3 Distributed Problem Solving and Planning

Edmund H. Durfee

3.1 Introduction

Distributed problem solving is the name applied to a subfield of distributed AI in which the emphasis is on getting agents to work together well to solve problems that require collective effort. Due to an inherent distribution of resources such as knowledge, capability, information, and expertise among the agents, an agent in a distributed problem-solving system is unable to accomplish its own tasks alone, or at least can accomplish its tasks better (more quickly, completely, precisely, or certainly) when working with others.

Solving distributed problems well demands both group coherence (that is, agents need to want to work together) and **competence** (that is, agents need to know how to work together well). As the reader by now recognizes, group coherence is hard to realize among individually-motivated agents (see chapters 2 and 5, for example). In distributed problem solving, we typically assume a fair degree of coherence is already present: the agents have been designed to work together; or the payoffs to self-interested agents are only accrued through collective efforts; or social engineering has introduced disincentives for agent individualism; etc. Distributed problem solving thus concentrates on competence; as anyone who has played on a team, worked on a group project, or performed in an orchestra can tell you, simply having the desire to work together by no means ensures a competent collective outcome!

Distributed problem solving presumes the existence of problems that need to be solved and expectations about what constitute solutions. For example, a problem to solve might be for a team of (computational) agents to design an artifact (say, a car). The solution they formulate must satisfy overall requirements (it should have four wheels, the engine should fit within the engine compartment and be powerful enough to move the car, etc.), and must exist in a particular form (a specification document for the assembly plant). The teamed agents formulate solutions by each tackling (one or more) subproblems and synthesizing these subproblem solutions into overall solutions.

Sometimes the problem the agents are solving is to construct a plan. And of-

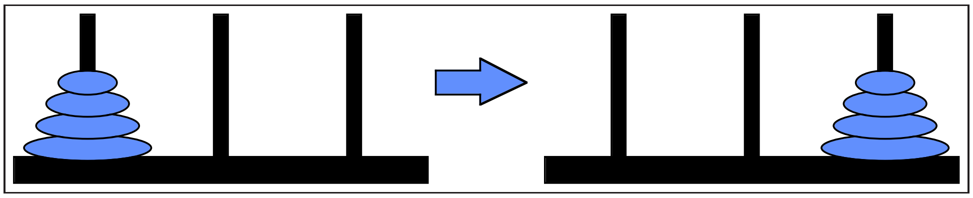

Figure 3.1 Tower of Hanoi (ToH).

ten, even if the agents are solving other kinds of problems, they also have to solve planning problems as well. That is, how the agents should plan to work together—decompose problems into subproblems, allocate these subproblems, exchange subproblem solutions, and synthesize overall solutions—is itself a problem the agents need to solve. Distributed planning is thus tightly intertwined with distributed problem solving, being both a problem in itself and a means to solving a problem.

In this chapter, we will build on the topics of the previous chapters to describe the concepts and algorithms that comprise the foundations of distributed problem solving and planning. The reader is already familiar with protocols of interaction; here we describe how those protocols are used in the context of distributed problem solving and planning. The reader is also assumed to be familiar with traditional AI search techniques; since problem solving and planning are usually accomplished through search, we make liberal use of the relevant concepts. The subsequent chapter delves more formally into distributed search specifically.

The remainder of the chapter is structured as follows. We begin by introducing some representative example problems, as well as overviewing a variety of other applications of the techniques to be described. Working from these motivating examples, we work our way up through a series of algorithms and concepts as we introduce increasingly complicated requirements into the kinds of problems to solve, including planning problems.

3.2 Example Problems

There are several motivations for distributed problem solving and distributed planning. One obvious motivation is that using distributed resources concurrently can allow a speedup of problem solving thanks to parallelism. The possible improvements due to parallelism depend, of course, on the degree of parallelism inherent in a problem.

One problem that permits a large amount of parallelism during planning is a classic toy problem from the AI literature: the **Tower of Hanoi** (ToH) problem (see Figure 3.1). As the reader will recall from an introductory AI course, ToH consists of 3 pegs and n disks of graduated sizes. The starting situation has all of the disks on one peg, largest at bottom to smallest at top. The goal is to move the disks from the start peg to another specified peg, moving only one disk at a time, without ever placing a larger disk on top of a smaller disk. The problem, then, is to find a sequence of moves that will achieve the goal state.

A second motivation for distributed problem solving and planning is that expertise or other problem-solving capabilities can be inherently distributed. For example, in concurrent engineering, a problem could involve designing and manufacturing an artifact (such as a car) by allowing specialized agents to individually formulate components and processes, and combining these into a collective solution. Or, supervisory systems for air-traffic control, factory automation, or crisis management can involve an interplay between separate pieces for event monitoring, situation assessment, diagnosis, prioritization, and response generation. In these kinds of systems, the problem is to employ diverse capabilities to solve problems that are not only large (the ToH can itself be arbitrarily large) but also multi-faceted.

As a simple example of distributed capability, we will use the example of **distributed sensor network establishment** for monitoring a large area for vehicle movements. In this kind of problem, the overall task of monitoring cannot be done in a central location since the large area cannot be sensed from any single location. The establishment problem is thus to decompose the larger monitoring task into subtasks that can be allocated appropriately to geographically distributed agents.

A third motivation is related to the second, and that is that beliefs or other data can be distributed. For example, following the successful solution of the distributed sensor network *establishment* problem just described, the problem of actually doing the **distributed vehicle monitoring** could in principle be centralized: each of the distributed sensor agents could transmit raw data to a central site to be interpreted into a global view. This centralized strategy, however, could involve tremendous amounts of unnecessary communication compared to allowing the separate sensor agents to formulate local interpretations that could then be transmitted selectively.

