Count	Pct.	Count
ABORT LOST FOUND OK/WAIT	0.0	4	0.0	29	0.0	2
HELP ALT/CANT LOST UNFOUND NO/CANT	0.0	3	0.0	24	0.0	2
ABORT OK/READY	0.0	4	0.0	16	0.0	2
HELP ALT/NO NO/CANT	0.0	3	0.0	15	0.0	2
HELP ALT/CANT LOST FOUND NO/CANT	0.0	2	0.0	11	0.0	2
PAGE LOST FOUND OK/WAIT	0.0	0	0.0	5	0.0	2
ABORT LOST FOUND NO/CANT	0.0	4	0.0	4	0.0	2
HELP LOST UNFOUND OK/ACTIVE	0.0	0	0.0	4	0.0	2
HELP LOST FOUND OK/ACTIVE	0.0	0	0.0	3	0.0	2
PAGE LOST UNFOUND OK/WAIT	0.0	0	0.0	2	0.0	2
PAGE LOST FOUND ALT/NO NO/NO	0.0	0	0.0	4	0.0	1
ABORT LOST UNFOUND ALT/CANT NO/CANT	0.0	1	0.0	2	0.0	1
HELP ALTPAGE/READY HELP NO OK/READY	0.0	0	0.0	1	0.0	1
ABORT ALT/CANT NO/CANT	0.0	2	0.0	0	0.0	1
HELP ALTPAGE/NO LOST UNFOUND						
SUPPORT OK/READY	0.0	0	0.0	0	0.0	1
HELP LOST FOUND ALT/NO NO/NO	0.0	0	0.0	0	0.0	1
PAGE ALT/WAIT NO/NO	0.0	0	0.0	0	0.0	1
PAGE LOST UNFOUND ALT/NO NO/NO	0.0	0	0.0	0	0.0	1
PAGE LOST UNFOUND ALT/NO OK/READY	0.0	0	0.0	0	0.0	1
PAGE NO/CANT	0.0	0	0.0	0	0.0	1
HELP LOST FOUND TRUE UNTRUE	0.0	0	0.0	50	0.0	0
HELP STATUS S-R ALTABORT/NO OK/WAIT	0.0	0	0.0	31	0.0	0
ABORT LOST UNFOUND OK/ACTIVE	0.0	0	0.0	8	0.0	0
PAGE ALT/NO LOST FOUND OK/WAIT	0.0	0	0.0	5	0.0	0
HELP LOST FOUND ALT/CANT OK/WAIT	0.0	0	0.0	5	0.0	0
ABORT ALT/NO NO/CANT	0.0	2	0.0	4	0.0	0
PAGE ALT/WAIT LOST FOUND OK/WAIT	0.0	0	0.0	2	0.0	0
PAGE LOST UNFOUND STATUS S-R NO/NO	0.0	0	0.0	1	0.0	0
PAGE LOST UNFOUND NO/CANT	0.0	0	0.0	1	0.0	0
PAGE LOST UNFOUND ALT/WAIT OK/READY	0.0	0	0.0	1	0.0	0
PAGE LOST FOUND ALT/WAIT OK/READY	0.0	0	0.0	1	0.0	0
PAGE LOST FOUND ALT/NO OK/READY	0.0	0	0.0	1	0.0	0
PAGE ALTPAGE/WAIT NO/NO	0.0	0	0.0	1	0.0	0
HELP STATUS LOST UNFOUND S-R						
ALTABORT/CANT LOST UNFOUND OK/WAIT	0.0	0	0.0	1	0.0	0
HELP LOST UNFOUND OK/WAIT	0.0	0	0.0	1	0.0	0
HELP ALTPAGE/NO SUPPORT OK/WAIT	0.0	0	0.0	1	0.0	0
HELP ALT/NO TRUE UNTRUE	0.0	0	0.0	1	0.0	0
ABORT LOST FOUND ALT/NO NO/CANT	0.0	0	0.0	1	0.0	0
PAGE LOST UNFOUND ALTPAGE/WAIT	0.0	0	0.0	1	0.0	0
OK/READY						
HELP LOST FOUND ALT/NO LOST FOUND	0.0	3	0.0	0	0.0	0
OK/WAIT						
HELP STATUS LOST UNFOUND S-R						
ALTABORT/CANT LOST UNFOUND NO/CANT	0.0	2	0.0	0	0.0	0
Totals		114,759		107,997		78,167

Chapter 7

Learning coordinated procedures

This chapter will describe the techniques that agents use to learn new coordinated procedures from past successful experiences. After the community of agents solves a problem, individual agents add the coordinated procedures to memory, indexed by the goals the procedure achieves and an abstraction of the state of world at the start of the procedure. Coordinated procedures are derived from execution traces and feature the points of coordination that lead to the solution of the problem. The inclusion of information about other agents' requests means that coordinated procedures lie outside of the scope of the agent's original domain model. All agents undertake this process on their own — no common knowledge is available and no learned knowledge is exchanged.

Chapter 4 provided a general overview of the seven steps that go into learning coordinated procedures. After the problem has been solved, an agent (1) cleans her execution trace and (2) segments it into groups of related actions. The rest of the steps work within the context of just one segment, coupled with the goals it achieves, rather than the entire cleaned execution trace. The segments have (3) inefficient behavior removed, are (4) optionally optimized, (5) summarized, and then

(6) converted into a representation that is suitable for (7) storing in a general case library. The techniques of each step are presented along with numerous trace snippets to motivate the rationale behind the techniques. After these seven sections, the last section contains a brief discussion about the issue of over-riding agents' run-time beliefs and intentions during learning.

7.1 Cleaning the execution trace

Cleaning removes all “unsuccessful” events from the execution trace of an agent. The sort of trace events removed in this step are failed primitive actions, refused requests, and idling while waiting to establish a communication connection. Cleaning is not meant to be a summarization phase, although it could be considered a simple one. It is expedient to convert run-time agreements into WAIT-FOR-REQUEST actions during this initial step in the learning process even though it has nothing to do with removing unsuccessful events.

Failed primitive actions lead to learning through operator probability trees rather than through memories in a case-base. Although failure-directed learning is the focus of other case-based reasoning research, in this system the case-base is a means of storing past successes. The environment does not provide predicate-based explanations of the failures needed by causal analysis techniques such as those of Hammond (1990). Ohko *et al.* (1996) present a multi-agent case-base approach that could augment operator probabilities, where agents learn to revise bid and task announcements over a contract net based upon past responses.

Most of the communication removed during cleaning are refused requests, but any conversation that does not establish a new agreement to achieve a coordination point is removed. Agents can have dialogs that do not end in refusals and yet do not

establish new agreements in many different circumstances. This happens most commonly when the listener was already working on the request. Other typical situations correspond to ongoing dialogs about a previous agreement or to discussions where an agent indicates a willingness to establish an agreement, but not until a later time. An exhaustive list of the possible scenarios where a conversation between an agent ends in neither a refusal nor establishing a new agreement are given in Section 6.4.6.

Numerous events are cleansed from the execution trace because they are associated with idle time. At the same time, a large number of WAIT operators are removed to ease processing. Whenever there is a sequence of related waits (i.e., about the same coordination point), the sequence is consolidated into a single wait. Descriptions of the situations in which an agent idles is given in Section 6.3.2.

Reason for idling	Baseline Behavior	Learned Behavior
Waiting for a connection	397.8 ticks	112.9 ticks
Resource conflict	96.3 ticks	26.0 ticks
No current plan	12.2 ticks	12.8 ticks
All goals true	1.2 ticks	19.2 ticks
Extreme frustration	0.46 times	0.01 times
Delayed joint action	0.01 times	0.00 times
Ticks to solve problem	1128.8 ticks	623.5 ticks

Table 7.1: Breakdown of the different reasons agents idle

Table 7.1 gives the number of ticks the group of agents is idle during a problem, broken down by the reasons the agent was idle. The number of ticks needed to solve the problem is included for comparison. This table gives the results for the system when no learning at all occurs (Baseline Behavior) and for the tenth problem-solving episode when agents are learning both operator probabilities and coordinated procedures. Note that the statistics show number of occurrences, rather than ticks, for occasions when an agent gives up out of extreme frustration or is delayed in an

attempt at a joint action. Table 7.1 indicates that the only kind of idling event removed that accounts for more than 1.5%¹ of the (learned) community’s effort is waiting to establish a communication connection.

7.2 Segmenting the execution trace

The agent needs to reorganize the raw list of events in an cleaned execution trace to facilitate the learning steps that follow. Many of the techniques, such as identifying inefficient actions, make decisions about individual actions that can only be made correctly given a context of the goals being achieved and the other actions that help to achieve the goals. Segmenting the execution trace provides this context by producing set-of-actions/goals groupings as output.

The purpose of this process is to identify sequences of actions that are logically grouped together. Currently, two logical relations are used to identify groups: all actions that accomplish the same goal (called “goal-groupings”) and all actions that occur during a time interval (called “time-groupings”). Since the agents are goal-directed, only time-groupings that subsume goal-groupings are included. Another promising possibility is to group actions that took place in the same location.

Goal-groupings are computed first; time-groupings are derived from them. For each top-level goal that the agent achieved, the actions associated with that goal are identified primarily from the actions’ role fillers. For each action in the cleaned trace, if the operator role fillers intersect the goal literals of a top-level goal the agent achieved, then the action is added to that goal-grouping. For those actions that contain no top-level goal literals in their role fillers, the top-level goals that were being actively pursued at the time are culled from the execution trace.

¹Percentages are divided by 3 since the number of *agent · ticks* needed solve a problem

The two kinds of groupings are represented schematically in Figure 7.1. The axis represents a portion of an agent’s activity time-line with markers indicating the execution of actions A1 through A9 (more actions may precede or follow these actions). Although shown as identical single points, these actions have varying, non-unitary duration. If the agent, individually or in cooperation with other agents, accomplished goals G1 and G2 during the intervals marked on the time-line, there would be three segments identified from this activity time-line. There would be a goal-grouping segment for G1 of (A2, A3, A7, A8), a goal-grouping segment for G2 of (A4, A5, A6) and a time-grouping segment for G1 and G2 of (A2, A3, A4, A5, A6, A7, A8). This last segment is representative of the simplest class of segments that are time-groupings but not goal-groupings.



Figure 7.1: Schematic of a time-grouping segment.

Time-groupings allow the agents to learn to interleave top-level goals. They are computed after the goal-groupings have been determined. First, for each goal-grouping G , the tick of the beginning of earliest action attached to G is called $start(G)$ and the tick of the end of the latest action is called $end(G)$. Then two goal grouping $G1$ and $G2$ are combined into a time-grouping whenever $start(G1) < start(G2)$ and $end(G1) > start(G2)$. Pseudo-code for this construction process is given by function `compute-time-groupings` in Appendix C.

Attaching actions to the time-grouping is done in the exact same manner as described above for goal-groupings. For each action in the cleaned trace, the operator role fillers are tested to see if they intersect with the goal literals of the set of top-level goals of the time-grouping. For those actions that contain no top-level literals in their role fillers, the top-level goals that were being actively pursued at the time are considered instead.

This definition of time-grouping is context dependent. In other words, the groupings identified depend upon the actions that are present in the execution trace passed in as an argument. Therefore, after actions have been removed from the grouping during optimization and summarization, it is necessary to verify that the time-groupings are still valid. An example of this is:

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR>
 <UNLOAD SBOX1 HANDTR>
 <PUT-DOWN SBOX1>
 <LIFT MBOX7>
 <CARRY MBOX7 STREET>
 <LOAD MBOX7 TRUCK>
 <MOVE ROOM>
 <LIFT SBOX1>
 <CARRY SBOX1 STREET>
 <LOAD SBOX1 TRUCK>)
```

At this stage in the learning process, this looks like a time-grouping for SBOX1 and MBOX7! However, the next step, which removes inefficiencies, reduces the segment to:

```
(<LIFT MBOX7>
 <CARRY MBOX7 STREET>
 <LOAD MBOX7 TRUCK>
 <MOVE ROOM>
 <LIFT SBOX1>
 <CARRY SBOX1 STREET>
 <LOAD SBOX1 TRUCK>)
```

At this point, there is no overlapping in the goal groups, so the time-grouping is thrown away. While it is somewhat vexing to make this second check on the validity of time-groupings, it is unavoidable. The techniques that identify inefficiencies can only work correctly when given the pair (grouping-actions, grouping-goals) as the context for processing. So, segmenting before removing inefficiencies might identify time-groupings that are invalid later, but must occur at this stage of learning nonetheless.

7.3 Removing inefficiencies

This section describes the criteria that individual agents use to identify and remove inefficient actions, both primitive and communicative, from segments of traces. The criteria are simple to state: agents remove primitive actions whose effects are reversed and communicative acts associated with such primitive actions; afterwards, if several communications about a single coordination point remain, all but one are discarded. One could look at this step as either being a continuation of the cleaning process or as the commencement of the summarization process.

The techniques described in this section are extensions of the type of pruning done by standard depth-first state-space planners to prevent returning plans that include sub-sequences like ((PICK-UP BLOCK)(PUT-DOWN BLOCK)). These extensions are needed because traditional techniques do not accurately identify the relationships between actions that may exist in dynamic, multi-agent environments.

When determining if an action, primitive or communicative, can be removed, it is necessary to consider the trace segment of which the action is a part and the goals which that trace segment achieves. This context influences the pairs of primitive actions that are compared to see if they reverse each other, which predicates are relevant for determining if a pair of actions reverse each other, if a request was achieved,

and if an achieved request enabled any productive further activity. On a practical note, this means the routine to remove inefficiencies must be called for each grouping, rather than once for the entire cleaned trace.

The following subsections discuss the techniques in great detail. First, the matter of what actions should be compared is brought up. Next, the point is made that identifying reversed actions requires considering predicates that **do not** change. Third, it is argued that, although not a black and white issue, it is better to work backwards through the trace rather than forwards. After that is a description of the ways in which communicative actions are identified as inefficient. Finally, a hack that is effective in MOVERS-WORLD for generating “pretty” chunks is presented.

Throughout this section, examples will be given with just one lifter and one hand-truck operator. This is to keep the length of the examples manageable. The techniques apply equally well to scenarios when two lifters are executing joint actions. Many of the trace snippets shown will look obviously inefficient, unlike any plan returned by a decent planner. However, execution time behavior is not a direct reflection of the underlying planner. No planning strategy can prevent the agent from needing to abandon plans and switch goals due to unforeseen (or out of the agent’s control) circumstances.

7.3.1 Which primitive actions to compare

When a trace segment is examined to find pairs of actions that reverse each other, the first question that must be answered is: which pairs of actions will be compared? The most obvious way to identify reversing actions is to compare all pairs of primitive actions in the trace. This handles situations such as:

Cleaned Segment

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR2>
 <UNLOAD SBOX1 HANDTR2>
 <PUT-DOWN SBOX1>
 <LIFT SBOX1>
 <LOAD SBOX1 HANDTR2>
 <MOVE STREET>
 <UNLOAD SBOX1 HANDTR2>
 <LOAD SBOX1 TRUCK>)
```

Efficient Segment

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR2>

 <MOVE STREET>
 <UNLOAD SBOX1 HANDTR2>
 <LOAD SBOX1 TRUCK>)
```

But traces are often more complex, e.g.,

```
(<UNLOAD SBOX1 MBOX7>
 <PUT-DOWN SBOX1>
 ... [ actions not involving SBOX1 ] ...
 <LIFT SBOX1>
 <CARRY SBOX1 STREET>
 <LOAD SBOX1 TRUCK>)
```

In this case, the determination of the “sensible” output depends on whether the intervening actions are part of the final output or not. For example, if the trace is being converted to a procedure to accomplish ((ON SBOX1 TRUCK)), those actions will not be included and the desired output should be exactly like the previous example. However, if the trace is being converted to a procedure to accomplish ((ON SBOX1 TRUCK) (ON MBOX7 TRUCK)), this might not be the case. As a case in point, none of the actions in the following trace would be removed as inefficient:

```
(<UNLOAD SBOX1 MBOX7>
 <PUT-DOWN SBOX1>
 <LIFT MBOX7>
 <CARRY MBOX7 STREET>
 <LOAD MBOX7 TRUCK>
 <MOVE ROOM>
 <LIFT SBOX1>
 <CARRY SBOX1 STREET>
 <LOAD SBOX1 TRUCK>)
```

This example should make it clear why the cleaned execution trace must be segmented into groupings before identifying the reversing actions: it is only possible to accurately determine reversing actions in the context of a group of related actions. Given that context, any intervening actions can be considered relevant. This prevents removing the pair PUT-DOWN/LIFT that, on the surface, looks likely for removal. Examine the ludicrous result of ignoring intervening actions:

Cleaned Segment	Incorrect Efficient Segment
(<UNLOAD SBOX1 MBOX7> <PUT-DOWN SBOX1> <LIFT MBOX7> <CARRY MBOX7 STREET> <LOAD MBOX7 TRUCK> <MOVE ROOM> <LIFT SBOX1> <CARRY SBOX1 STREET> <LOAD SBOX1 TRUCK>)	(<UNLOAD SBOX1 MBOX7> <LIFT MBOX7> <CARRY MBOX7 STREET> <LOAD MBOX7 TRUCK> <MOVE ROOM> <CARRY SBOX1 STREET> <LOAD SBOX1 TRUCK>)

In summary, only pairs of actions that are adjacent in the segment of the cleaned trace that is associated with a set of goals should be compared to see if they reverse each other. Since removing a pair of actions changes which actions are consecutive, this criteria requires recursion or back-tracking.

7.3.2 Identifying inefficient primitive actions

The necessity of identifying pairs of reversing primitive actions within the context of a grouping is a byproduct of the fact that coordinated procedures are culled from the execution trace. If one is merely caching into memory the output of a standard planner, there is no distinction between the ordered list of actions and the “desired output.” There is another, related situation in which handling a cleaned execution

segment must be handled differently from the output of a planner: the actions associated with a grouping may be lacking some actions that logic requires. For example, consider a MOVERS-WORLD example for a segment to move a single box onto the truck involving loading the box onto, and later unloading it from, a hand-truck. The following segments only show primitive actions and have been formatted to convey the interleaving of L1's acts and HTO's acts for ease of exposition.

L1 cleaned segment

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR2>

<UNLOAD SBOX1 HANDTR2>
<LOAD SBOX1 TRUCK>)
```

HTO cleaned segment

```
(<TILT-HANDTR HANDTR2>
 <PUSH-HANDTR HANDTR2 STREET>
 <STAND-HANDTR HANDTR2>)
```

This appears odd at first glance since the LOAD and the UNLOAD are not separated by a MOVE as they would be if the entire sequence of actions were generated by a planner. The explanation for this is that L1 arrived at the street via <CARRY MBOX7 STREET>, which is not part of this segment (but is part of a segment that achieves both the small and the medium-sized box). It would be wrong to include the CARRY in a procedure to achieve ((ON SBOX1 TRUCK)) and it would be unprincipled to stick in a fictitious MOVE.

Because of situations like this, the definition of “reversing actions” needs to be carefully considered. Two actions are traditionally considered to be “reversing actions” if the add effects of one action match the delete effects of the other action and vice versa. This makes sense if the planning agent is the only one who is modifying the world. In a more general setting, however, relevant changes to the environment may have occurred that were not instigated by the planning agent. Thus the identifi-

cation of reversing actions needs to also consider some predicates whose truth values do not change as a result of the action.

For example, the location of L1 and HANDTR2 is constant for both LOAD and UNLOAD and thus does not show up on the add and delete lists. However, the fact that the actions take place in different locations is the only relevant difference between that pair of actions and a similar pair that truly did reverse each other.

-
1. Determine the relevant predicates for A_1 by considering:
 - The combined set of role fillers for A_1 and A_2 .
 - The top-level goals of the trace segment.
 - The state of the world before executing A_1 , called $BeforeState_1$.
 - The state of the world after executing A_1 , called $AfterState_1$.
 2. The predicates are partitioned by whether they are true before executing A_1 , after executing A_1 or their truth value remained unchanged by executing A_1 . These groups will be referred to as $Before_1$, $After_1$, and $Unchanged_1$.
 3. Repeat the above two steps for A_2 .
 4. A_1 and A_2 reverse each other if **all** of the following are true:
 - Every predicate in both $After_1$ and $Unchanged_1$ is true in $BeforeState_2$.
 - Every predicate in both $After_2$ and $Unchanged_2$ is true in $BeforeState_1$.
 - Every predicate in $Before_2$ is true in $AfterState_1$.
 - Every predicate in $Before_1$ is true in $AfterState_2$.

Figure 7.2: Criteria for actions A_1 and A_2 to reverse each other.

Figure 7.2 specifies the criteria for determining if a pair of actions reverse each other. Most of the figure is just defining the variables that go into the determination of whether the actions, called A_1 and A_2 in the figure, reverse each other. The translation to English is easy enough: all relevant predicates that hold after executing A_1 , including relevant predicates that did not change, must be true before executing

A_2 and all relevant predicates that hold before executing A_1 must hold after executing A_2 . And vice versa.

The key to the algorithm in Figure 7.2 is the identification of relevant predicates. An important point is that this determination is influenced by the goals the overall trace segment is achieving. Also, all definitions and criteria need to be symmetric for consistency, so that A_1 reverses A_2 if and only if A_2 reverses A_1 . This is why the combined set of role fillers for A_1 and A_2 are considered when determining the relevant predicates for each action.

A starting set of relevant literals is determined by removing locations from the incoming set of role fillers. Then all predicates in the state intersecting the relevant literals are saved. If a saved predicate contains a relevant literal that is not already in the role filler set, the literal is added and the state is checked again for additional predicates to save. Then the saved set of predicates is trimmed by excluding predicates that do not intersect the literals of the goals of this segment but do intersect other top-level goal literals. Also, a predicate is trimmed if it does not involve the agent herself but does involve other agents.

For example, in a segment to achieve ((ON SBOX TRUCK)), if the role fillers include LBOX (because, say, one of the actions was <UNLOAD SBOX LBOX>) then predicates such as (CLEAR LBOX) or (AT LBOX ROOM) are removed. If the segment were achieving ((ON SBOX TRUCK)(ON LBOX TRUCK)), these predicates would not be removed. Also, when L1 is determining whether actions reverse each other, she does not consider predicates like (HANDEEMPTY L2) or (AT HTO STREET) relevant, but she does consider (HOLDING-TOGETHER L1 L2 BOX) relevant.

7.3.3 Order of removal

Another issue at hand is whether actions should be removed from the beginning or end of the segment. For example, which of the following possibilities is more sensible? (Note that both possibilities require back-tracking or recursion in order to identify both pairs of reversing actions since only consecutive actions are compared.)

Cleaned Segment	Possibility #1	Possibility #2
(<UNLOAD SBOX1 MBOX7> <LOAD SBOX1 HANDTR> <UNLOAD SBOX1 HANDTR> <PUT-DOWN SBOX1> <LIFT SBOX1> <LOAD SBOX1 HANDTR> <MOVE STREET> <UNLOAD SBOX1 HANDTR> <LOAD SBOX1 TRUCK>)	(<UNLOAD SBOX1 MBOX7> <LOAD SBOX1 HANDTR> <MOVE STREET> <UNLOAD SBOX1 HANDTR> <LOAD SBOX1 TRUCK>)	(<UNLOAD SBOX1 MBOX7> <LOAD SBOX1 HANDTR> <MOVE STREET> <UNLOAD SBOX1 HANDTR> <LOAD SBOX1 TRUCK>)

While not vastly so, possibility 2 (removing reversing actions from the end backwards), is the superior choice. It is superior because the two actions, which jointly achieve the effect of moving SBOX1 from MBOX7 onto the hand-truck, that are kept in the segment were executed consecutively.

It is not obvious at this point in the discussion yet, but the choice will have further ramifications. The operator augmentation process is effected by the choice. The removal of primitive actions also effects which communicative actions are removed. Finally, the final stored chunk will be indexed based upon the observable features of the world in the preliminary stages of the coordinated procedure.

For instance, if SBOX1 were LIFTed instead of UNLOADed in the above cleaned segment, the storing indices of the procedure would vary based on the order of reversal. If actions are removed from the end, it would be based on the state at the time the first LIFT was executed; otherwise it would be the related to the second.

Removing actions undone by others is rarely relevant in the MOVERS-WORLD experiments since there is just one hand-truck operator (and so no others are undoing any hand-truck manipulations) and two lifters (and so joint actions manipulating boxes will be undone only by reversing pairs of actions). Furthermore, identifying such actions is difficult since identifying the key relevant changes caused by a single action can be problematic.

7.3.4 Removing communicative acts

Identifying communicative acts that are unnecessary or redundant is the last part to removing inefficiencies from a trace segment. Just as with primitive actions, the context is essential in considering whether to keep dialogs. Furthermore, determining if communication should be kept is based upon the primitive actions that remain in the segment after inefficiencies are removed. At that point, determining if a conversation about a joint action should be kept is trivial. On the other hand, requests for service require more careful analysis.

The decision whether or not a conversation is unnecessary or redundant is guided by the actions that remain in the trace after inefficiencies are removed. For example, consider a slightly more complicated variant of a preceding situation in which a PUT-DOWN has been replaced by a LOAD that requires the hand-truck operator to ready HANDTR3 for loading.

```
(<UNLOAD SBOX1 MBOX7>
<SIGN HTO (READY-TO-LOAD HANDTR3)>
<LOAD SBOX1 HANDTR3>
<LIFT MBOX7>
<CARRY MBOX7 STREET>
<LOAD MBOX7 TRUCK>
<MOVE ROOM>
<SIGN HTO (READY-TO-LOAD HANDTR3)>
<UNLOAD SBOX1 HANDTR3>
<CARRY SBOX1 STREET>
<LOAD SBOX1 TRUCK>)
```

In this example, a communicative action (i.e., a SIGN) is kept if and only if the action that followed it in the trace (e.g., LOAD or UNLOAD) is kept. In general, communication is kept only if some action it enabled is kept as well. This yields the difference between the two efficient segments involving SBOX1:

Efficient Segment for SBOX1 and MBOX7	Efficient Segment for SBOX1
(<UNLOAD SBOX1 MBOX7>	(<UNLOAD SBOX1 MBOX7>
<SIGN HTO (READY-TO-LOAD HANDTR3)>	
<LOAD SBOX1 HANDTR3>	
<LIFT MBOX7>	
<CARRY MBOX7 STREET>	
<LOAD MBOX7 TRUCK>	
<MOVE ROOM>	
<SIGN HTO (READY-TO-LOAD HANDTR3)>	
<UNLOAD SBOX1 HANDTR3>	
<CARRY SBOX1 STREET>	<CARRY SBOX1 STREET>
<LOAD SBOX1 TRUCK>)	<LOAD SBOX1 TRUCK>)

There may be redundant communicative actions in the trace segment. Although the cleaning stage does remove conversations that do not establish new agreements, it does not distinguish between multiple agreements about the same coordination point. For example, the following dialog:

```
Ticks 379 to 391: HTO and L2 converse
  "L2, I'm tired of waiting.
    Are you still working on (ON PR59-XLBOX2 TRUCK3)?"
  "HTO, I'm still working on it. Chill out!"
```

is removed by cleaning since L2 indicates that she is still working on the previous agreement. Compare this to

```
Ticks 438 to 452: HTO and L1 converse
  "L1, I'm tired of waiting.
    Are you still working on (ON PR00-XLBOX1 HANDTR3)?
  "HTO, I failed last time. I'll try again in a sec."
```

where a new agreement is clearly established so the conversation is untouched by the cleaning step. A situation that lies in between the two is:

Ticks 655 to 667: HT0 and L1 converse

"L1, I'm tired of waiting.

Are you still working on (ON PR59-XLBOX2 TRUCK3)?"

"HT0, I'm trying it again after failing last time!"

This is removed by the cleaning step because it does not establish a new agreement. No new agreement is established since L1 was already actively working on the goal. However, L1 indicated that the previous agreement is over in L1's mind (an agreement is for only a single attempt at a goal), so it is not a continuation of the previous agreement as was the case in the first of these three dialogs.

When working from a trace segment that already has inefficient actions removed, it is trivial to determine whether or not to keep a request for joint action. If a related joint action (i.e., about the same coordination point) does not come after the request in the segment, the request is unnecessary and should be removed. If there are multiple requests related to and preceding a joint action, the request that occurred last is kept and the others are removed as redundant. This reflects the close coupling that exists between requests for joint action and joint actions.

Determining whether requests for services are unnecessary or redundant is more complex. The basic idea is that the requestee must achieve the request and that the achievement of the request must enable productive future action by the requester. For example, in the following example, the load action which satisfied the agreement was removed, so the agreement itself is also removed.

Cleaned Segment	Removed Inefficiencies Segment
(<agree to (ON SBOX1 HANDTR)> <LIFT SBOX1> <LOAD SBOX1 HANDTR> <UNLOAD SBOX1 HANDTR> <CARRY SBOX1 STREET> <LOAD SBOX1 TRUCK>)	(<LIFT SBOX1> <CARRY SBOX1 STREET> <LOAD SBOX1 TRUCK>)

An example where the request did not enable productive action requires a longer trace:

Cleaned Segment	Removed Inefficiencies Segment
(<SIGN L1 (ON XLBOX2 HANDTR3)> <TILT-HANDTR HANDTR3> <PUSH-HANDTR HANDTR3 STREET1> <STAND-HANDTR HANDTR3 STREET1> cleaned: L1 refuses to unload XLBOX2 <TILT-HANDTR HANDTR3> <PUSH-HANDTR HANDTR3 ROOM1> <STAND-HANDTR HANDTR3 ROOM1> unshown: the lifters unload XLBOX2 <SIGN L1 (ON XLBOX2 HANDTR3)> <TILT-HANDTR HANDTR3> <PUSH-HANDTR HANDTR3 STREET1> <STAND-HANDTR HANDTR3 STREET1> <SIGN L2 (ON XLBOX2 TRUCK3)>)	(<SIGN L1 (ON XLBOX2 HANDTR3)> <TILT-HANDTR HANDTR3> <PUSH-HANDTR HANDTR3 STREET1> <STAND-HANDTR HANDTR3 STREET1> <SIGN L2 (ON XLBOX2 TRUCK3)>)

In this example, both requests by the hand-truck operator to get XLBOX2 onto the hand-truck were agreed-to and accomplished. Also, both operators enabled productive action (pushing the hand-truck to the street). However, when the inefficiency of pushing the hand-truck back and forth is removed, only the first request enabled productive action.

These examples make the general idea clear, but actual traces can be more complicated, so the criteria to determine if communication about a service should be removed needs to be more precise. For example, if inefficient primitive actions were removed from the beginning instead of the end (an issue discussed in Section 7.3.3), simple lookahead would incorrectly consider both requests to load the hand-truck as enabling productive actions. The technique described below would correctly identify only the second action as enabling productive action in this case. The diagram in Figure 7.3, while a bit cluttered, is an excellent tool for explaining the criteria used.

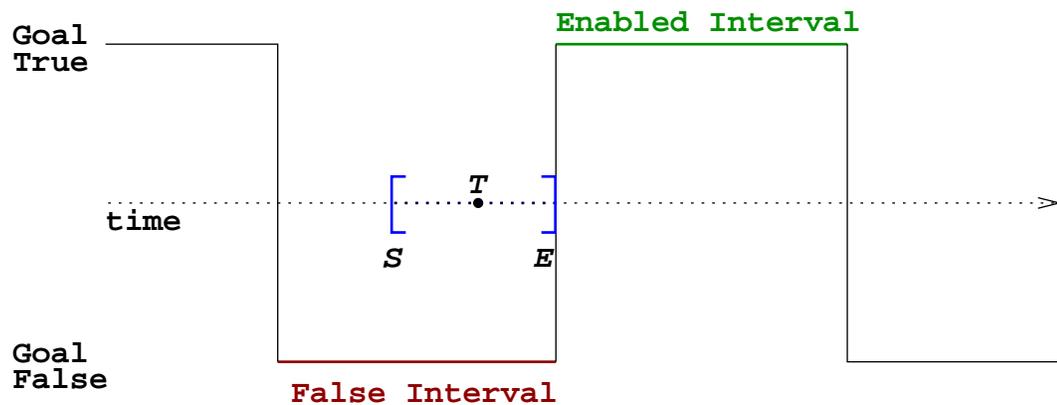


Figure 7.3: The truth values of goals helps determine if communication is kept.

The solid line in Figure 7.3 tracks the truth value of the goal of the request (i.e., the coordination point). When the solid line is above the time axis (the dotted line), the goal is true; when it is below the time axis, it is false. The far left of the time axis represents the start of this problem-solving episode and the far right represents the end of the episode. In this figure, the goal starts true and ends false; top-level MOVERS-WORLD goals start false and end true, but this is irrelevant since the point of the diagram is to clarify the time intervals of interest relative to a particular request.

The dot, labelled T , on the time axis represents a communicative act that started at time T . Note that all dialogs about a goal occur during an interval when a goal is false, such as the one labelled “False Interval.”² The closed interval $[S, E]$ surrounding T represents the interval of time that the agent was actively working on the goal. For requests, it is common for $S = T$. The relationship between the time T , the interval $[S, E]$, and the primitive actions that remain in the trace determine whether or not the communicative act will be kept or not.

There are two cases to consider depending on whether the communicative act being considered was a request by the agent or a request for the agent. If it is a request for the agent, then the criteria are:

1. There must be an interval $[S, E]$ on the agent’s list of active time-lines that is marked as ending due to completion and such that $S \leq T < E$.
2. A primitive action that ended at time E must be in the trace segment.
3. The trace segment cannot contain a related communication (i.e., about the same coordination point) that started at time T_1 , for $S \leq T_1 < T$.

If it is a request made by the agent, then E marks the time when someone else satisfied the request for the agent. In this case, the interval $[S, E]$ does not factor into the decision. Instead, the “False Interval” and the “Enabled Interval” in Figure 7.3 are relevant.

1. The trace segment cannot contain a related communication that started at time T_1 , for $S_1 \leq T_1 < T$, where S_1 is the start of the False Interval.

²Truth value in the figure is relative to the agent’s state of the world, which may be in error. Also, because actions have non-unitary durations, there are periods where a goal is “fuzzily” true. For example, if the goal is (ON BOX TRUCK), the goal is fuzzily true between the time a <LOAD BOX TRUCK> action starts and the time the action ends. So, the way to read the graph is as follows: for the periods marked as true, any action attempted by the agent during this period commenced at a time when the goal was true.

2. For top-level requests, that is all that is required.
3. For subgoal requests, there must be an action remaining in the agent's segment that started during the Enabled Interval and is related to the request. Related here is defined to mean the literals of the request intersect the literals of an add or delete effect of the action.

7.3.5 Prettying chunks

The fact that time-groupings must be validated after removing inefficiencies allows for the inclusion of a hack that creates “pretty” MOVERS-WORLD chunks. They are considered pretty since they include just the desired location-changing actions (i.e., MOVE, CARRY, CARRY-TOGETHER, PUSH-HANDTR) and no others. All actions that change the agent's location are planner-reconstructible, though, so this technique has no impact on the empirical results generated.

An example of a good opportunity for prettying is the following trace:

```
(<LIFT SBOX1>  
<LOAD SBOX1 HANDTR>  
<MOVE STREET: goal LBOX2>  
<failed to LIFT LBOX2>  
<MOVE ROOM: goal MBOX7>  
<LIFT MBOX7>  
<CARRY MBOX7 STREET>  
<LOAD MBOX7 TRUCK>  
<UNLOAD SBOX1 HANDTR>  
<LOAD SBOX1 TRUCK>)
```

Based on the techniques shown so far, two of the resulting segments would be:

Segment for SBOX1 and MBOX7

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR>
 <MOVE ROOM>
 <LIFT MBOX7>
 <CARRY MBOX7 STREET>
 <LOAD MBOX7 TRUCK>
 <UNLOAD SBOX1 HANDTR>
 <LOAD SBOX1 TRUCK>)
```

Segment for SBOX1

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR>
 <UNLOAD SBOX1 HANDTR>
 <LOAD SBOX1 TRUCK>)
```

So the chunk for SBOX1 lacks a MOVE action (which can be trivially regenerated if this plan is re-used) and the chunk for the two goals has an extraneous MOVE action (which can be trivially trimmed upon re-use). However, if all of the location changing actions were included during the step that removes inefficiencies and if a carry of an unrelated box can be considered to match a move, the result is the following “pretty” segments:

Pretty Segment for SBOX1 and MBOX7

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR>
 <LIFT MBOX7>
 <CARRY MBOX7 STREET>
 <LOAD MBOX7 TRUCK>
 <UNLOAD SBOX1 HANDTR>
 <LOAD SBOX1 TRUCK>)
```

Pretty segment for SBOX1

```
(<LIFT SBOX1>
 <LOAD SBOX1 HANDTR>
 <MOVE STREET>
 <UNLOAD SBOX1 HANDTR>
 <LOAD SBOX1 TRUCK>)
```

Working this into the learning process is quite easy because two passes through the segmentation stage are needed to verify that time-chunks are valid. A high-level description of this completely optional add-on is:

1. First pass of segmentation: include all location-changing actions.

2. Process chunks as usual except when identifying pairs of actions that reverse each other: if either is an unrelated location-changing action, do not consider predicates whose truth value did not change.
3. Second pass of segmentation: instead of only verifying, recompute the actions associated with each grouping. In this pass, only attach the same actions as would normally be attached without prettying.

This process works because the routine that identifies relevant predicates will throw away predicates about unrelated top-level goals. Thus, the predicate (HOLDING L1 BOX2) for <CARRY BOX2> is thrown away when creating a segment to achieve ((ON BOX1 TRUCK)). However, <MOVE STREET> for lifter L1 will always have an unchanged predicate of (HANDEMPY L1), regardless of the goals of the segment. So, this step is a hack that requires ignoring predicates whose truth value did not change — an incorrect policy in general but that is acceptable for actions whose desired effect is to change the agent’s location.

7.4 Optimizing coordinated procedures

An execution trace segment is considered a “complete” coordinated procedure after the inefficiencies have been removed. The procedure is complete in comparison with the stored procedure, which will have actions removed during optimization and summarization. This section describes two optimizations that allow agents to improve upon, rather than just repeat, the way in which some coordination points are jointly achieved.

The experience of past activities can lead to improved coordination and reduced communication when agents convert explicit expectations to implicit ones in their

learned procedures. For example, over time, a lifter can learn to load the hand-truck without being explicitly told to do so and the hand-truck operator can learn when to expect the lifter to load the hand-truck without being asked to do so.

The optimizations relate only to requests for service and not to requests for joint action. There are two types of coordination mechanisms contained in complete coordinated procedures that relate to remembered requests for service. The agent that initiated the request for service (the requester) will remember a SIGN action that will prompt the requester to initiate the same request again in the future. The agent that satisfied the request (the requestee) will remember the agreement that was made in the form of a WAIT-FOR-REQUEST action and will idle in the future while waiting for the request to come. In this sense, a WAIT-FOR-REQUEST is just like a WAIT. However, if the requestee becomes frustrated and starts a dialog, the conversations will proceed differently (see Section 6.4.1).

The optimizations to these coordination mechanisms are straight-forward: SIGNs are changed into WAIT-IMPLICITs, reflecting an (optimistic) expectation that the request for service will be satisfied without a direct request and WAIT-FOR-REQUESTs are dropped, reflecting an (optimistic) expectation that the agent knows the right time to accomplish the request without being specifically asked. As with WAIT-FOR-REQUEST, the WAIT-IMPLICIT is functionally equivalent to WAIT, but there are semantic differences which are relevant during communication. Table 7.2 gives a taxonomy of wait operators that MOVERS-WORLD agents may have in their plans.

Told to Wait	Learn to Wait
WAIT	WAIT-FOR-REQUEST WAIT-IMPLICIT

Table 7.2: A simple taxonomy of wait operators.

7.5 Summarizing coordinated procedures

Storing a complete coordinated procedure (or an optimized one) is possible, but it is advantageous to summarize it first. It is essential that the summarization process not remove any actions that cannot be repeated; therefore summarization removes precisely those actions which are planner-reconstructible.

Two important consequences of removing reconstructible actions are to improve plan quality and reduce plan-merging effort at communication time. Plan quality is improved since the particular time at which subordinate goals were achieved at runtime may be misleading. For example, if a lifter picked up a box before being asked to load it onto the hand-truck, it would be better if the stored plan is not tied to this sequencing. Plan-merging effort can be considerable for autonomous heterogeneous agents (Foulser *et al.*, 1992). Reducing the level of detail in stored plans reduces this complexity. For example, the lifter's reasoning is simplified when she receives a request to load the hand-truck before having lifted the box if her plan does not specify when the lift should occur. Another benefit of removing planner-reconstructible actions is that it makes the stored plan more easily adapted in the future (at the cost of regenerating the original action if it is needed again).

7.6 Preparing summarized procedures

The low-level representation for the actions in the summarized trace segment is not suitable for re-use because it is tied to the specifics of the current problem. In order to prepare the coordinated procedure for re-use, it is necessary to create new representations that can be adapted to future problem situations. An example of the kind of representational change needed is to replace goal literals with variables.

More substantially, as a consequence of removing actions from the complete procedure (either by optimization or summarization), the preparation step involves potentially augmenting action representations, in the form of new roles and preconditions.

Augmenting action representations is guided by action annotations, so the annotation process will be described first. An annotation is a recursive structure that records the manner in which each literal in the action representation reached its final ground value (cf. Kambhampati and Hendler, 1992). A simplified version of the annotation algorithm is given in Figure 7.4; this figure gives the basic idea of how actions and preconditions are marked when planning from scratch for a new goal. Unlike MACROPS (Fikes *et al.*, 1972), the augmented action is not added to the set of operators available during first-principles planning.

-
1. Annotate literals in goal G as top-level literals.
 2. For each action A in reverse (P)
 - U = Template for A after unification with its intended effect
 - F = Final instantiated template for A , after local role-binding search
 - Let $reasons = \text{nil}$
 - For each role filler v in U
 - (a) If v is ground in U , add inherited reason for v to $reasons$
 - (b) Else
 - Let $g = \text{final ground value in } F$
 - Let $role = v$'s role in action A
 - Let $precs = \text{all preconditions in } U \text{ that include } v$
 - Add make-binding-reason ($v, g, role, precs$) to $reasons$
 - Annotate A using the information in $reasons$
 - For each precondition object of A , annotate object with corresponding portion of A 's annotation

Figure 7.4: Annotating actions and preconditions for plan P that achieves goals G .

As shown in Figure 7.4, the first step is to annotate the literals of the incoming goal(s) G since they will intersect post-conditions of the last action in the plan P that achieves G . These literals are marked as “top-level” literals. Thereafter, the variables that make up an action description are marked depending on how they reached their final ground value. (Initially, they are all marked as unknown.) If the final ground value was determined by unification with the goal it achieves, the marking is inherited from the goal. Otherwise, the variable reaches its ground value by the local role-binding search process and is marked as a “binding reason”; the marking includes all the partially instantiated preconditions involving the variable that lead to this final ground value. Finally, the annotation structure for an object is built up from the reasons for each of the literals in the object.

This technique allows for the same literal to show up in the annotation structure several times with different reasons. Annotation structures often have nested variable references. In such cases, the full binding reason for each variable in an annotation structure only appears once; the other times the variable is listed with an unknown reason.

For example, Figure 7.5 shows a sample annotation structure generated by lifter L1 to accomplish ((ON PR09-MBOX8 TRUCK3)). In this example, the action described is the last operator in the plan, namely a LOAD action. The structures are messy, so the example has been extensively edited and does not include the full templates or annotation structures.

Inheritance works because a precondition object is identical to the object used by an action that accomplishes the precondition. For example, the operator for the action preceding the LOAD in L1’s plan will be <CARRY L1 PR09-MBOX8 PR09-STREET1> and the CARRY will inherit the binding reason for ?2 given in Figure 7.5.

```

Planning goal ((ON (PR09-MBOX8 . :TOPLEVEL) (TRUCK3 . :TOPLEVEL)))

Operator      (LOAD L1 PR09-MBOX8 TRUCK3)
U roles       ((:OBJECT PR09-MBOX8) (:LOCATION ?2) (:ITEM TRUCK3))
U preconditions ((AT TRUCK3 ?2) (HOLDING L1 PR09-MBOX8) (AT L1 ?2))
F roles       ((:OBJECT PR09-MBOX8) (:LOCATION PR09-STREET1)
              (:ITEM TRUCK3))
F preconditions ((AT TRUCK3 PR09-STREET1) (HOLDING L1 PR09-MBOX8)
              (AT L1 PR09-STREET1))

?2 reason     (PR09-STREET1 :BIND ?2 :LOCATION
              ((AT (TRUCK3 . :TOPLEVEL) (?2 . :UNKNOWN))
               (AT (L1 . :TOPLEVEL) (?2 . :UNKNOWN))))

Sample precondition (AT (L1 . :TOPLEVEL)
                       (PR09-STREET1 :BIND ?2 :LOCATION
                       ((AT (TRUCK3 . :TOPLEVEL) (?2 . :UNKNOWN))
                        (AT (L1 . :TOPLEVEL) (?2 . :UNKNOWN)))))
annotation

```

Figure 7.5: Sample annotation structure for a LOAD action.

A template is a structure that includes the operator name, the roles for the action, the preconditions, and all effects. U refers to the partially instantiated template for the action after unifying with the goal of (ON PR09-MBOX8 TRUCK3). F refers to the final, ground template after the local role-binding search.

Given this background on annotation structures, the augmentation of actions can be more easily explained. First, find all of the variables in all actions' annotation structures that are not a local variable for one of the actions. The actions being considered are those in the coordinated procedure that will be stored, not the complete coordinated procedures. The variables found will be local variables for actions removed during optimization and summarization.

Next, during the re-binding process where literals are converted to variables, the roles and preconditions of an action are augmented to include this additional data.

The action to be augmented is the action that first needs to determine a value for the variable during a role-binding search. For backward-chaining planners, this is often **not** the first action listed in the stored procedure that refers to the variable. For example, if both LOAD and CARRY inherited the reason for ?2, LOAD would be augmented since its local variables are determined first.

7.7 Storing procedures in memory

The final step of learning coordinated behaviors from prior activity is to store prepared execution trace segments in memory. A prepared segment and the top-level goals it achieves form a potentially new addition to the case-base. This procedure is compared to other entries in the procedural knowledge case-base. The segment is added if there is no matching entry or the matching entry cannot be generalized.

Indexing at both storage and retrieval time is based on the top-level goals being achieved and observable characteristics in the setting. At storage time, individual agents may have derived compatible procedures, but have incompatible views of the setting because there is no guarantee that the different agents will believe the procedures commenced at the same time or, even if they did, that they agree what the setting was at that time. Likewise, since remembering is dependent on the agent's assessment of the situation, agents can retrieve incompatible plans. Communication provides an opportunity to discover incompatibilities and possibly retrieve a more compatible plan.

The coordination points contained in a complete coordinated procedure (i.e., before optimizing and summarizing) provide important information about future requests that the agent is likely to receive. It is important to index the stored procedure so that it can be retrieved when the agent hears a similar request in the future.

Problem-Solving Episode	Optimized			Not Optimized		
	% of Plans from CP	Number of Entries	Unique Plans	% of Plans from CP	Number of Entries	Unique Plans
1	0.0	7.2	3.1	0.0	4.9	3.1
2	6.2	19.2	6.2	9.0	14.0	6.3
3	11.5	31.3	8.4	14.7	24.4	9.2
4	14.9	43.0	10.0	18.6	35.8	11.6
5	16.7	53.7	10.9	19.4	46.0	13.2
6	18.3	64.3	11.6	19.8	56.5	14.5
7	19.7	74.9	12.1	21.3	67.6	15.6
8	19.6	85.0	12.6	20.5	78.8	16.4
9	20.4	94.5	12.9	20.3	90.0	17.1
10	20.8	104.4	13.2	21.3	100.8	17.7

Table 7.3: Coordinated procedures case-base statistics.

Communication can then provide an opportunity for agents to synchronize their plans when working from stored memories.

In the experiments conducted so far, the number of entries in the case-base, shown in Table 7.3, was never large enough to make storage or retrieval slow. In general, though, further refinement such as distinguishing between short and long term memory or selectively “forgetting” past cases (Smyth and Keane, 1995) may be needed.

Table 7.3 shows the cumulative statistics for the community, measured across the individual case-bases since there is not a common memory structure that all agents have access to. It compares the results of two runs of ten problems each: “Optimized” corresponds to learning both OP and CP and using the heuristic optimizations, “Not Optimized” corresponds to learning both OP and CP without these optimizations. (These runs are from Experiment 2 in the results section; see Figure 5.13 for graphs of the number of ticks and conversations.) The columns labelled “% of Plans from CP” measures the average percentage of plans computed by the agents were derived from a coordinated procedure rather than from first-principles. The columns labelled “Number of Entries” contains the sum of the average number of entries in each agent’s

memory at the end of the given problem-solving episode. Finally, the plans contained in the entries are compared to all other plans, including those of other agents, to compute the number of “Unique Plans” in all of the case-bases.

7.7.1 Storage indices for coordinated procedures

The complete procedure (i.e., before optimizations and summarization) is not the procedure stored in memory. However, it plays a role in the determination of the storage indices for the procedure that is stored. Several storage indices are possible for each new procedure due to the disparity between the complete procedure and the stored procedure. For example, the chunk could be stored as of the moment when the first action in the stored procedure was started or as of the time the first action in the complete procedure was. Also, the coordination points from actions that are removed from the complete procedure can be used during the storing process to allow future dialogs about similar coordination points to be able to retrieve the stored procedure.

The storage indices for a new learned procedure are determined in a “roll-back” fashion. This means that the first step in the complete procedure is used to compute storage indices. Then the complete procedure is “rolled back,” by removing the first step and computing a new storage index. This process ends when the first action in the rolled back procedure is the first action in the stored procedure. Pseudo-code is given by the function `compute-storing-indices` in Appendix C. There are user variables *MaxStores* and *KeepEarliest*, that control the maximum number of storage indices to compute and, when *MaxStores* = 1, whether to keep the state corresponding to the first action in the complete procedure or the first action in the stored procedure. For the data in this dissertation, *MaxStores* is 1 and *KeepEarliest* is true.

For each of the storage indices, an entry will be created without any reference to an expected request. Such an entry is called a top-level entry because it can be used by the agent when searching for plans to achieve top-level goals, as opposed to planning to achieve subgoal requests. In addition to top-level entries, case-base entries will be formed that are indexed according to an associated expected request.

After the top-level entries are added, an additional case-base entry will be formed for each expected request that was rolled back. To be specific, the coordination point for any action that is a joint action or a WAIT-FOR-REQUEST is added to a list of expected requests. For each request on this list, an entry is created with a storage index based on the state at the time of the request. Another difference between an expected request entry and a top-level entry is that any top-level literals (literals that had a top-level reason) in the request must be added to those of the stored procedure.

Figure 7.6 shows some sample output when agent L1 is adding multiple case-base entries for the same coordinated procedure. In this example, the agent is only creating one top-level entry (*MaxStores* is 1 and *KeepEarliest* is true), but is creating three request entries. If *MaxStore* were greater than or equal to four, memory MEM102 would also be indexed as a top-level entry based on the state at ticks 330, 440, and 475.

7.7.2 Tailoring general memories

When trying to match the case-base entries against the current situation in order to retrieve an entry, there are many role binding variables that get bound to the same final value time and time again. For example, if an unknown role is a hand-truck and there is only one hand-truck present, it is easy to infer the correct filler without any role binding information. To save computational effort, these bindings can be

L1 has a storable procedure for ((ON XLBOX2 TRUCK3)):