Finally, a fourth motivation is that the results of problem solving or planning might need to be distributed to be acted on by multiple agents. For example, in a task involving the delivery of objects between locations, **distributed delivery** agents can act in parallel (see Figure 3.2). The formation of the plans that they execute could be done at a centralized site (a dispatcher) or could involve distributed problem- solving among them. Moreover, during the execution of their plans, features of the environment that were not known at planning time, or that unexpectedly change, can trigger changes in what the agents should do. Again, all such decisions could be routed through a central coordinator, but for a variety of reasons (exploiting parallelism, sporadic coordinator availability, slow communication channels, etc.) it could be preferable for the agents to modify their plans unilaterally or with limited communication among them.

In the above, we have identified several of the motivations for distributed problem solving and planning, and have enumerated examples of the kinds of applications

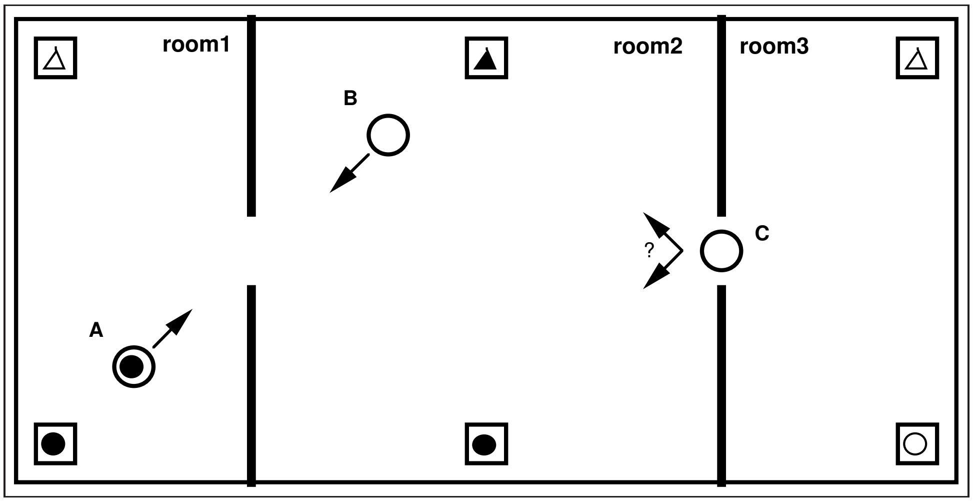

Figure 3.2 Distributed delivery example.

for which these techniques make sense. In the rest of this chapter, we will refer back to several of these kinds of application problems, specifically:

- Tower of Hanoi (ToH)
- Distributed Sensor Network Establishment (DSNE)
- Distributed Vehicle Monitoring (DVM)
- Distributed Delivery (DD)

3.3 Task Sharing

The first class of distributed problem-solving strategies that we will consider have been called "task sharing" or "task passing" strategies in the literature. The idea is simple. When an agent has many tasks to do, it should enlist the help of agents with few or no tasks. The main steps in task sharing are:

1. **Task decomposition**: Generate the set of tasks to potentially be passed to others. This could generally involve decomposing large tasks into subtasks that could be tackled by different agents.
2. **Task allocation**: Assign subtasks to appropriate agents.
3. **Task accomplishment**: The appropriate agents each accomplish their subtasks, which could include further decomposition and subsubtask assignment, recursively to the point that an agent can accomplish the task it is handed alone.

4. **Result synthesis**: When an agent accomplishes its subtask, it passes the result to the appropriate agent (usually the original agent, since it knows the decomposition decisions and thus is most likely to know how to compose the results into an overall solution).

Note that, depending on the circumstances, different steps might be more or less difficult. For example, sometimes an overburdened agent begins with a bundle of separate tasks, so decomposition is unnecessary; sometimes the agent can pass tasks off to any of a number of identical agents, so allocation is trivial; and sometimes accomplishing the tasks does not yield any results that need to be synthesized in any complex way.

3.3.1 Task Sharing in the ToH Problem

To get a feel for the possibilities of task sharing, we start with the very simple ToH problem. Consider the task-sharing steps when it comes to this problem:

1. Task decomposition: Means-ends analysis (see Figure 3.3), where moving the largest disk that is not at its destination peg is considered the most important difference, leads to a recursive decomposition: solve the problem of getting to the state where the largest disk can be moved, and get from the state after it is moved to the goal state. These subproblems can be further decomposed into problems of moving the second largest disk to the middle peg to get it out of the way, so the state where that can be done needs to be reached, etc.
2. Task allocation: If we assume an indefinite number of identical idle agents capable of solving (pieces of) the ToH problem, then allocation reduces to just assigning a task randomly to one of these agents.
3. Task accomplishment: In general, an agent can use means-ends analysis to find the most significant difference between the start and goal states that it is responsible for, and will decompose the problem based on these. If the decomposed problems are such that the start and goal states are the same (that is, where the most significant difference is also the only difference), then the recursive decomposition terminates.
4. Result synthesis: When an agent has solved its problem, it passes the solution back on up. When an agent has received solutions to all of the subproblems it passed down, it can compose these into a more comprehensive sequence of moves, and then pass this up as its solution.

ToH represents an ideal case of the possibilities of distributed problem solving due to the hierarchical nature of the problem. In general, for a problem like ToH, the search space is exponential in size. If we assume a branching factor of b (meaning that from a state, there are b alternative states that can be reached by moving some disk to some peg), and assuming that in the best case it will take n disk movements

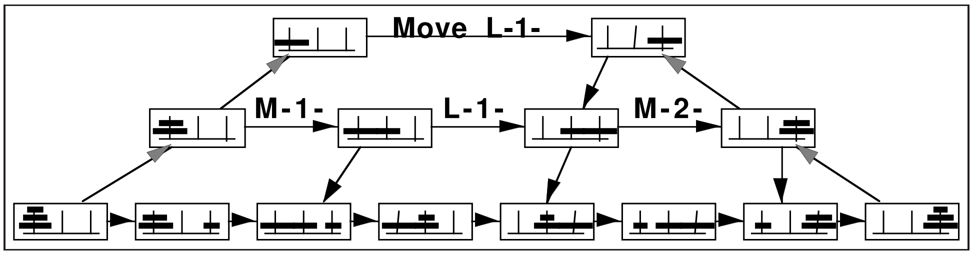

Figure 3.3 Means-ends decomposition for ToH.

to go from the start state to the end state, then the search complexity is b^n.