```
Summarized: Ticks (252 276): <MOVE ROOM1>
Summarized: Ticks (330 359): <WAIT-FOR L2 (LIFT-TOGETHER L1 L2 XLBOX2)>
Optimized:  Ticks (440 474): <WAIT-FOR HTO (ON XLBOX2 HANDTR3)>
              Ticks (475 509): <LOAD-TOGETHER XLBOX2 HANDTR3>
Summarized: Ticks (567 591): <MOVE STREET1>
Summarized: Tick      629: <WAIT-IMPLICIT HTO (READY-TO-LOAD HANDTR3)>
Summarized: Ticks (664 698): <UNLOAD-TOGETHER XLBOX2 HANDTR3>
              Ticks (717 751): <LOAD-TOGETHER XLBOX2 TRUCK3>
```

L1 adding case-base MEM102

This is a top-level entry.

Top-level goals: ((ON ?L1-130 ?L1-129))

Expected request: NIL

The procedure is:

```
(<LOAD-TOGETHER ?L1-130 ?L1-128> <LOAD-TOGETHER ?L1-130 ?L1-129>)
```

The index is based on the state at tick 252

L1 adding case-base MEM103

Top-level goals: ((ON ?L1-130 ?L1-129))

Expected request: (LIFT-TOGETHER L1 ?L1-133 ?L1-130) by ?L1-133

The procedure is mapped from MEM102.

The index is based on the state at tick 330

L1 adding case-base MEM104

Top-level goals: ((ON ?L1-130 ?L1-129))

Expected request: (ON ?L1-130 ?L1-128) by ?L1-136

The procedure is mapped from MEM102.

The index is based on the state at tick 440

L1 adding case-base MEM105

Top-level goals: ((ON ?L1-130 ?L1-129))

Expected request: (LOAD-TOGETHER L1 ?L1-87 ?L1-130 ?L1-128) by ?L1-87

The procedure is mapped from MEM102.

The index is based on the state at tick 475

Figure 7.6: Multiple case-base entries for a single coordinated procedure.

computed once at the beginning of the problem. This tailors the general memories in the case-base to fit the current problem configuration and creates an active working memory that the agent can access efficiently. The agent can recompute working memory whenever new fillers become known or old fillers disappear.

This general process, designed to save some computational effort, can be expanded to increase the number of top-level entries at the agents' disposal. This can be done by converting entries that expect requests into top-level entries. Expected requests usually contain literals that have no binding information and are not literals from the top-level goal set. For example, a lifter would not have any role binding information about a hand-truck after agreeing to help the hand-truck operator. However, assuming there is just one hand-truck and the hand-truck is the only unknown role in a request entry, the entry can be retrieved even without the guidance of a request.

Whether or not a chunk can be converted from a request entry to a top-level entry does not usually matter, since a similar top-level entry already exists. However, some coordinated procedures are stored in only request entries. For example, here is a two action coordinated procedure for L2:

```
<WAIT-FOR L1 (LOAD-TOGETHER L2 L1 PR37-XLBOX2 HANDTR3)>
<WAIT-FOR L1 (LOAD-TOGETHER L2 L1 PR37-XLBOX2 TRUCK3)>
```

This technique is useful, but must be carefully controlled. If every entry that was indexed with an expected request was converted into a top-level request, the agent's working memory would be flooded with entries that have identical (or very nearly so) contents. This can be prevented by marking the case-base entries in such a fashion that at most one entry with the same coordinated procedure procedure is converted into a top-level entry.

As mentioned in the previous section, storage indices are computed by a roll-back procedure, after which multiple case-base entries are created when multiple expected

requests are rolled over. The marking of the case-base is done at the same time. All non-top-level entries will have a common value for a field called *marker*. If a top-level entry is created, *marker* is set to :NOGUESS. Otherwise, it is a unique identifier corresponding to the coordinated procedure, not the entire entries. In other words, the value of *marker* may be shared by other entries created before or after this batch, if the coordinated procedures can be mapped to one another.

The value of *marker* controls the tailoring. Only entries whose value for *marker* is not on a list of previously converted *marker* values is examined to see if they could be converted to top-level entries. (The list is initialized to contain :NOGUESS.) If an entry is converted, its value of *marker* is added to the list.

7.7.3 Retrieving coordinated procedures

Retrieval is done in a brute force manner. First, entries are retrieved from working memory whose goals are a subset of the ones being planned for. If this is a request, the entries retrieved are those that expect requests and whose stored request can be unified with the current one. Next, each entry considers the stored features of all context items compared to the current setting to compute a similarity score. For those that meet or exceed the current highest similarity (or, if none, a pre-set threshold of 0.50), the coordinated procedure stored in the entry is checked to see if it can be adapted to the current state of the world. If so, the current highest similarity is updated. Finally, the best of the best is selected from the remaining entries. There are various ways to determine the best of the best such as randomly, by the number of storages for the entry, or by the plan rating.

7.8 A note about intentionality

Actions' effects may be significant in unintended ways, therefore techniques should not tie learning to agents' run-time intentions. For example, relying on the agent history of active goals when constructing groupings can easily associate actions with the "wrong" sets of goals. At another step in the learning process, actions can seem planner-reconstructible (thereby inconsequential to explicitly keep in a procedure), yet have consequential effects. For example, lifters intending to achieve HANDEEMPTY by loading a box onto the truck are unintentionally satisfying a top-level goal.

Overriding an agent's belief about the intended effect of actions presumes that the agent is more informed about the importance of actions during learning than the agent was at run-time. This is in keeping with learning coordinated procedures at the end of activity, when the agent has the time and hindsight necessary to reflect on her run-time behavior.

Associating actions with goal-groupings on the basis of the role fillers of the action instead of the active goal history is necessary because the agent's intentions (as recorded in the goal history) can be misleading. To take an extreme example, it is possible for lifter L1 to execute the following sequence of actions with the indicated desired add effect and the active goal at the time:

Actions	Desired Add Effect	Active Goal
(<UNLOAD SBOX1 MBOX7>	(CLEAR MBOX7)	(ON MBOX7 TRUCK)
<CARRY SBOX1 STREET>	(AT L1 STREET)	(ON LBOX2 TRUCK)
<LOAD SBOX1 TRUCK>)	(HANDEEMPTY L1)	(ON MBOX7 TRUCK)

The techniques to identify planner-reconstructible actions presented in this paper do not override agent beliefs, even though they may be misguided. The next two paragraphs outline how it could be incorporated depending on whether the action in question achieved a top-level goal or not.

For an action that does not achieve a top-level goal, the determination of what effects are consequential can be done in a principled manner. The reason the technique is principled is that more information is available to the agent at this stage than was available at run-time. At run-time, the intended effects of an action can only be determined from the agent's plan at the time the action was executed. For example, if the action was (LOAD SBOX5 HANDTR3) and the rest of the plan was ((LIFT MBOX7) (CARRY MBOX7 STREET) (LOAD MBOX7 TRUCK)), then the load would be labelled with an intended effect of achieving (HANDEEMPTY). However, in retrospect, one can determine the actions that were enabled by the action in practice, rather than the theoretical expectation inferred from the plan. For example, if the lifter later unloaded SBOX5 from HANDTR3 at the street, the agent can recognize that this was enabled by the (ON SBOX5 HANDTR3) effect of the LOAD action.

Such an approach does not handle actions that achieve top-level goals since these actions do not enable other actions in the trace. Unfortunately, these actions are the ones that it is most beneficial to keep since their absence precludes adding a procedure to memory. There are two possible ways to work around this, neither wholly satisfactory. The first is to change the baseline planner so that such actions are always marked with the intended effect of achieving the top-level goal. The second is to explicitly scan for such actions in the execution trace and replace the run-time intended effects. The two approaches may seem equivalent, but they can result in different run-time behavior because agents communicate information about active goals and the intended effect of joint actions. For example, a lifter could conceivably ask another to load a large box onto the truck together in order to achieve HANDEEMPTY (as a subgoal to achieving another goal) instead of as a way of achieving the top-level goal.

Chapter 8

Conclusion

This dissertation reported a framework for studying coordination problems that arise in joint activities. Within this framework, learning techniques were shown to provide a significant benefit to autonomous agents who were designed with accurate and common knowledge of the domain. The larger benefit, however, of the framework was that learning led to high levels of performance even in the absence of such knowledge.

The following summary of the implemented architecture describes the life cycle of a joint activity. This summary shows how a group of independent, heterogeneous agents act and learn without the benefit of common knowledge.

First, the agents that make up the community are each given a set of “top-level” goals to work on. To provide interesting coordination problems, some of these top-level goals require the joint efforts of multiple agents; in that sense, the problem involves a cooperative activity. Also, there may be partial or complete overlap of the individual sets of top-level goals; however, the agents do not make any assumptions along those lines. Any overlap is only learned of through communication during the course of the activity.

After goal allocation, activity commences. Each agent will be acting asynchronously and, although the system measures everything with a common clock (measured in ticks), the agents do not assume that is the case. So the run-time coordination of agents is limited to happening "now" or "later" and never "in 87 ticks."

Agents mark (internally) some of their top-level goals as being actively pursued. Agents generally work on only one top-level goal at a time out of practicality, but will work on multiple top-level goals if a past experience involving several goals is similar to the current situation. Learning allows agents to develop compatible viewpoints on efficient goal ordering, without relying on controlled task allocations or guided by built-in common knowledge.

When an agent has active goals but no plan to achieve those goals, she constructs one. Initially, this is done via a hierarchical, best-first, probabilistic planner. The quality of the plans generated by this baseline planner improves over the course of a single activity as the agent learns more accurate probability estimates from her interaction experiences. The agent also has access to a library of coordinated procedures learned from past solution experiences. A coordinated procedure is defined as a sequence of individual actions separated by mechanisms (i.e., SIGNs and WAITs) that support the achievement of coordination points.

Learning coordinated procedures allows the agents to solve coordination problems more efficiently by recalling past, unplanned successes rather than through access to common knowledge during planning or to the internal structures of another agent during communication. An agent will not initiate dialog about upcoming coordination points until all of the preliminary individual actions have been successfully executed. The agent who will be asked to help achieve coordination points is selected randomly at first, but learned interaction experience guides later selections.

How easily a coordination point is jointly achieved depends on the particular mechanisms in both agents' plans. The key point is that it initially requires a fortuitous string of decisions to be made in compatible manners for the group to perform close to optimal. However, past solutions form the basis for agents to independently plan and act in ways that are in the best interest of both the agent and the group as a whole. Learning enables this improvement in behavior without any built-in common planning knowledge.

After all of the agents have accomplished all of their top-level goals, activity is considered over. At this time, the agents widen the scope of their domain model based on successful experiences, namely by learning coordinated procedures that are outside of the scope of the baseline planner. By contrast, the outcome of past low-level interactions, be they individual primitive actions or joint primitive actions or communicative actions, are added to operator probability trees during the activity.

The empirical results in Chapter 5 showed that learning only operator probabilities or learning only coordinated procedures provided statistically significant benefits to the agent community. The combined effect of learning coordinated procedures and operator probabilities had substantially more impact than either alone. Analysis of the results revealed that learning improved the performance of the community by improving the quality of the individual agents' goal selections, plan determinations, and cooperative decisions.

Goal selection. The quality of an agent's goal selection was primarily improved by the development of compatible viewpoints that results from using a case-base of past successes.

Plan Determination. The quality of plans that an agent determined improved over time because:

- Learning better probability estimates via operator probability trees led to first-principles plans that were more likely to succeed.
- Coordinated procedures represented ways to solve problems outside of the scope of the baseline planner.
- The coordinated procedures case-base contained past, successful ways to interleave subgoals.

Cooperation Decisions. The development of compatible viewpoints meant that agents were faced with fewer hard decisions. Agents were better at making hard decisions because operator probabilities allowed them to more accurately assess plans.

The empirical results further showed that coordinated procedures contained key pieces of domain knowledge. Furthermore, even when these key pieces were built into the agent architecture, learning coordinated procedures significantly improved performance. This was true whether or not procedures were optimized. Overall, the results indicated that agents automatically acquired essential domain knowledge with domain-independent techniques.

In conclusion, this dissertation presented a framework to support joint activity that is both robust and leads to high levels of performance by including learning as a core component of the underlying agent architecture. Robustness is achieved by not relying on common knowledge. A theoretical advantage of this assumption is that it better models agents as heterogeneous, independent experts; a practical advantage is that it places less engineering burden on implementors. The robustness and efficacy of techniques to learn coordinated procedures and operator probability trees were validated by rigorous empirical studies conducted in an implemented test-bed system.

Appendix A

Sample MOVERS-WORLD problem

The following transcript is verbatim output of the system. This trace is included for two reasons: to illustrate the level of detail that is modeled in this system and to demonstrate how the lack of common knowledge is reflected in the run-time behavior of MOVERS-WORLD agents. The problem shown was chosen as one of modest difficulty. It involves one box of three different sizes (small, large and extra-large), none of which are initially stacked on top of another box. The problem was then run 6000 times using different initial random seed values; the output shown is from a run of median length (as measured in ticks). It is “vanilla” system output in that it reflects the baseline set of parameters that control agent behavior such as the goal-selection strategy and the cooperation strategy. However, learning is restricted to very specific interaction knowledge that is necessary for the agents to explore more of the planning search space; operator probability trees are not being updated.

All conversations among the agents are shown, i.e., there is no off-line / out-of-bands discussion and the “natural” language glosses shown are for the human

reader's benefit. The agent's decisions are based upon the contents of request and response frames of various types. A gloss usually, but not always, shows all pertinent information. Section 6.4 details exactly what information is transmitted.

Running problem 12 with goals ((ON PR12-XLBOX1 TRUCK3) (ON PR12-LBOX3 TRUCK3)
(ON PR12-MBOX8 TRUCK3))

Ticks 1 to 15: HTO and L2 converse
 "L2, would you help me achieve (ON PR12-LBOX3 HANDTR3)?
 This is part of a plan involving you to achieve (ON PR12-LBOX3 TRUCK3)."
 "HTO, I'll help you ... not! Get a clue!"
 Ticks 1 to 15: <LIFT PR12-LBOX3> by L1 failed
 Ticks 16 to 30: HTO and L1 converse
 "L1, would you help me achieve (ON PR12-XLBOX1 HANDTR3)?
 This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."
 "HTO, I'll help you ... not! Get a clue!"
 Ticks 16 to 45: <LIFT PR12-MBOX8> by L2 successful
 Tick 31: HTO got no answer when calling L2. Letting ring.
 Ticks 31 to 45: <LIFT PR12-XLBOX1> by L1 failed
 Ticks 46 to 60: HTO and L2 converse
 "L2, would you help me achieve (ON PR12-MBOX8 HANDTR3)?
 This is part of a plan involving you to achieve (ON PR12-MBOX8 TRUCK3)."
 "HTO, I'll help you ... not! Get a clue!"
 Ticks 48 to 62: <LIFT PR12-XLBOX1> by L1 failed
 Tick 61: HTO got no answer when calling L1. Letting ring.
 Ticks 61 to 100: <CARRY PR12-MBOX8 PR12-STREET1> by L2 successful
 Ticks 62 to 86: <MOVE PR12-STREET1> by HTO successful
 Ticks 63 to 77: <LIFT PR12-LBOX3> by L1 failed
 Tick 78: L1 got no answer when calling HTO. Letting ring.
 Ticks 87 to 106: L1 and HTO converse
 "HTO, would you help me achieve (HOLDING-TOGETHER PR12-XLBOX1)
 via (LIFT-TOGETHER PR12-XLBOX1)?
 This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."
 "L1, Sorry Dude :(I'd like to help, but no can do."
 Ticks 101 to 135: <LOAD PR12-MBOX8 TRUCK3> by L2 successful
 Tick 107: L1 got no answer when calling L2. Letting ring.
 Ticks 107 to 131: <MOVE PR12-ROOM1> by HTO successful
 Ticks 132 to 146: HTO and L1 converse
 "L1, would you help me achieve (ON PR12-LBOX3 HANDTR3)?
 This is part of a plan involving you to achieve (ON PR12-LBOX3 TRUCK3)."
 "HTO, I'll help, but you'll have to wait a bit."
 Ticks 136 to 160: <MOVE PR12-ROOM1> by L2 successful
 Tick 147: L1 got no answer when calling L2. Letting ring.
 Ticks 161 to 180: L1 and L2 converse
 "L2, would you help me achieve (HOLDING-TOGETHER PR12-LBOX3)
 via (LIFT-TOGETHER PR12-LBOX3)?
 This is part of a plan involving you to achieve (ON PR12-LBOX3 TRUCK3)."
 "L1, sure! Lets get to it."
 Ticks 181 to 210: <LIFT-TOGETHER PR12-LBOX3> by L1 and L2 successful

Ticks 211 to 230: L2 and L1 converse

"L1, would you help me achieve (AT PR12-STREET1)
via (CARRY-TOGETHER PR12-LBOX3 PR12-STREET1)?

This is part of a plan involving you to achieve (ON PR12-LBOX3 TRUCK3)."

"L2, sure! Lets get to it."

Ticks 231 to 270: <CARRY-TOGETHER PR12-LBOX3 PR12-STREET1> by L1 and L2 successful

Ticks 271 to 288: L1 and L2 converse

"L2, would you help me achieve (ON PR12-LBOX3 TRUCK3)
via (LOAD-TOGETHER PR12-LBOX3 TRUCK3)?"

"L1, sure! Lets get to it."

Ticks 289 to 323: <LOAD-TOGETHER PR12-LBOX3 TRUCK3> by L1 and L2 successful

Tick 307: HTO got no answer when calling L1. Letting ring.

Ticks 324 to 340: HTO and L1 converse

"L1, I'm tired of waiting. Are you still working on (ON PR12-LBOX3 HANDTR3)?
This is part of a plan involving you to achieve (ON PR12-LBOX3 TRUCK3)."

"HTO, that top-level goal is true from my vantage point!"

Ticks 324 to 348: <MOVE PR12-ROOM1> by L2 successful

Ticks 341 to 365: <MOVE PR12-ROOM1> by L1 successful

Ticks 341 to 365: <MOVE PR12-STREET1> by HTO successful

Ticks 349 to 363: <LIFT PR12-XLBOX1> by L2 failed

Ticks 364 to 378: <LIFT PR12-XLBOX1> by L2 failed

Tick 366: L1 got no answer when calling L2. Letting ring.

Ticks 366 to 390: <MOVE PR12-ROOM1> by HTO successful

Ticks 379 to 398: L1 and L2 converse

"L2, would you help me achieve (HOLDING-TOGETHER PR12-XLBOX1)
via (LIFT-TOGETHER PR12-XLBOX1)?

This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L1, sure! Lets get to it."

Tick 391: HTO got no answer when calling L2. Letting ring.

Ticks 399 to 428: <LIFT-TOGETHER PR12-XLBOX1> by L1 and L2 successful

Tick 429: L1 got a busy signal when calling L2.

Ticks 429 to 443: HTO and L2 converse

"L2, would you help me achieve (ON PR12-XLBOX1 HANDTR3)?

This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"HTO, I'll help, but you'll have to wait a bit."

Ticks 444 to 463: L1 and L2 converse

"L2, would you help me achieve (AT PR12-STREET1)
via (CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1)?

This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L1, sure! Lets get to it."

Ticks 464 to 483: <CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1> by L1 and L2 failed

Ticks 484 to 503: L1 and L2 converse

"L2, would you help me achieve (AT PR12-STREET1)
via (CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1)?

This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L1, sure! Lets get to it."

Ticks 504 to 523: <CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1> by L1 and L2 failed

Ticks 524 to 543: L1 and L2 converse

"L2, would you help me achieve (AT PR12-STREET1)
via (CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1)?

This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L1, sure! Lets get to it."

Ticks 544 to 563: <CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1> by L1 and L2 failed

Ticks 564 to 583: L2 and L1 converse

"L1, would you help me achieve (AT PR12-STREET1)
via (CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1)?
This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L2, sure! Lets get to it."

Ticks 584 to 603: <CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1> by L1 and L2 failed

Tick 604: HTO got a busy signal when calling L2.

Ticks 604 to 623: L1 and L2 converse

"L2, would you help me achieve (AT PR12-STREET1)
via (CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1)?
This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L1, sure! Lets get to it."

Tick 605: HTO got no answer when calling L2. Letting ring.

Ticks 624 to 643: <CARRY-TOGETHER PR12-XLBOX1 PR12-STREET1> by L1 and L2 failed

Tick 644: L1 got a busy signal when calling L2.

Ticks 644 to 658: HTO and L2 converse

"L2, I'm tired of waiting. Are you still working on (ON PR12-XLBOX1 HANDTR3)?
This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"HTO, I got sidetracked. Just hold on a bit more."

Tick 645: L1 got no answer when calling L2. Letting ring.

Ticks 659 to 678: L2 and L1 converse

"L1, would you help me achieve (ON PR12-XLBOX1 HANDTR3)
via (LOAD-TOGETHER PR12-XLBOX1 HANDTR3)?
This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L2, sure! Lets get to it."

Ticks 679 to 713: <LOAD-TOGETHER PR12-XLBOX1 HANDTR3> by L1 and L2 successful

Tick 714: <TILT-HANDTR HANDTR3> by HTO delayed

Tick 714: <UNLOAD PR12-XLBOX1 HANDTR3> by L2 delayed

Ticks 714 to 731: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 failed

Tick 732: <TILT-HANDTR HANDTR3> by HTO delayed

Tick 732: <UNLOAD PR12-XLBOX1 HANDTR3> by L2 delayed

Ticks 732 to 749: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 failed

Tick 750: <UNLOAD PR12-XLBOX1 HANDTR3> by L2 delayed

Tick 750: <TILT-HANDTR HANDTR3> by HTO delayed

Ticks 750 to 767: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 failed

Tick 768: <UNLOAD PR12-XLBOX1 HANDTR3> by L2 delayed

Tick 768: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 delayed

Ticks 768 to 787: <TILT-HANDTR HANDTR3> by HTO successful

Tick 788: L2 got a busy signal when calling HTO.

Ticks 788 to 801: L1 and HTO converse

"HTO, would you help me achieve (READY-TO-LOAD HANDTR3)?
This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."

"L1, I'm already working on it!"

Tick 802: L2 got no answer when calling HTO. Letting ring.

Ticks 802 to 831: <PUSH-HANDTR HANDTR3 PR12-STREET1> by HTO successful

Ticks 803 to 827: <MOVE PR12-STREET1> by L1 successful

Tick 828: L1 got no answer when calling HTO. Letting ring.

Ticks 832 to 856: <MOVE PR12-STREET1> by L2 successful

Ticks 832 to 851: <STAND-HANDTR HANDTR3 PR12-STREET1> by HTO successful

```

Tick 852: HTO got no answer when calling L2. Letting ring.
Ticks 852 to 869: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 failed
Ticks 857 to 873: HTO and L2 converse
    "L2, would you help me achieve (ON PR12-XLBOX1 TRUCK3)?"
    "HTO, I've lost track of those items."
    "L2, I can't help you with that."
    "HTO, Sorry Dude :( I'd like to help, but no can do."
Ticks 870 to 887: <UNLOAD PR12-XLBOX1 HANDTR3> by L1 failed
Tick 874: HTO got no answer when calling L1. Letting ring.
Tick 888: HTO got a busy signal when calling L1.
Ticks 888 to 908: L1 and L2 converse
    "L2, would you help me achieve (HOLDING-TOGETHER PR12-XLBOX1)
    via (UNLOAD-TOGETHER PR12-XLBOX1 HANDTR3)?
    This is part of a plan involving you to achieve (ON PR12-XLBOX1 TRUCK3)."
    "L1, sure! Lets get to it."
Tick 909: HTO got no answer when calling L1. Letting ring.
Ticks 909 to 943: <UNLOAD-TOGETHER PR12-XLBOX1 HANDTR3> by L1 and L2 successful
Tick 944: HTO got a busy signal when calling L1.
Ticks 944 to 961: L2 and L1 converse
    "L1, would you help me achieve (ON PR12-XLBOX1 TRUCK3)
    via (LOAD-TOGETHER PR12-XLBOX1 TRUCK3)?"
    "L2, sure! Lets get to it."
Tick 962: HTO got no answer when calling L1. Letting ring.
Ticks 962 to 996: <LOAD-TOGETHER PR12-XLBOX1 TRUCK3> by L1 and L2 successful

```

Figure A.1 shows the overall distribution of simulated execution time of the 6000 runs overlaid on the probability density function for a fitted lognormal curve. The data are in a histogram with a bucket size of 33 ticks. The distribution is clearly not a normal curve with the same mean and variance as the sample data. The floor effect that is present in MOVERS-WORLD problems is shown by the skewed, tail-heavy shape. The histogram is roughly approximated by a lognormal fit (changing the bucket size does not yield closer fits). Since the lognormal distribution is not symmetric about the mean, the data were shifted, in this case by 500 ticks, in order to compute a reasonable match.

The lifters have a great amount of difficulty achieving the level of coordination needed to effectively use the hand-truck unless they are learning coordinated procedures from past execution traces. The table below gives a statistical summary for the distribution of the number of ticks required to solve the problem split by the number of times the community use the hand-truck to move boxes to the street.

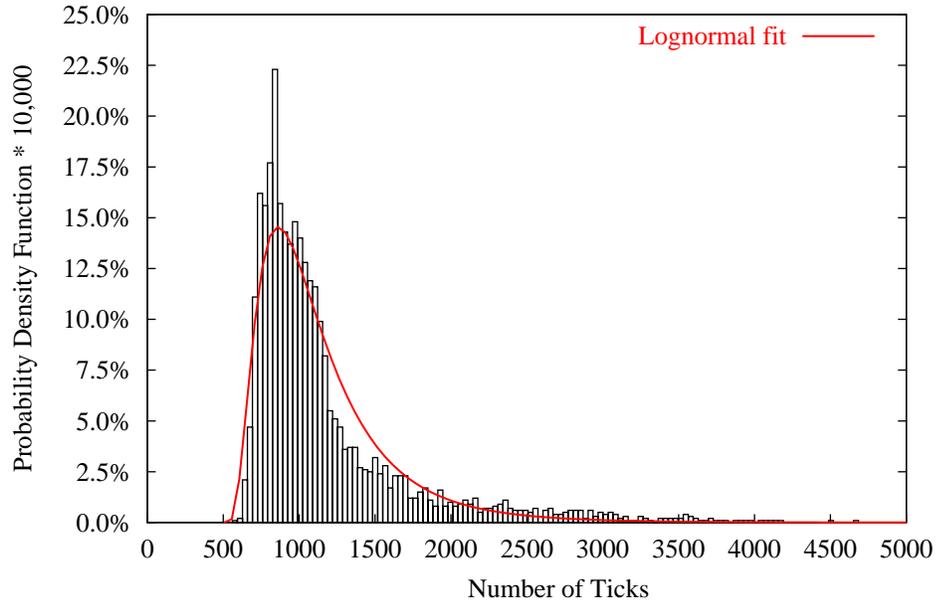


Figure A.1: Distribution of simulated execution time.

	Overall	Number of Times the Hand-truck is Used	
		Once	Twice
Data Points	6000	4919	1081
Average	1167.8	970.0	2067.8
Sample Deviation	536.8	208.8	642.6
Range	575 – 4672	575 – 2401	1053 – 4672
5% - 95% Range	717 – 2365	709 – 1370	1289 – 3394
Median	1014	925	1918
Mode	841 (23 times)	841 (23 times)	11 values (4 times)

Table A.1: Splitting the data on the number of times the hand-truck is used.

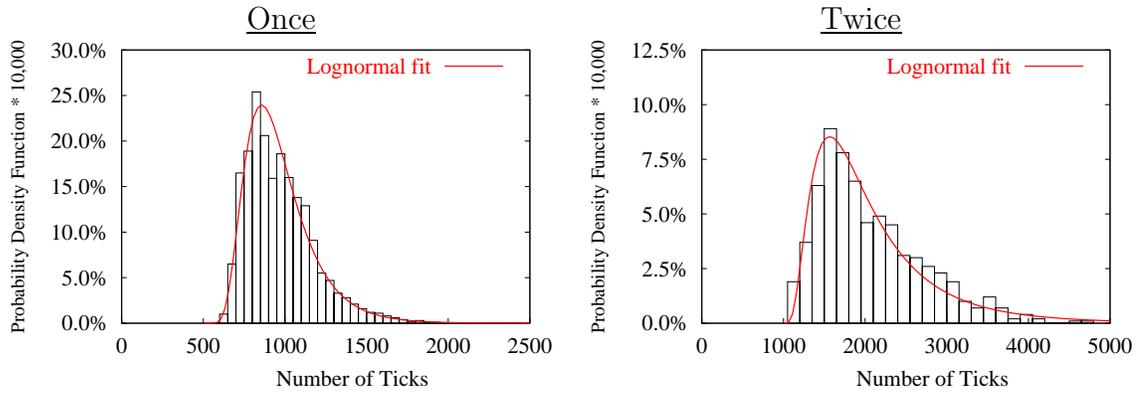


Figure A.2: Distributions when the data are split.

To complement these estimates of population parameters, the underlying population data is examined again. Figure A.2 shows that the lognormal fits are excellent matches. The left-hand side of the figure shows that the distribution of ticks required to solve the problem, in buckets of size 50, when the hand-truck is only used once. The distribution when the hand-truck is used twice is shown on the right-hand side. For this graph, the bucket size was 150 ticks and the data were shifted by 1000 to compute the lognormal fit.

Appendix B

Other empirical investigations

B.1 Cooperation strategies

For the results presented in the main text, the agents' decision making strategy is based purely on rationality. In this section, that strategy will be abbreviated RAT and the results generated by RAT will be compared to those generated by seven other strategies. These decision strategies are only relevant for the "difficult" decisions talked about in the results section. Namely, if the agent is idle or incapable of helping or the request is directly related to the agent's current plan, the response will not be guided by the strategy. For example, the data will show that an agent who is randomly deciding whether to cooperate or not will, in fact, agree to help around 75% of the time.

Before moving on to the comparisons with other techniques, a short description of how RAT is implemented is in order. This strategy gives preference to the plan that achieves goals in the most ticks-efficient manner. This is simply another way of saying that the plans are evaluated by the same heuristic that the baseline planner uses (expected reward per tick). The one major difference is that agents need to be

able to evaluate plans that do not satisfy any top-level goals, throwing the normal computation of expected reward off. To handle this situation, plans are evaluated as if they did achieve one top-level goal, but the reward for this achievement is delayed beyond the end of the plan. A number of domain- and plan-independent metrics were tested, but the metrics that lead to the best performance did so because they were well suited to MOVERS-WORLD. Therefore, to be up front about this fact, the metric selected (`MW-domain-distance`, for which pseudo-code is given in Appendix C) was one that was clearly domain-specific but not highly crafted.

The seven other cooperation strategies investigated besides RAT are:

YES Agent always agrees to help.

RAN Agent decides whether to agree randomly, with a 50% chance of agreeing to help.

HYB Agent decides whether to agree randomly, with the chance of agreeing to help determined by the relative plan evaluations.

SEN Agent agrees if the requester has more seniority and refuses if the requester has less seniority. If the seniorities are the same, there is a 50% chance of agreeing to help.

SRH Agent decides according to HYB, but factors in the seniority of the requester. The evaluation of the agent's current plan is increased if she is more senior than the requester and the evaluation of the plan that accomplishes the request is increased if she is less senior. Agents will not opt out of agreements with more senior agents regardless of plan ratings.

SRR Agent decides according to RAT, but factors in the seniority of the requester. The evaluation of the agent's current plan is increased if she is more senior than

the requester and the evaluation of the plan that accomplishes the request is increased if she is less senior. Agents will not opt out of agreements with more senior agents regardless of plan ratings.

FID Agent decides according to RAT, but values fidelity. This means that the agent will neither agree to switch goals during communication or even adapt her current plan during communication if it involves opting-out of a previous agreement.

Most of the routines are self-descriptive; for clarity, Appendix C contains pseudo-code for some of them. When seniority is a factor to be considered, the seniorities were set to make the hand-truck operator the *de facto* supervisor (HTO seniority 3, L1 seniority 1, L2 seniority 2). This arrangement has pros and cons. If HTO is not senior to at least one of the lifters, an extra-large box will only get loaded onto a hand-truck if HTO asks a lifter for help at a moment when the lifter has no plan. On the other hand, for small- and medium-sized boxes, lifters will agree to load the hand-truck whether or not it is sensible.

Tests were run to measure impact of the cooperation strategies on the performance of the community and, in particular, on the effectiveness of the learning techniques. For these runs of the system, the agents solved sequences of 5 problem-solving episodes when they were learning both operator probabilities and coordinated procedures. Tables B.1, B.2, and Figure B.1 show that the cooperation strategy can make an enormous difference in initial performance but that learning is effective in all cases. Further, learning greatly reduces the disparity between the strategies as agents are faced with fewer difficult decisions.

A comparison of some performance measures for MOVERS-WORLD is given in Table B.1 for the first problem in a sequence. This shows that rationality-based

	Ticks	Actions		Requests	Agreements	Agreement Percent
		Attempted	Successful			
FID	1011.6	40.5	31.6	21.9	14.8	66.4%
RAT	1038.1	41.1	30.9	22.6	15.8	68.7%
SRR	1055.6	40.0	29.3	20.2	17.6	87.0%
HYB	1096.4	42.0	30.9	23.2	17.1	74.1%
SRH	1139.4	42.0	30.3	22.2	19.4	86.8%
YES	1205.1	43.7	31.8	23.6	21.9	92.0%
RAN	1403.9	50.4	34.4	30.5	22.4	73.6%
SEN	1685.2	62.2	34.9	34.3	28.3	82.7%

Table B.1: Performance in the first episode for possible cooperation strategies.

strategies are much more effective in the short run, followed by HYB. RAN and SEN are very bad in comparison to the others. Table B.2 shows the same statistics for the fifth problem-solving episode. Improvement was dramatic across the board. YES is now the worst strategy and even that strategy had almost a 33% improvement in performance.

	Ticks	Actions		Requests	Agreements	Agreement Percent
		Attempted	Successful			
RAT	675.5	25.4	24.1	11.5	9.0	81.1%
FID	701.3	26.1	24.8	12.1	9.4	80.5%
HYB	739.0	26.9	25.0	12.7	10.3	83.3%
SRR	740.2	26.3	24.9	12.0	10.8	90.8%
SRH	750.0	26.4	25.0	12.2	11.1	91.1%
RAN	814.8	29.0	26.0	14.2	11.6	84.7%
SEN	818.7	28.1	24.0	13.6	11.8	88.1%
YES	829.1	28.7	27.2	13.6	13.2	96.9%

Table B.2: Performance in the fifth episode for possible cooperation strategies.

Figure B.1 contains a learning curve (tracking the number of ticks) for six of the strategies. (FID and SRH are not shown since they so closely track with RAT and SRR, respectively.) From this it is clear that learning makes the initial selection for the underlying cooperative strategy less important than might be expected.

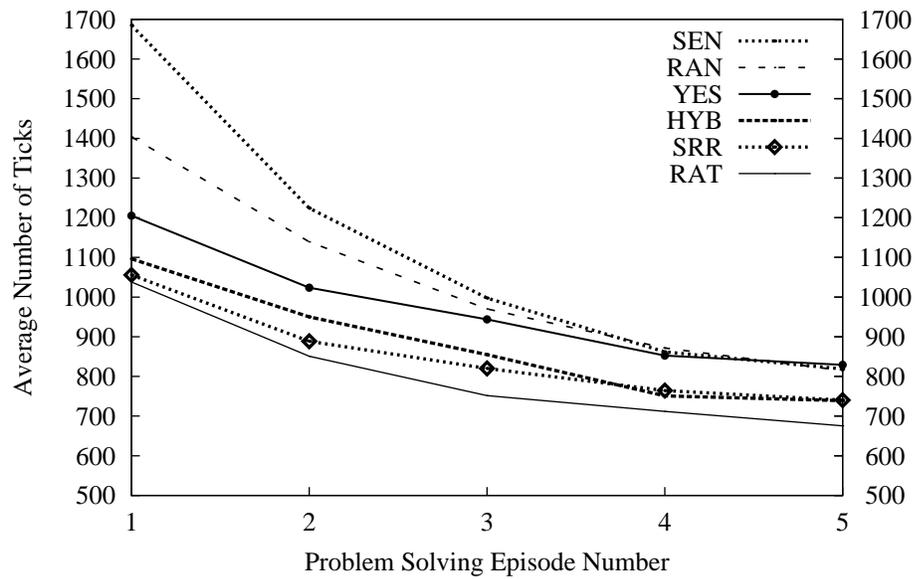


Figure B.1: Overall effort for various cooperation strategies.

B.2 Synchronous activity

In previous research (Garland and Alterman, 1998, 1996, 1995), actions and conversations all lasted for the same length of time (a “round”), which corresponds roughly to *ATP* ticks. This course-grained view of allotting the same amount of time to all actions, whether joint or individual, whether successful or not, and whether primitive or communicative, is a reasonable starting point since they are likely to be the same order of magnitude when compared to planning costs. By comparison, the average attempted action lasts 27.1 ticks and the average conversation lasts 19.2 ticks for the first problem in a sequence for the results in the main body of the dissertation.

The asynchronous flow of an activity was presented in Section 3.2. There are four stages to the process. Initially, the agent community is given a set of goals to achieve. Then, if the agents are working from a case-base of previous activity, general memories are tailored to fit the current setting. The last stage is when new coordinated procedures are added to memory. Those stages remain the same. However, the description of the activity itself, stage 3, is somewhat simplified in the synchronous view:

3. Activity *per se*:

Commences when any agent interacts with the world.

Loops through the following:

- (a) Active agents create or adapt a plan as needed.
- (b) Agents with plans attempt a single action, including communication.