Thanks to the hierarchical structure of the problem, the means-ends heuristic can reduce this complexity dramatically. Let us assume that ultimately the hierarchy divides the problem of size n into problems each of size k, yielding n/k subproblems, each of which requires $f(k)$ time to solve. These solutions are fed to the next level up in the hierarchy such that k are given to each of the agents at this level. Each of these n/k^2 agents has to synthesize k results, again requiring $f(k)$ time. This aggregation process continues up the hierarchy, such that at the next-to-topmost level, n/k^{l-1} agents are combining k results from below in the hierarchy with l levels. The topmost agent then combines these n/k^{l-1} results together, requiring $f(n/k^{l-1})$ time. The total expenditure is thus:

$$f(n/k^{l-1}) + (n/k^{l-1} \cdot f(k)) + (n/k^{l-2} \cdot f(k)) + \ldots + (n/k \cdot f(k)) \quad .$$

Since k is a constant, and we can choose $l = \log_k n$, the equation can be reduced to $O([(k^l - 1)/(k-1)]f(k))$ which can be simplified simply to $O(n)$ [23, 23]. More importantly, if each level of the hierarchy has agents that solve their subproblems in parallel, then the time needed below the top of the hierarchy (assuming negligible distribution and communication time) is simply $f(k)$ for each level, so $(l-1)f(k)$. This is added to the top agent's calculation $f(n/k^{l-1})$. Again, since k (and hence $f(k)$) is constant, and $l = \log_k n$, this reduces simply to $O(\log_k n)$. This means that through decomposition and parallel problem solving, the exponential ToH problem can be reduced to logarithmic time complexity [33].

What the ToH problem illustrates is the potential for improved parallelism due to distributed problem solving in the ideally decomposable case. Unfortunately, few problems satisfy the assumptions in this analysis of ToH, including:

1. There is no backtracking back upward in the abstraction hierarchy, meaning that each distributed subproblem is solvable independently and the solution of one does not affect the solution of others. We will consider the effects of relaxing this assumption in Subection 3.3.4.
2. The solution found hierarchically approximates (is linear in length to) the solution that would be found using brute-force centralized search. This depends

on having hierarchical abstraction spaces that do not exclude good solutions as a consequence of reducing complexity.

3. The number of abstraction levels grows with the problem size. While doing this is easy for ToH, often the number of levels is fixed by the domain rather than the specific problem instance.
4. The ratio between levels is the base of the logarithm, k. Again, this depends on how the abstraction space is constructed.
5. The problems can be decomposed into equal-sized subproblems. This is very difficult in domains where problems are decomposed into qualitatively different pieces, requiring different expertise. We consider the effects of relaxing this assumption in Subsection 3.3.2.
6. There are at least as many agents as there are "leaf" subproblems. Clearly, this will be difficult to scale!
7. The processes of decomposing problems, distributing subproblems, and collecting results takes negligible time. We consider some of the effects of relaxing this assumption at various places in this chapter.

3.3.2 Task Sharing in Heterogeneous Systems

One of the powerful motivations for distributed problem solving is that it is difficult to build artifacts (or train humans) to be competent in every possible task. Moreover, even if it feasible to build (or train) an omni-capable agent, it is often overkill because, at any given time, most of those capabilities will go to waste. The strategy in human systems, and adopted in many distributed problem-solving systems, is to bring together on demand combinations of specialists in different areas to combine their expertise to solve problems that are beyond their individual capabilities.

In the ToH example, the subproblems required identical capabilities, and so the decisions about where to send tasks was extremely simple. When agents can have different capabilities, and different subproblems require different capabilities, then the assignment of subproblems to agents is not so simple.

Conceptually, it is possible for an agent to have a table that identifies the capabilities of agents, so that it can simply select an appropriate agent and send the subproblem off, but usually the decisions need to be based on more dynamic information. For example, if several candidate agents are capable of solving a subproblem, but some are already committed to other subproblems, how is this discovered? One way is to use the Contract Net protocol (Chapter 2) with directed contracts or focused addressing: the agent (in Contract-Net terms, the *manager*) announces a subproblem to a specific agent (in the case of directed contracts) or a focused subset of other agents (in focused addressing) based on the table of capabilities, and requests that returned bids describe acceptance/availability. The manager can then award the subproblem to the directed contractor if it accepts, or to one of the available contractors in the focused addressing set. However, if none of the agents are available, the manager has several options, described in the following paragraphs.

Broadcast Contracting

In the kind of open environment for which Contract Net was envisioned, it is unlikely that a manager will be acquainted with all of the possible contractors in its world. Thus, while directed contracts and focused addressing might be reasonable first tries (to minimize communication in the network), a manager might want to update its knowledge of eligible contractors by broadcasting its announcement to reach agents that it is currently unaware of as well. This is the most commonly considered mode of operation for Contract Net. Directed contracts and focused addressing can be thought of as caching results of such broadcasts, but since the cached results can become outdated, many implementations of Contract Net do not include this function. It is interesting to note, however, that this kind of "capabilities database" has found renewed favor in knowledge sharing efforts such as KQML (Chapter 2), where some agents explicitly adopt the task of keeping track of what other agents purport to be good at.

Retry

One very simple strategy is to retry the announcement periodically, assuming that eventually a contractor will free up. The retry interval then becomes an important parameter: if retries happen too slowly, then many inefficiencies can arise as agents do not utilize each other well; but if retries happen to quickly, the network can get bogged down with messages. One strategy for overcoming such a situation is to turn the protocol on its head. Rather than announcing tasks and collecting bids, which implies that usually there are several bidders for each task, instead the protocol can be used by potential contractors to announce availability, and managers can respond to the announcements by bidding their pending tasks! It is possible to have a system alternate between the task and availability announcement strategies depending on where the bottlenecks are in the system at various times [41].

Announcement Revision

Part of the announcement message that a manager sends is the eligibility specification for potential contractors. When no (satisfactory) contractors respond to an announcement, it could be that the manager was being too exclusive in whom it would entertain bids from. Thus, the manager could engage in iterative revision of its announcement, relaxing eligibility requirements until it begins to receive bids.