Terminates when all agents believe the cooperative goals have been achieved.

	Statistic	Synchronous		Asynchronous	
Events	Time Steps	24.2 rounds		623.5 ticks	
	Total actions duration	31.5 rounds	43.4%	940.0 ticks	50.3%
	Total conversations duration	21.8 rounds	30.0%	405.4 ticks	21.7%
	Waiting for a request	13.8 rounds	19.0%	370.3 ticks	19.8%
	Idle	5.5 rounds	7.6%	151.7 ticks	8.1%
Acting	Attempted actions	25.0		24.2	
	Successful actions	24.5		23.6	
	Wanderings	1.4		1.2	
	Attempted joint actions	6.5	26.0%	6.4	26.4%
	Successful joint actions	6.4		6.3	
Talking	Conversations	10.9		10.0	
	Agreements	8.3	76.1%	7.7	77.0%
	Agreements to change goals	0.9		0.7	
	Refusals	3.0		2.9	
Planning	Calls to the planner	39.2		34.9	
	Planning nodes expanded	1291	32.9	1060	30.4
	Role-binding nodes expanded	1904		1858	
	No plan generated	1.0		1.9	
	Current plan adapted	7.6		7.1	

Table B.3: Synchronous compared to asynchronous for the tenth episode.

Table B.3 compares the performance of the community for the tenth problem-solving episode under the two paradigms. These statistics suggest that the underlying dynamics are not effected dramatically by adopting the simpler, synchronous viewpoint. A lower percentage of effort is spent acting (43.4% compared to 50.3%) and a higher percentage is spent conversing (30.0% versus 21.7%). This is not surprising since, on average, actions lasted 41.1% longer than conversations in the asynchronous runs.

To complete the comparison, the left hand sides of the graphs below show learning curves for the community when they are acting synchronously. The right hand sides show the same learning curves when acting asynchronously. The results are extremely similar, showing that the techniques presented are effective for either model.

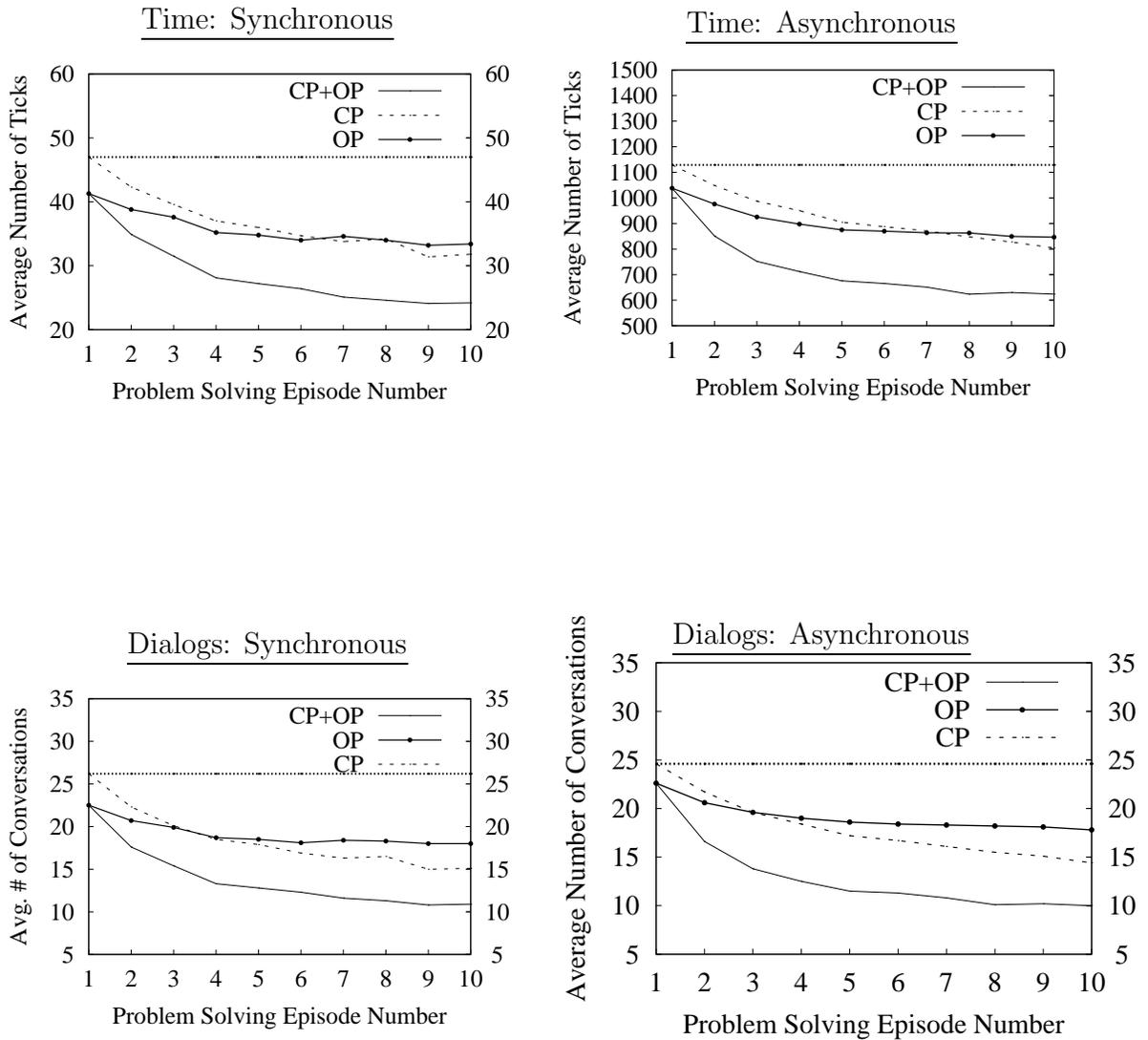


Figure B.2: Synchronous versus asynchronous learning curves.

B.3 Varying action and communication durations

Despite the fact that the results presented in the last section were similar in either a synchronous or asynchronous model, the flexibility of the asynchronous model offers an advantage. In an asynchronous model, varying assumptions about communication can be incorporated easily by changing the parameters *CommRatio* and *CommOverheadRatio*. This allows modeling many domains whose communication costs (relative to primitive actions) vary widely. For instance, costs could be set low to model agents that are hard-wired together over a fast network or very high to model agents communicating over a cellular link. If all interactions take a single round, a *post hoc* adjustment to reflect changing communication costs cannot capture the execution-time dynamics that would have occurred had conversations taken more (or less) time than other actions.

Action durations could also be varied *en masse* by adjusting *ATP*. Currently, however, changing *ATP* is not meaningful. *ATP* is meant to represent how long a typical action lasts compared to a typical planning session (cf. the DARPA/Rome planning initiative (Burstein, 1994)). But in the current implementation, agents are not delayed by their planning sessions¹, so the principle effect of changing *ATP* is to change the relative amount of time spent acting versus communicating. This can be studied more directly by changing *CommRatio*.

To get a feel for how changing *CommRatio* from its default value of 1 effects the performance of the community, MOVERS-WORLD experiments were run with three other values: 0, 0.5 and 2. When *CommRatio* is 0, *CommOverheadRatio* is also set to 0; for the other data *CommOverheadRatio* is 0.125, which equates to 5 ticks of overhead when *ATP* is 40. In these experiments, agents were learning both

¹The issue is not ignored. Planning occurs incrementally, generally for one goal at a time, without communication in order to keep planning times reasonable.

operator probabilities and coordinated procedures; the length of test sequences was five problem-solving episodes.

	Value of <i>CommRatio</i>			
	0.0	0.5	1.0	2.0
Average conversation duration	1.0	12.4	19.2	33.0
Estimated Ticks	761.1	892.0		1330.2
Ticks	760.3	926.7	1038.1	1237.0
Conversations	27.5	23.6	22.6	21.3
Attempted actions	43.8	41.4	41.1	39.5

Table B.4: Performance for the first episode when varying communication costs.

A comparison of some performance measures for MOVERS-WORLD is given in Table B.4 for the first problem in a sequence. As the value of *CommRatio* increases, the average conversation length increases as expected (recall there is a 5 tick overhead when *CommRatio* > 0). An estimated number of ticks for *CommRatio* \neq 1 can be computed from these durations, the average number of conversations for *CommRatio* = 1, and the number of ticks for *CommRatio* = 1. These estimates are not that far off, suggesting that perhaps the run-time dynamics are not influenced much by the changes to *CommRatio*. However, this is not true — agents compensate for longer dialogs by conversing less frequently and attempting fewer actions.

	Value of <i>CommRatio</i>			
	0.0	0.5	1.0	2.0
Average conversation duration	1.0	12.9	20.1	35.0
Estimated ticks	530.0	598.9		828.6
Ticks	501.2	632.7	675.5	844.9
Conversations	12.3	12.4	11.5	11.8
Attempted actions	26.2	26.5	25.4	25.8

Table B.5: Performance for the fifth episode when varying communication costs.

Table B.5 shows the same statistics for the fifth problem-solving episode. Conversations now contain around 7.5 symbols per conversation, up from around 7.0. The estimated number of ticks is more accurate on average because the number of conversations and attempted actions are nearly constant across all values of *CommRatio*.

Figure B.3 contains a learning curve (tracking the number of ticks) for the different values of *CommRatio*. As with the other studies so far, learning is effective in all cases. Unlike the other studies, however, the performance of the community does not converge for the different assumptions. This is due to the fact there is a lower limit to the number of conversations and actions need to solve MOVERS-WORLD problems, regardless of *CommRatio*.

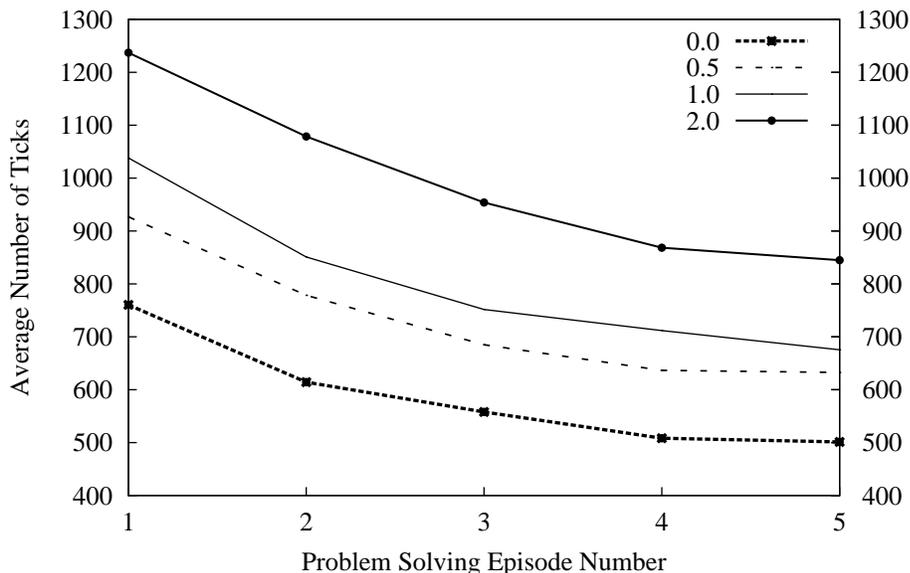


Figure B.3: Overall effort when varying communication costs.

B.4 Goal selection strategies

A goal-selection strategy is implemented as a list of functions that are called, in order, to determine which goal(s) the agent will actively pursue next. Each function prunes an incoming list of candidate goals according to a different criteria. The list of candidate goals (candidates, for short) is initially the set of unmet top-level goals. If more than candidate remains after all functions have been called, Pick-Random is called to select (with uniform probability) one of the candidate goals. The results presented in the main test were based upon a common goal-selection strategy (but this fact is not known by the agents). Since the agents have different experiences and make different random selections, the order in which top-level goals are selected to work on will vary.

The default decision strategy in MOVERS-WORLD is (Drop-Failed Search-CB). Drop-Failed prunes the list of candidate goals to remove those that the agent has failed at before; if they have all failed at some point, no pruning is done. Search-CB checks the case-base of coordinated procedures for a plan that accomplishes a subset of the candidate goals. If such a plan is found, the agent labels the corresponding subset of goals active.

The list below gives a sampling of other functions that could appear in a goal-selection strategy. The function `goal-select` in Appendix C gives pseudo-code for the overall goal-selection process.

Drop-Failed Remove goals from the candidates if any attempt at the goal ended in failure. Return the result or, if none, all candidates.

Search-CB Search the case-base for a plan that achieves a subset of the candidates. If found, return the goals (and plan) and terminate the goal-selection process; else return all candidates and continue.

Pick-Random Randomly select one goal from the candidates.

Drop-Attempted Remove goals previously attempted by the agent from the candidates. Return the result or, if none, all candidates.

Keep-Dropped Keep goals from the candidates if the last attempt at the goal by the agent ended in the goal being dropped. Return the result or, if none, all candidates.

Keep-Failed Keep goals from the candidates if the last attempt at the goal ended in a failure by the agent. Return the result or, if none, all candidates.

Order-Failed Order goals by the number of failed attempts by the agent. Keep only goals with the least number of failed attempts. Return the result.

Keep-XLBOX This is the domain-specific strategy used in the experiments of Section 5.4. It presumes agents are able to observe the size of the boxes, which is not normally the case. Then the strategy keeps only goals involving extra large boxes that have failed less than four times. Return the result or, if none, all candidates. The strategy has a ceiling of four attempts because it is possible for the agents to get stuck in an endless loop otherwise.

Drop-Subgoal-Completed This strategy removes a top-level goal from the candidates if the agent has already completed a subgoal request from another agent toward accomplishing the goal.

Drop-Control This strategy removes a top-level goal from the candidates if any of the top-level literals are currently under the control of another agent.

	Ticks	Actions		Requests	Agreement Percent
		Attempted	Successful		
(D-A)	997.9	41.3	32.7	22.1	65.3%
(D-F K-D K-F)	1002.0	40.5	31.2	21.6	69.2%
(O-F)	1032.7	41.4	31.8	22.2	69.6%
(D-F)	1038.1	41.1	30.9	22.6	68.7%
(D-C)	1049.0	39.8	29.5	23.8	66.8%
()	1060.4	41.1	31.2	23.3	69.2%
(D-F K-X)	1079.7	40.0	29.6	24.5	67.5%
(D-S-C)	1162.3	43.5	33.0	25.4	70.4%

Table B.6: Performance of first episode for goal-selection strategies.

To test the significance of goal-selection on performance, the system was run on seven strategies other than the default. In these runs, the agents learned both operator probabilities and coordinated procedures over the course of five problems. Tables B.6 and B.7 show that the choice of goal-selection is relatively unimportant and that learning is effective regardless.

In these tables, the strategies are abbreviated by the first letter in each word, so Drop-Subgoal-Completed is abbreviated as D-S-C, *et cetera*. Since these were learning trials, all of the strategies shown are followed by Search-CB² and, if needed, Pick-Random. Thus, the default strategy is listed as (D-F).

	Ticks	Actions		Requests	Agreement Percent
		Attempted	Successful		
(D-F K-X)	671.8	25.0	23.8	11.8	82.5%
(D-F)	675.5	25.4	24.1	11.5	81.1%
(D-F K-D K-F)	675.7	25.8	24.5	11.5	81.0%
(O-F)	681.3	25.8	24.6	11.6	80.8%
(D-A)	691.6	26.5	25.4	11.6	82.3%
(D-C)	719.0	26.4	24.5	13.1	77.3%
()	727.5	26.6	25.0	13.0	77.7%
(D-S-C)	736.3	26.5	25.0	13.3	77.7%

Table B.7: Performance of fifth episode for goal-selection strategies.

²Figure 5.12 in the results chapter showed that learned coordinated procedures were not as effective without using Search-CB.

Appendix C

Pseudo-code

The algorithms given only scratch the surface. They are meant to clarify a handful of techniques referred to in the text.

See Section 6.1 for a description of this function