An interesting aspect of this relaxation process is that the eligibility specifications could well reflect preferences over different classes of contractors – or, more specifically, over the quality of services that different contractors provide. In concert with other methods of handling a lack of bids (described above), a manager will be deciding the relative importance of having a preferred contractor eventually pursue the subproblem compared to finding a suboptimal contractor sooner. In many cases, these preferences and tradeoffs between them can be captured using economic representations. By describing parts of its marginal utility curve, for example, a manager can provide tradeoff information to an auction, which can then apply principled algorithms to optimize the allocation of capabilities (see Chapter 5).

Alternative Decompositions

The manager can try decomposing the overall problem differently such that contractors are available for the alternative subproblems. In general, the relationship between problem decomposition and subproblem allocation is extremely complex and has not received sufficient attention. Sometimes a manager should first determine the space of alternative contractors to focus problem decomposition, while other times the space of decompositions can be very restrictive. Moreover, decisions about the number of problems to decompose into and the granularity of those subproblems will depend on other features of the application environment, including communication delays. We say no more about these issues here, other than to stress the research opportunities in this area.

3.3.3 Task Sharing for DSNE

Smith and Davis (and others since) have explored the use of the Contract Net protocol for a variety of problems, including the Distributed Sensor Net Establishment (DSNE) problem [4]. To give the reader a flavor of this approach, we briefly summarize the stages of this application.

At the outset, it is assumed that a particular agent is given the task of monitoring a wide geographic area. This agent has expertise in how to perform the overall task, but is incapable of sensing all of the area from its own locality. Therefore, the first step is that an agent recognizes that it can perform its task better (or at all) if it enlists the help of other agents. Given this recognition, it then needs to create subtasks to offload to other agents. In the DSNE problem, it can use its representation of the structure of the task to identify that it needs sensing done (and sensed data returned) from remote areas. Given this decomposition, it then uses the protocol to match these sensing subtasks with available agents. It announces (either directed, focused, or broadcast) a subtask; we leave out the details of the message fields since they were given in Chapter 2.

The important aspects of the announcement for our purposes here are the eligibility specification, the task abstraction, and the bid specification. To be eligible for this task requires that the bidding agent have a sensor position within the required sensing area identified and that it have the desired sensing capabilities. Agents that meet these requirements can then analyze the task abstraction (what, at an abstract level, is the task being asked of the bidders) and can determine the degree to which it is willing and able to perform the task, from its perspective. Based on this analysis, an eligible agent can bid on the task, where the content of a bid is dictated by the bid specification.

The agent with the task receives back zero or more bids. If it gets back no bids, then it faces the options previously described: it can give up, try again, broaden the eligibility requirements to increase the pool of potential bidders, or decompose the task differently to target a different pool of bidders. If it gets back bids, it could be that none are acceptable to it, and it is as if it got none back. If one or more is acceptable, then it can award the sensing subtask to one (or possible several) of the bidding agents. Note that, because the agent with the task has a choice over what it announces and what bids it accepts, and an eligible agent has a choice over whether it wants to bid and what content to put into its bid, no agent is forced to be part of a contract. The agents engage in a rudimentary form of negotiation, and form teams through *mutual selection*.

3.3.4 Task Sharing for Interdependent Tasks

For problems like ToH, tasks can be accomplished independently; the sequence of actions to get from the start state to an intermediate state can be found completely separately from the sequence to get from that intermediate state to the goal state. Thus, the subtasks can be accomplished in any order (or concurrently), and synthesis need only wait to complete until they are all done.

In some cases, contracted tasks are not independent. In a concurrent engineering application, for example, process planning subtasks usually need to wait until product design tasks have progressed beyond a certain point. For relatively clearcut subtask relationships, a manager for the subtasks can coordinate their execution by initiating a subtask based on the progress of another, or by relaying interim results for one subtask to contractors of related subtasks.

More generally, however, aspects of subtask relationships might only become apparent during the course of problem solving, rather than being dictated ahead of time by the problem decomposition. For example, when using a distributed sensor network to perform vehicle monitoring, the runtime relationships between what is being monitored in different areas is as variable as the possible movements of vehicles through the areas. While a task-sharing strategy, exemplified in the Contract Net protocol, can *establish* a distributed sensor network, it does not provide a sufficient basis for using the network. Or, put more correctly, when task sharing is used to allocate *classes* of tasks among agents, then if different instances of those tasks have different interrelationships, discovering and exploiting those relationships requires the generation and sharing of tentative results.

3.4 Result Sharing

A problem-solving task is accomplished within the context of the problem solver, so the results of the task if performed by one problem solver could well differ from the results of the same task being performed by another problem solver. For example, students in a class are often given the same task (homework problem), but their independently derived solutions will not (better not!) be identical.

By sharing results, problem solvers can improve group performance in combinations of the following ways:

1. **Confidence**: Independently derived results for the same task can be used to corroborate each other, yielding a collective result that has a higher confidence of being correct. For example, when studying for an exam, students might separately work out an exercise and then compare answers to increase confidence in their solutions.
2. **Completeness**: Each agent formulates results for whichever subtasks it can (or has been contracted to) accomplish, and these results altogether cover a more complete portion of the overall task. For example, in distributed vehicle monitoring, a more complete map of vehicle movements is possible when agents share their local maps.
3. **Precision**: To refine its own solution, an agent needs to know more about the solutions that others have formulated. For example, in a concurrent engineering application, each agent might separately come up with specifications for part of an artifact, but by sharing these the specifications can be further honed to fit together more precisely.
4. **Timeliness**: Even if an agent could in principle solve a large task alone, solving subtasks in parallel can yield an overall solution faster.

Accruing the benefits of result sharing obviously means that agents need to share results. But making this work is harder than you might think! First of all, agents need to know what to do with shared results: how should an agent assimilate results shared from others in with its own results? Second, given that assimilation might be non-trivial, that communicating large volumes of results can be costly, and that managing many assimilated results incurs overhead, agents should attempt to be as selective as possible about what they exchange. In the remainder of this section, we look at these issues.

3.4.1 Functionally Accurate Cooperation

In task-passing applications like ToH, the separate problem-solving agents are completely accurate in their computations (they have all information and a complete specification for their subtasks) and operate independently. In contrast, agents doing Distributed Vehicle Monitoring (DVM) lack information about what is happening elsewhere that could impact their calculations. As a result, these agents need to cooperate to solve their subtasks, and might formulate tentative results along the way that turn out to be unnecessary. This style of collective problem solving has been termed functionally-accurate (it gets the answer eventually, but with possibly many false starts) and cooperative (it requires iterative exchange) [28].

Functionally-accurate cooperation has been used extensively in distributed problem solving for tasks such as interpretation and design, where agents only discover the details of how their subproblem results interrelate through tentative formulation and iterative exchange. For this method to work well, participating agents need to treat the partial results they have formulated and received as tentative, and therefore might have to entertain and contrast several competing partial hypotheses at once. A variety of agent architectures can support this need; in particular, blackboard architectures (Chapter 2) have often been employed as semi-structured repositories for storing multiple competing hypotheses.

Exchanging tentative partial solutions can impact completeness, precision, and confidence. When agents can synthesize partial solutions into larger (possibly still partial) solutions, more of the overall problem is covered by the solution. When an agent uses a result from another to refine its own solutions, precision is increased. And when an agent combines confidence measures of two (corroborating or competing) partial solutions, the confidence it has in the solutions changes. In general, most distributed problem-solving systems assume similar representations of partial solutions (and their certainty measures) which makes combining them straightforward, although some researchers have considered challenges in crossing between representations, such as combining different uncertainty measurements [47].

In functionally accurate cooperation, the iterative exchange of partial results is expected to lead, eventually, to some agent having enough information to keep moving the overall problem solving forward. Given enough information exchange, therefore, the overall problem will be solved. Of course, without being tempered by some control decisions, this style of cooperative problem solving could incur dramatic amounts of communication overhead and wasted computation. For example, if agents share too many results, a phenomenon called **distraction** can arise: it turns out that they can begin to all gravitate toward doing the same problem-solving actions (synthesizing the same partial results into more complete solutions). That is, they all begin exploring the same part of the search space (Chapter 4). For this reason, limiting communication is usually a good idea, as is giving agents some degree of skepticism in how they assimilate and react to information from others. We address these issues next.

3.4.2 Shared Repositories and Negotiated Search

One strategy for reducing potential flurry of multicast messages is to instead concentrate tentative partial results in a single, shared repository. The blackboard architecture, for example, allows cooperating knowledge sources to exchange results and build off of them by communicating through a common, structured blackboard (Chapter 2).

This strategy has been adopted in a variety of distributed problem-solving approaches, including those for design applications [25, 45]. In essence, using a shared repository can support search through alternative designs, where agents with different design criteria can revise and critique the alternatives. In many ways, this is a distributed constraint satisfaction problem (Chapter 4), but it differs from traditional formulations in a few respects.

Two important differences are: agents are not assumed to know whose constraints might be affected by their design choices, and agents can relax constraints in a pinch. The first difference motivates the use of a shared repository, since agents would not know whom to notify of their decisions (as is assumed in typical DCSP formulations as in Chapter 4). The second difference motivates the need for heuristics to control the distributed search, since at any given time agents might need to choose between improving some solutions, rejecting some solutions, or relaxing expectations (thus making some solutions that were previously considered as rejected now acceptable).

For example, agents engaged in negotiated search [25] have at their disposal a variety of operators for progressing the distributed problem-solving effort: *initiate-solution* (propose a new starting point for a solution); *extend-solution* (revise an already existing partial solution); *critique-solution* (provide feedback on the viability of an already existing partial solution); and *relax-solution-requirement* (change local requirements for solution acceptability). At any given time, an agent needs to decide which of these operators to apply, and where. While a systematic exploration of the space can be considered (Chapter 4), the problem domains for negotiated search are typically complex enough that heuristic guidance is preferred. Heuristic measures for when to invoke operators (such as invoking the *relax-solution-requirement* operator when lack of progress is detected) and on what (such as relaxing requirements corresponding to the most constrained component) are generally application-specific.

3.4.3 Distributed Constrained Heuristic Search

Constraint satisfaction problems in distributed environments also arise due to contention for resources. Rather than assuming a shared repository for tentative partial solutions, a search strategy that has been gainfully employed for distributed resource allocation problems has been to associate an "agent" with each resource, and have that agent process the contending demands for the resource. One form that this strategy takes is so-called market-oriented programming [44] where associated with resources are auctions that support the search for equilibria in which resources are allocated efficiently. Market mechanisms are covered in detail in Chapter 5.

A second form that this strategy takes is to allow resources to compute their aggregate demands, which then the competing agents can take into account as they attack their constraint-satisfaction problem. For example, distributed constrained heuristic search (DCHS) uses aggregate demand to inform a heuristic search for solving a distributed constraint satisfaction problem [43]. The idea is that more informed search decisions decrease wasted backtracking effort, and that constraint satisfaction heuristics such as variable and value ordering can be gainfully employed in a distributed environment.

DCHS works as follows (Figure 3.4):

1. An agent begins with a problem state comprised of a problem topology (the tasks to do and their relationships including constraints).
2. An agent propagates constraints within its state; it backtracks if an inconsistency is detected. Otherwise, it determines what resources it requires for what time intervals and computes a demand profile for those resources.
3. If the system is just beginning, or if the demand profiles differ from previous profiles, an agent sends the profile(s) to the resource(s).
4. A resource computes aggregate demand and informs the agents making the demands.
5. An agent uses the aggregate demands to order its variables (resource-and-time-interval pairs) and order the activities that it might assign to the highest-demand pair. It identifies a preferred resource/time-interval/activity assignment.
6. An agent requests that the resource reserve the interval for it.
7. The resource in turn grants the reservation if possible and updates the resource schedule. Otherwise the request is denied.
8. An agent processes the response from the resource. If the reservation is granted, the agent goes to step 2 (to propagate the effects of concretely scheduling the activity). If the reservation is not granted, the agent attempts another reservation, going to step 6.