```
function plan-rating (plan state seniority-adjustment your-plan-flag)
  /* your-plan-flag is used for requests in fonction calc-action-prob */
  goals ← top-level goals achieved by plan
  if goals ≠ ∅
    then return seniority-adjustment *
      expected-reward-per-tick ( plan state goals your-plan-flag 0 )
  else return seniority-adjustment *
    expected-reward-per-tick ( plan state ∅ your-plan-flag
      domain-duration (attempted-top-level-goal ending-state (plan)))
```

```
function expected-reward-per-tick (plan state goals your-plan-flag extra-duration)
  return erpt ( plan goals your-plan-flag extra-duration 1.0 0 )
```

```
function domain-duration (goal end-state)
  ATP * MW-domain-distance ( goal end-state )
return  $\frac{\text{ATP} * \text{MW-domain-distance}(\text{goal end-state})}{\text{DefaultActionProbability}}$ 
```

```
function MW-domain-distance (goal-object end-state)
return
  1 if goal-object is being held or on a hand-truck that is at the street
  2 if goal-object is on another object or the ground at the street
  3 if goal-object is being held or on a hand-truck that is not at the street
  4 otherwise
```

```

function compute-reward (state goals)
  return number of goals which hold given the state
function erpt (plan state goals your-plan-flag extra-dur cum-prob cum-dur)
  reward  $\leftarrow$  compute-reward ( state goals )
  if plan then
    step  $\leftarrow$  first action  $\in$  plan
    prob  $\leftarrow$  calc-action-prob ( step your-plan-flag )
    cum-prob * ( 1 - prob ) * reward
    reward-if-fail  $\leftarrow$   $\frac{\text{cum-prob} * (1 - \text{prob}) * \text{reward}}{\text{cum-dur} + \text{failing-dur}(\text{step})}$ 
  return reward-if-fail +
    erpt ( remaining-plan incorporate-effects ( step state )
          goals your-plan-flag extra-dur ( cum-prob * prob )
          ( cum-dur + success-dur ( step ) ) )
  else return ( cum-prob * reward ) / ( cum-dur + extra-dur )
function calc-action-prob (action your-plan-flag)
  if action is a WAIT-FOR
    if your-plan-flag is true
      then return 1 /* Will get the matching request later */
      /* This underestimates: WAIT-FORs are expanded into two or more actions later */
    else return square ( prob ( action ) )
  else if action is a WAIT
    then return 1 /* Just got the matching request */
  else return prob ( action )

```

See Section 7.2 for a description of this function

```

function compute-time-groupings (goal-groups)
  time-groups  $\leftarrow$   $\emptyset$  /* goal-groups are sorted by start-time, end-time */
  foreach goal-group  $\in$  goal-groups
    foreach time-group  $\in$  time-groups
      if goal-group  $\notin$  time-group and
        end-time (time-group)  $\geq$  start-time (goal-group)
      then /* create a new member of time-groups */
        start-time (new)  $\leftarrow$  start-time (time-group)
        end-time (new)  $\leftarrow$  max ( { start-time (time-group), start-time (goal-group) } )
        goals (new)  $\leftarrow$  append ( goals(time-group),
          set-difference ( goals(time-group), goals(goal-group) ) )
        members (new)  $\leftarrow$  members (time-group)  $\cup$  { goal-group }
    foreach later-group  $\in$  goal-groups that comes after goal-group
      if end-time (goal-group)  $\geq$  start-time (later-group)
        then create new member of time-groups
          by appending later-group to goal-group
      else break

```

See Section 7.7.1 for a description of this function

```

function compute-storing-indices (tobestored complete)
  /* The arguments are the optimized, summarized procedure to be stored and the
     complete coordinated procedure */
  group-intervals  $\leftarrow$  group-intervals (tobestored)
  times  $\leftarrow$   $\emptyset$ 
  unless (MaxStores = 1) and (KeepEarliest = true )
    start  $\leftarrow$  start-time ( earliest-interval ( group-intervals ) )
    match  $\leftarrow$  action  $\in$  complete such that start-time ( action ) = start
    times  $\leftarrow$  make-start-interval ( match start )
  stop  $\leftarrow$  interval-start head ( times )      /* head returns first element */
  return-states  $\leftarrow$  { find-chunk-state ( x ) | x  $\in$  times }
  trace-events  $\leftarrow$  complete
  do while ( true )
    if length (times) = MaxStores
      then break
    if trace-events =  $\emptyset$ 
      then break
    top  $\leftarrow$  head ( trace-events )
    trace-events  $\leftarrow$  trace-events  $\setminus$  { top }
    new-interval  $\leftarrow$  make-start-interval ( event event-start ( event ) )
    new  $\leftarrow$  earliest interval after merging new-interval with group-intervals
    if interval-start ( new ) = stop
      then break
    newstate  $\leftarrow$  find-chunk-state ( new )
    if newstate  $\notin$  return-states then
      return-states  $\leftarrow$  { newstate }  $\cup$  return-states
      times  $\leftarrow$  { new }  $\cup$  times
  return two values: return-states times

```

```

function make-start-interval ( event default )
  if event-action ( event ) was a SIGN or a WAIT
    then return make-interval ( find-before-message ( start-time ( event )
                                                       action-message ( event ) ) )
  else return make-list ( default )

```

```

function trace-match ( message event )
  if event-reason (event) = :WAITING and request-for-service ( event ) and
    message = (event-message event )
    then return true
  if event-reason (event)  $\in$  { :BUSY, :NO } and request-for-service ( event ) and
    message = (event-message event )
    then return true

```

```

function general-trace-match ( message trace )
  event  $\leftarrow$  head ( trace )
  time  $\leftarrow$  start-time ( event )
  if trace-match ( message event )
    then return true
  if the agent had a conversation about message starting at time
    then return true
  if someone made a request to the agent at time that was not about message and
    the agent's response  $\in$  { :NO, :CANT, :ACTIVE } and
    the agent was intending to either SIGN or WAIT about message at time and
    general-trace-match ( message tail ( trace ) ) /* tail drops first element */
    then return true

```

/ agent is an (unshown) argument to all of these functions */*

```

function find-before-message ( time message )
  trace  $\leftarrow$  agent-trace ( agent ) /* reverse chronological order */
  while start-time ( head ( trace ) ) > time
    trace  $\leftarrow$  trace \ { head ( trace ) }
  if trace eq  $\emptyset$ 
    then return 1
  else
    prev  $\leftarrow$  head ( trace )
    trace  $\leftarrow$  trace \ { prev }
    if  $\neg$  ( trace-match ( message ( prev ) ) )
      then return time
    else while ( true )
      if trace eq  $\emptyset$ 
        then return 1
      unless general-trace-match ( message trace )
        return start-time (prev)
      prev  $\leftarrow$  head ( trace )
      trace  $\leftarrow$  trace \ { prev }

```

See Section B.4 for a description of this function

```

function select-agent-goals (agent unmet-goals)
  use-memory  $\leftarrow$  true
  While  $\exists$  more than 1 unmet goal to choose from
    func  $\leftarrow$  next pruning function of agent's goal-selection strategy
    call func with arguments ( agent unmet-goals )
    /* this may return up to three values */
    unmet-goals  $\leftarrow$  the first returned value
    if ( the second returned value = false )
      then use-memory  $\leftarrow$  false
    if ( the third returned value = true )
      then break
  return two values: unmet-goals use-memory

```

See Section B.1 for descriptions of these functions

function RAT (my-plan your-plan state)

```

/* your-plan is the plan to achieve a request */
my-rating ← plan-rating ( my-plan state 1 false )
your-rating ← plan-rating ( your-plan state 1 true )
if my-rating > your-rating
  then return my-plan
else return your-plan

```

function HYB (my-plan your-plan)

```

my-rating ← plan-rating ( my-plan 1 false )
your-rating ← plan-rating ( your-plan 1 true )
if random() < ( my-rating / ( my-rating + your-rating ) )
  then return my-plan
else return your-plan

```

function SRR (my-plan your-plan)