This view of the search strategy, while simplified, highlights the use of resources being contended for to focus communication, and of an exchange of information that tends to decrease the amount of backtracking. That is, by giving agents an opportunity to settle the "difficult" contention issues first, much useless work is avoided in settling the easier issues and then discovering that these fail to allow the hard issues to be settled.

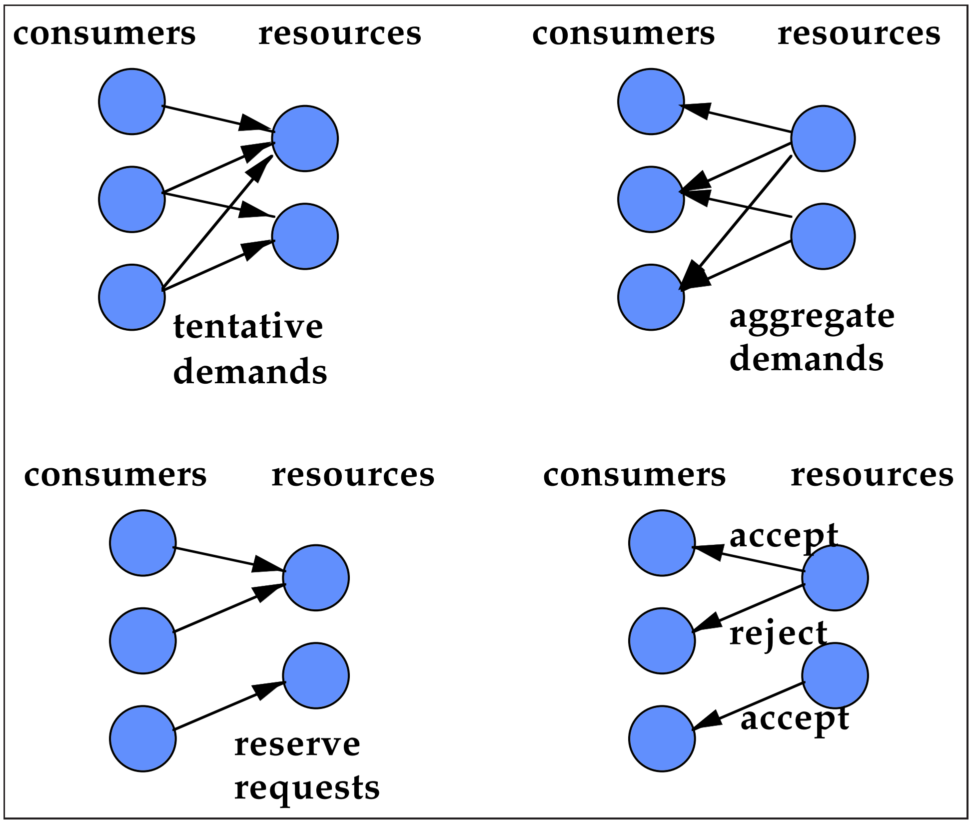

Figure 3.4 DCHS steps.

3.4.4 Organizational Structuring

When a shared repository cannot be supported or when problem-solving is not tantamount to resource scheduling, an alternative strategy for reducing communication is to exploit the task decomposition structure, to the extent that it is known. In a distributed design problem, for example, it makes sense to have designers working on components that must "connect" speak with each other more frequently than they speak with designers working on more remote parts of the design (of course, physical proximity might be only one heuristic!). Or, in a DVM task, agents monitoring neighboring parts of the space should communicate when their maps show activity at or near their mutual boundary. The notion is that agents have general roles to play in the collective effort, and by using knowledge of these roles the agents can make better interaction decisions.

This notion can be explicitly manifested in an organizational structure, which defines roles, responsibilities, and preferences for the agents within a cooperative society, and thus in turn defines control and communication patterns between them.

From a global view, the organizational structure associates with each agent the types of tasks that it can do, and usually some prioritization over the types such that an agent that currently could do any of a number of tasks can identify the most important tasks as part of its organizational role. Allowing prioritization allows the structure to permit overlapping responsibilities (to increase the chances of success despite the loss of some of the agents) while still differentiating agents based on their primary roles.

Since each agent has responsibilities, it is important that an agent be informed of partial results that could influence how it carries out its responsibilities. More importantly, agents need not be told of results that could not affect their actions, and this can be determined based on the organizational structure. Thus, an organizational structure provides the basis for deciding who might potentially be interested in a partial result. It also can dictate the degree to which an agent should believe and act on (versus remain skeptical about) a received result.

While an organizational structure needs to be coherent from an overall perspective, it is important to note that, as in human organizations, an agent only needs to be aware of its local portion of the structure: what it is supposed to be doing (and how to decide what to do when it has choices), who to send what kinds of information to, who to accept what kinds of information from and how strongly to react to that information, etc. For practical purposes, therefore, organizational structures are usually implemented in terms of stored pattern-response rules: when a partial result that matches the pattern is generated/received, then the response actions are taken (to transmit the partial result to a particular agent, or to act on it locally, or to decrement its importance, etc.). Note that a single partial result could trigger multiple actions.

Finally, we have briefly mentioned that an organizational structure can be founded upon the problem decomposition structure, such as for the DSNE problem where agents would be made aware of which other agents are responsible for neighboring areas so that partial results that matched the overlapping regions of interest would be shared. The design of organizational structures for multi- agent systems, however, is generally a complex search problem in its own right. The search can be conducted in a bottom-up distributed manner, where boundaries between the roles of agents can be determined as the problem instance is initialized [5] or as problem solving progresses [19, 35], where adjustments to the structure can be based on reacting to performance inefficiencies of the current structure. In some cases, the organizational structure can be equated to a priority order for a distributed constraint satisfaction problem, and the agents are trying to discover an effective ordering to converge on a solution efficiently (see Chapter 4).