```

if your-seniority > max ( { my-seniority } ∪ { seniorityi | already helping agenti } )
  then return your-plan
if your-seniority < my-seniority then
  myfact ← 1 + my-seniority - your-seniority
  yourfact ← 1
else
  yourfact ← 1 + your-seniority - my-seniority
  myfact ← 1
my-rating ← plan-rating ( my-plan myfact false )
your-rating ← plan-rating ( your-plan yourfact true )
if my-rating > your-rating
  then return my-plan
else return your-plan

```

function SRH (my-plan your-plan)

```

if your-seniority > max ( { my-seniority } ∪ { seniorityi | already helping agenti } )
  then return your-plan
if your-seniority < my-seniority then
  myfact ← 1 + my-seniority - your-seniority
  yourfact ← 1
else
  yourfact ← 1 + your-seniority - my-seniority
  myfact ← 1
my-rating ← plan-rating ( my-plan myfact false )
your-rating ← plan-rating ( your-plan yourfact true )
if random() < ( my-rating / ( my-rating + your-rating ) )
  then return my-plan
else return your-plan

```

Bibliography

- Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proc. Sixth National Conference on Artificial Intelligence*, pages 268–272, 1987.
- Richard Alterman and Andrew Garland. Convention in joint activity. *Cognitive Science*, 2001. To appear.
- Richard Alterman, Seth Landsman, Alex Feinman, and Joshua Introne. Groupware for planning. Technical Report CS-98-200, Brandeis University, 1998a.
- Richard Alterman, Roland Zito-Wolf, and Tamitha Carpenter. Pragmatic action. *Cognitive Science*, 22(1):53–106, 1998b.
- Richard Alterman. Adaptive planning. *Cognitive Science*, 12:393–421, 1988.
- Richard Alterman. The closed room. *Mind, Culture, and Activity*, 6(2):117–142, 1999.
- J.M. Bradshaw. An introduction to software agents. In J.M. Bradshaw, editor, *Software Agents*, pages 3–46. MIT Press, 1997.
- Michael E. Bratman. Shared cooperative activity. In *Faces of Intention*, pages 93–108. Cambridge University Press, 1999.
- Rodney Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
- Barry Brumitt, Martial Hebert, and the CMU UGV Group. Experiments in autonomous driving with concurrent goals and multiple vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1998.
- Mark H. Burstein, editor. *Proceedings of the ARPA/Rome Laboratory Planning Initiative Workshop*. Morgan Kaufmann, 1994.
- C.F. Camerer and E.J. Johnson. The process-performance paradox in expert judgement: How can experts know so much and predict so badly? In K.A. Ericsson and J. Smith, editors, *Toward a General Theory of Expertise: Prospects and limits*, pages 195–217. Cambridge University Press, Cambridge, England, 1991.

- Jamie Carbonell. Derivational analogy and its role in problem solving. In *Proc. Third National Conference on Artificial Intelligence*, 1983.
- Tamitha Carpenter. *Instruction Usage*. PhD thesis, Brandeis University, 1998.
- David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.
- Herbert H. Clark. *Using Language*. Cambridge University Press, 1996.
- Daniel D. Corkill. Hierarchical planning in a distributed environment. In *Proc. Sixth International Joint Conference on Artificial Intelligence*, pages 168–175, 1979.
- Randall Davis and Reid Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.
- Keith S. Decker and Victor R. Lesser. Generalized partial global planning. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):319–346, 1992.
- Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.
- L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. A personal learning apprentice. In *Proc. Tenth National Conference on Artificial Intelligence*, pages 96–103. AAAI Press, 1992.
- Marie E. desJardins, Edmund H. Durfee, Charles L. Ortiz, and Michael J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.
- Edmund H. Durfee and Victor R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 1991.
- Edmund H. Durfee, Patrick G. Kenny, and Karl C. Kluge. Integrated premission planning and execution for unmanned ground vehicles. *Autonomous Robots*, 5:1–14, 1998.
- Eithan Ephrati, Martha E. Pollack, and Jeffrey S. Rosenschien. A tractable heuristic that maximizes global utility through local plan combination. In *Proc. First International Conference on Multiagent Systems*, pages 94–101, 1995.
- K. Anders Ericsson and Neil Charness. Expert performance: Its structure and acquisition. *American Psychologist*, 49(8):725–747, 1994.

- K. Anders Ericsson and A. C. Lehmann. Expert and exceptional performance: Evidence of maximal adaptation to task constraints. *Annual Review of Psychology*, 47:273–305, 1996.
- Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.
- Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.
- T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzon, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, and C. Beck. Specification of the KQML agent-communication language. Technical Report EIT TR 92-04, Enterprise Integration Technologies, Palo Alto, CA, 1993. Revised June 15, 1993.
- R. James Firby. An investigation into reactive planning in complex domains. In *Proc. Sixth National Conference on Artificial Intelligence*, pages 202–206, 1987.
- Douglas Fisher and Jungsoon Yoo. Categorization, concept learning, and problem solving: A unifying view. *Categorization by Humans and Machines*, 29:219–255, 1993. Special volume of *The Psychology of Learning and Motivation*.
- Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.
- David E. Foulser, Ming Li, and Qiang Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2):43–181, 1992.
- Uwe Franke, D. Gavrila, Steffen Görzig, Frank Lindner, Frank Paetzold, and Christian Wöhler. Autonomous driving goes downtown. *IEEE Intelligent Systems*, 13(6), November/December 1998.
- Andrew Garland and Richard Alterman. Preparation of multi-agent knowledge for reuse. In *1995 AAAI Fall Symposium on Adaptation of Knowledge for Reuse*, pages 26–33, 1995. <http://www.cs.brandeis.edu/~aeg/papers/garland.aaifs95.ps>.
- Andrew Garland and Richard Alterman. Multi-agent learning through collective memory. In *1996 AAAI Spring Symposium on Adaptation, Coevolution and Learning in Multiagent Systems*, pages 33–38, 1996. <http://www.cs.brandeis.edu/~aeg/papers/garland.aaiss96.ps>.
- Andrew Garland and Richard Alterman. Learning cooperative procedures. In Ralph Bergmann and Alexander Kott, editors, *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*. AAAI Technical Report WS-98-02, pages 54–61. AAAI Press, 1998. <http://www.cs.brandeis.edu/~aeg/papers/garland.aipsw98.ps>.

- Michael Genesereth, Matt Ginsberg, and Jeffrey Rosenschien. Cooperation without communication. In *Proc. Fifth National Conference on Artificial Intelligence*, pages 51–57, 1986.
- Michael P. Georgeff. The representation of events in multi-agent domains. In *Proc. Third National Conference on Artificial Intelligence*, pages 125–129, 1983.
- P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe. The utility of communication in coordinating intelligent agents. In *Proc. Ninth National Conference on Artificial Intelligence*, pages 166–172. AAAI/MIT Press, 1991.
- Johann Wolfgang von Goethe. *Scientific Studies*, volume 12 of *Goethe edition*. Suhrkamp, New York, NY, 1988. Translated and edited by Douglas Miller.
- Marc Goodman. Projective visualization: Acting from experience. In *Proc. Eleventh National Conference on Artificial Intelligence*, 1993.
- Barbara Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.
- Barbara Grosz and Sarit Kraus. The evolution of SharedPlans. In A. Rao and M. Woolridge, editors, *Foundations and Theories of Rational Agency*, pages 227–262. Kluwer, 1999.
- Barbara Grosz and Candace Sidner. Plans for discourse. In Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, pages 417–444. Bradford Books, Cambridge, MA, 1990.
- Kristian J. Hammond. Case-based planning: A framework for planning from experience. *Cognitive Science*, 14:385–443, 1990.
- Thomas Haynes and Sandip Sen. Learning cases to resolve conflicts and improve group behavior. *International Journal of Human-Computer Studies*, 48:31–49, 1998.
- F. Ho and M. Kamel. Learning coordination strategies for cooperative multiagent systems. *Machine Learning*, 33(2-3):155–177, 1998. Special issue on Multiagent Learning, edited by Michael Huhns and Gerhard Weiß.
- Marcus J. Huber and Edmund H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proc. First International Conference on Multiagent Systems*, pages 163–170, 1995.
- Marcus J. Huber and Tedd Hadley. Multiple roles, multiple teams, dynamic environment: Autonomous Netrek agents. In W. Lewis Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, New York, 1997. ACM Press.

- Edwin Hutchins. *Cognition in the Wild*. MIT Press, Cambridge MA, 1995.
- Heecheol Jeon, Charles Petrie, and Mark R. Cutkosky. JATLite: A Java agent infrastructure with message routing. *IEEE Internet Computing*, 4(2):87–96, March–April 2000.
- Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.
- Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, Stanford University, 1990.
- Leslie Pack Kaelbling. <http://www.cs.brown.edu/people/lpk/home.html>, May 2000.
- Subbarao Kambhampati and James A. Hendler. Control of refitting during plan reuse. *Artificial Intelligence*, 55:193–258, 1992.
- Craig A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proc. Eighth National Conference on Artificial Intelligence*, pages 923–928, 1990.
- Janet L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
- Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76, 1995.
- Yannis Labrou and Tim Finin. Proposal for a new KQML specification. Technical Report CS-97-03, University of Maryland Baltimore County, February 1997.
- J. E. Laird, P. S. Rosenbloom, and A. Newell. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.
- Pat Langley and John A. Allen. Learning, memory, and search in planning. In *Proc. Thirteenth Annual Conference of the Cognitive Science Society*, pages 364–369, Chicago, Illinois, 1991.
- Yezdi Lashkari, Max Metral, and Pattie Maes. Collaborative interface agents. In *Proc. Twelfth National Conference on Artificial Intelligence*, pages 444–449, 1994.
- Jean Lave and Etienne Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge University Press, Cambridge, 1991.
- V.R. Lesser, R. D. Fennell, L. D. Erman, and D. R. Reddy. Organization of the HEARSAY-II speech understanding system. *IEEE Transactions on Acoustics, Speech, Signal Processing*, 23:11–23, 1975.

- Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *Proc. Eighth National Conference on Artificial Intelligence*, pages 94–99, July 1990.
- Moses Ma, editor. *Agents in E-commerce*, pages 78–110. Communications of the ACM. ACM Press, New York, NY, March 1999.
- Patti Maes and Rodney A. Brooks. Learning to coordinate behaviors. In *Proc. Eighth National Conference on Artificial Intelligence*, pages 796–802, 1990.
- P. Maes and R. Kozierok. Learning interface agents. In *Proc. Eleventh National Conference on Artificial Intelligence*, pages 459–465. AAAI Press, 1993.
- Patty Maes. Agents that reduce work and information overload. In J.M. Bradshaw, editor, *Software Agents*, pages 145–164. MIT Press, 1997.
- Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *IEEE Internet Computing*, 26(1):87–119, March 1994.
- Maja Mataric. Designing emergent behaviors: From local interactions to collective intelligence. In *From Animals to Animats 2*, pages 432–441, 1992.
- John McCarthy. Epistemological problems in Artificial Intelligence. In *Proc. Fifth International Joint Conference on Artificial Intelligence*, 1977.
- M. Minsky. A framework for representing knowledge. In P.W. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, NY, 1975.
- Steven Minton. *Learning Search Control Knowledge*. Kluwer, 1988.
- T. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.
- M V NagendraPrasad, Victor R. Lesser, and Susan Lander. Retrieval and reasoning in distributed case bases. Technical Report CS TR 95-27, University of Massachusetts, 1995.
- D. A. Norman. *The Invisible Computer: Why Good Products Can Fail, the PC Is So Complex, and Information Appliances Are the Answer*. MIT Press, Cambridge, MA, 1998.
- Takuya Ohko, Kazuo Hiraki, and Yuichiro Anzai. Learning to reduce communication costs in task negotiation among multiple autonomous mobile robots. In Gerhard Weiß and Sandip Sen, editors, *Adaptation and Learning in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence, pages 177–190. Springer-Verlag, Berlin, 1996.

- M. P. Papazoglou. The role of agent technology in business to business electronic commerce. In Matthias Klusch, Onn M. Shehory, and Gerhard Weiß, editors, *Cooperative Information Agents III*, number 1652 in Lecture Notes in Artificial Intelligence, pages 245–264. Springer-Verlag, Berlin, 1999. Invited contribution.
- Martha Pollack. Plans as complex mental attitudes. In Philip Cohen, Jerry Morgan, and Martha Pollack, editors, *Intentions in Communication*, pages 77–103. MIT Press, 1990.
- Martha E. Pollack. Planning in dynamic environments: The DIPART system. In Austin Tate, editor, *Advanced Planning Technology*. AAAI Press, Cambridge, 1996.
- Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proc. First International Conference on Multiagent Systems*, pages 312–319, 1995.
- Bradley Rhodes and Thad Starner. The remembrance agent: A continuously running automated information retrieval system. In *The Proceedings of The First International Conference on The Practical Application of Intelligent Agents and Multi Agent Technology (PAAM '96)*, pages 487–495, 1996.
- Charles Rich and Candy Sidner. COLLAGEN: A collaboration manager for software interface agents. Technical Report 97-21a, Mitsubishi Electric Research Labs, March 1998.
- E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.
- Roger Schank. *Dynamic Memory: A Theory of Reminding and Learning In Computers and People*. Cambridge University Press, Cambridge, 1982.
- Thomas C. Schelling. *The strategy of conflict*. Oxford University Press, New York, NY, 1963. First published in 1960.
- Sandip Sen and Mahendra Sekeran. Individual learning of coordination knowledge. *Journal of Experimental and Theoretical Artificial Intelligence*, 10, 1998.
- Sandip Sen, Mahendra Sekeran, and John Hale. Learning to coordinate without sharing information. In *Proc. Twelfth National Conference on Artificial Intelligence*, pages 426–431, 1994.
- Barry Smyth and Mark T. Keane. Remembering to forget. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence*, pages 377–382, 1995.
- Anthony Stentz, John Bares, Sanjiv Singh, and Patrick Rowe. A robotic excavator for autonomous truck loading. *Autonomous Robots*, June 1999.

- Peter Stone and Manuela Veloso. Towards collaborative and adversarial learning: a case study in robotic soccer. *International Journal of Human-Computer Studies*, 48:83–104, 1998.
- Peter Stone and Manuela Veloso. Task decomposition, role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 111(1):241–273, 1999.
- Lucy A. Suchman. *Plans and Situated Actions*. Cambridge University Press, Cambridge, 1987.
- Toshiharu Sugawara and Victor Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33(2-3):129–153, 1998. Special issue on Multiagent Learning, edited by Michael Huhns and Gerhard Weiß.
- Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–43, 1988.
- Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
- Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.
- W. Truszkowski, H. Hallock, and J. Kurien. Agent technology from a NASA perspective. In Matthias Klusch, Onn M. Shehory, and Gerhard Weiß, editors, *Cooperative Information Agents III*, number 1652 in Lecture Notes in Artificial Intelligence, pages 1–33. Springer-Verlag, Berlin, 1999. Invited contribution.
- Elise Turner. Exploiting problem solving to select information to include in dialogues between cooperating agents. In *Proc. Sixteenth Annual Conference of the Cognitive Science Society*, pages 882–886, Hillsdale, NJ, 1994. Lawrence Erlbaum Associates.
- Manuela Veloso and Jaime Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10:249–278, 1993.
- Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Lecture Notes in Artificial Intelligence, volume 886. Springer-Verlag, Berlin, 1994.
- J.F. Voss, T.R. Green, T.A. Post, and B.C. Penner. Problem-solving skill in the social sciences. In G.H. Bower, editor, *The Psychology of Learning and Motivation: Advances in Research and Theory*, volume 17, pages 165–213. Academic, New York, 1983.

- David Wilkens and Karen Myers. A multiagent planning architecture. In *Proc. Fourth International Conference on Artificial Intelligence Planning Systems*, pages 154–162, 1998.
- Michael Wolverton and Marie desJardins. Controlling communication in distributed planning using irrelevance reasoning. In *Proc. 16th National Conference on Artificial Intelligence*, pages 868–874, 1998.
- Holly Yanco and Lynn Andrea Stein. An adaptive communication protocol for cooperating mobile robots. In *Animals to Animats: Proc. of the Second International Conference on the Simulation of Adaptive Behavior*, pages 478–485. MIT Press, 1993.
- Hua Yang. Learning abstract and macro operators in AI planning. Technical Report CS-93-05, Vanderbilt University, 1993.
- Roland Zito-Wolf. *Case-Based Representations for Procedural Knowledge*. PhD thesis, Brandeis University, 1993.