Alternatively, organizational structuring can be viewed as a top-down design problem, where the space of alternative designs can be selectively explored and candidate designs can be evaluated prior to their implementation [3, 34, 40]. The use of computational techniques to study, and prescribe, organizational structures is covered in Chapter 7.

3.4.5 Communication Strategies

Organization structures, or similar knowledge, can provide static guidelines about who is generally interested in what results. But this ignores timing issues. When deciding whether to send a result, an agent really wants to know whether the potential recipient is likely to be interested in the result now (or soon). Sending a result that is potentially useful but that turns out to not be at best clutters up the memory of the recipient, and at worst can distract the recipient away from the useful work that it otherwise would have done. On the other hand, refraining from sending a result for fear of these negative consequences can lead to delays in the pursuit of worthwhile results and even to the failure of the system to converge on reasonable solutions at all because some links in the solution chain were broken.

When cluttering memory is not terrible and when distracting garden paths are short, then the communication strategy can simply be to send all partial results. On the other hand, when it is likely that an exchange of a partial result will lead a subset of agents into redundant exploration of a part of the solution space, it is better to refrain, and only send a partial result when the agent that generated it has completed everything that it can do with it locally. For example, in a distributed theorem-proving problem, an agent might work forward through a number of resolutions toward the sentence to prove, and might transmit the final resolvent that it has formed when it could progress no further.

Between the extremes of sending everything and sending only locally complete results are a variety of gradations [7], including sending a small partial result early on (to potentially spur the recipient into pursuing useful related results earlier). For example, in the DVM problem, agents in neighboring regions need to agree when they map vehicles from one region to the other. Rather than waiting until it forms its own local map before telling its neighbor, an agent can send a preliminary piece of its map near the boundary early on, to stimulate its neighbor into forming a complementary map (or determining that no such map is possible and that the first agent is working down a worthless interpretation path).

So far, we have concentrated on how agents decide when and with whom to voluntarily share results. But the decision could clearly be reversed: agents could only send results when requested. Just like the choice between announcing tasks versus announcing availability in the Contract Net depends on which is more scarce, the same holds true in result sharing. When the space of possible interesting results is large compared to the actual results that are generated, then communicating results makes sense. But when the space of results formed is large and only few are really needed by others, then sending requests (or more generally, goals) to others makes more sense. This strategy has been explored in the DVM problem [3], as well as in distributed theorem proving [15, 31]. For example, in DARES [31], when a theorem proving agent would fail to make progress, it would request to

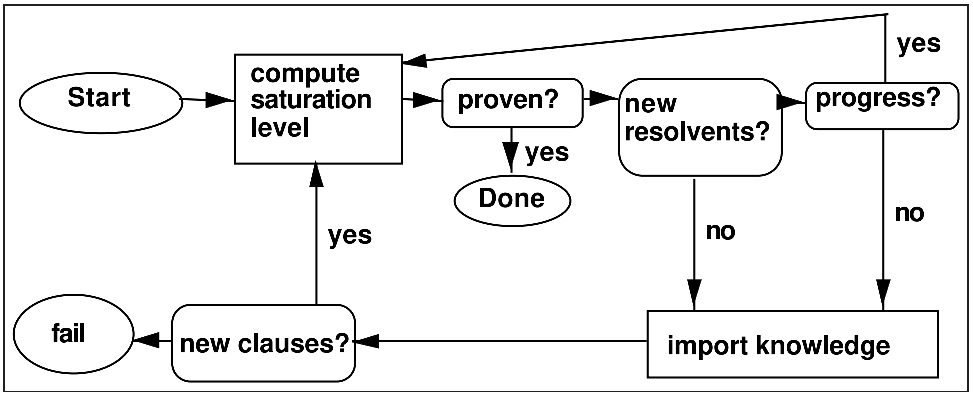

Figure 3.5 DARES agent control flow.

import clauses from other such agents, where the set of desired literals would be heuristically chosen (Figure 3.5).

It is also important to consider the delays in iterative exchange compared to a blind inundation of information. A request followed by a reply incurs two communication delays, compared to the voluntary sharing of an unrequested result. But sharing too many unrequested results can introduce substantial overhead. Clearly, there is a tradeoff between reducing information exchanged by iterative messaging versus reducing delay in having the needed information reach its destination by sending many messages at the same time. Sen, for example, has looked at this in the context of distributed meeting scheduling [38]. Our experience as human meeting schedulers tells us that finding a meeting time could involve a series of proposals of specific times until one is acceptable, or it could involve having the participants send all of their available times at the outset. Most typically, however, practical considerations leave us somewhere between these extremes, sending several options at each iteration.

Finally, the communication strategies outlined have assumed that messages are assured of getting through. If messages get lost, then results (or requests for results) will not get through. But since agents do not necessarily expect messages from each other, a potential recipient will be unable to determine whether or not messages have been lost. One solution to this is to require that messages be acknowledged, and that an agent sending a message will periodically repeat the message (sometimes called "murmuring") until it gets an acknowledgment [29]. Or, a less obtrusive but more uncertain method is for the sending agent to predict how the message will affect the recipient, and to assume the message made it through when the predicted change of behavior is observed (see discussion of plan recognition in Subsection 7.4).

3.4.6 Task Structures

Up to this point, we have made intuitive appeals to why agents might need to communicate results. The TAEMS work of Decker and Lesser has investigated this question much more concretely [6]. In their model, an agent's local problem solving can have non-local effects on the activity of other agents. Perhaps it is supplying a result that another agent must have to *enable* its problem-solving tasks. Or the result might *facilitate* the activities of the recipient, allowing it to generate better results and/or generate results faster. The opposites of these (*inhibit* and *hinder*, respectively) are among the other possible relationships.

By representing the problem decomposition structure explicitly, and capturing within it these kinds of task relationships, we can employ a variety of coordination mechanisms. For example, an agent that provides an enabling result to another can use the task structure representation to detect this relationship, and can then bias its processing to provide this result earlier. In fact, it can use models of task quality versus time curves to make commitments to the recipient as to when it will generate a result with sufficiently high quality. In situations where there are complex networks of non-local task interrelationships, decisions of this kind of course get more difficult. Ultimately, relatively static organizational structures, relationships, and communication strategies can only go so far. Going farther means that the problem-solving agents need to analyze their current situation and construct plans for how they should interact to solve their problems.

3.5 Distributed Planning

In many respects, distributed planning can be thought of simply as a specialization of distributed problem solving, where the problem being solved is to design a plan. But because of the particular features of planning problems, it is generally useful to consider techniques that are particularly suited to planning.

Distributed planning is something of an ambiguous term, because it is unclear exactly what is "distributed." It could be that the operative issue is that, as a consequence of planning, a plan is formulated that can be distributed among a variety of execution systems. Alternatively, the operative issue could be that the planning process should be distributed, whether or not the resulting plan(s) can be. Or perhaps both issues are of interest. In this section, we consider both distributed plans and distributed plan formation as options; we of course skip over the case where neither holds (since that is traditional centralized planning) and consider where one or both of these distributions exists.

3.5.1 Centralized Planning for Distributed Plans

Plans that are to be executed in a distributed fashion can nonetheless be formulated in a centralized manner. For example, a partial order planner can generate plans where there need not be a strict ordering between some actions, and in fact where those actions can be executed in parallel. A centralized coordinator agent with such a plan can break it into separate threads, possibly with some synchronization actions. These separate plan pieces can be passed (using task-passing technology) to agents that can execute them. If followed suitably, and under assumptions of correctness of knowledge and predictability of the world, the agents operating in parallel achieve a state of the world consistent with the goals of the plan.

Let us consider this process more algorithmically. It involves:

1. Given a goal description, a set of operators, and an initial state description, generate a partial order plan. When possible, bias the search to find a plan in which the steps have few ordering constraints among them.
2. Decompose the plan into subplans such that ordering relationships between steps tend to be concentrated within subplans and minimized across subplans. [26].
3. Insert synchronization (typically, communication) actions into subplans.
4. Allocate subplans to agents using task-passing mechanisms. If failure, return to previous steps (decompose differently, or generate a different partial order plan, ...). If success, insert remaining bindings into subplans (such as binding names of agents to send synchronization messages to).
5. Initiate plan execution, and optionally monitor progress (synthesize feedback from agents to ensure complete execution, for example).

Notice that this algorithm is just a specialization of the decompose-allocate-execute-synthesize algorithm used in task passing. The specific issues of decomposition and allocation that are involved in planning give it a special flavor. Essentially, the objective is to find, of all the possible plans that accomplish the goal, the plan that can be decomposed and distributed most effectively. But since the availability of agents for the subplans is not easy to determine without first having devised the subplans, it is not certain that the most decomposable and distributable plan can be allocated in any current context.

Moreover, the communication infrastructure can have a big impact on the degree to which plans should be decomposed and distributed. As an extreme, if the distributed plans require synchronization and if the communication channels are slow or undependable, then it might be better to form a more efficient centralized plan. The monetary and/or time costs of distributing and synchronizing plans should thus be taken into account. In practical terms, what this usually means is that there is some minimal subplan size smaller than which it does not make sense to decompose a plan. In loosely-coupled networks, this leads to systems with fewer agents each accomplishing larger tasks, while in tightly-connected (or even shared-memory) systems the degree of decomposition and parallelism can be increased.

3.5.2 Distributed Planning for Centralized Plans

Formulating a complex plan might require collaboration among a variety of **cooperative planning** specialists, just like generating the solution to any complex problem would. Thus, for complex planning in fields such as manufacturing and logistics, the process of planning could well be distributed among numerous agents, each of which contributes pieces to the plan, until an overarching plan is created.

Parallels to task-sharing and result-sharing problem solving are appropriate in this context. The overall problem-formulation task can be thought of as being decomposed and distributed among various planning specialists, each of which might then proceed to generate its portion of the plan. For some types of problems, the interactions among the planning specialists might be through the exchange of a partially-specified plan. For example, this model has been used in the manufacturing domain, where a general-purpose planner has been coupled with specialist planners for geometric reasoning and fixturing [21]. In this application, the geometric specialist considers the shape of a part to be machined, and generates an abstract plan as an ordering over the geometric features to put into the part. The general-purpose planner then uses these ordering constraints to plan machining operations, and the augmented plan is passed on to the fixture specialist, which ensures that the operations can be carried out in order (that the part can be held for each operation, given that as each operation is done the shape of the part can become increasingly irregular). If any of these planners cannot perform its planning subtask with the partially- constructed plan, they can backtrack and try other choices (See Chapter 4 on DCSPs). Similar techniques have been used for planning in domains such as mission planning for unmanned vehicles [7] and for logistics planning [46].

The more asynchronous activity on the part of planning problem-solvers that is characteristic of most distributed problem-solving systems can also be achieved through the use of result sharing. Rather than pass around a single plan that is elaborated and passed on (or discovered to be a deadend and passed back), a result-sharing approach would have each of the planning agents generate a partial plan in parallel and then share and merge these to converge on a complete plan in a negotiated search mode. For example, in the domain of communication networks, localized agents can tentatively allocate network connections to particular circuits and share these tentative allocations with neighbors [2]. When inconsistent allocations are noticed, some agents try other allocations, and the process continues until a consistent set of allocations have been found. In this example, result-sharing amounts to a distributed constraint satisfaction search, with the usual concerns of completeness and termination (See Chapter 4 on DCSPs).

3.5.3 Distributed Planning for Distributed Plans

The most challenging version of distributed planning is when both the planning process and its results are intended to be distributed. In this case, it might be unnecessary to ever have a multi-agent plan represented in its entirety anywhere in the system, and yet the distributed pieces of the plan should be compatible, which at a minimum means that the agents should not conflict with each other when executing the plans, and preferably should help each other achieve their plans when it would be rational to do so (e.g. when a helping agent is no worse off for its efforts).

The literature on this kind of distributed planning is relatively rich and varied. In this chapter, we will hit a few of the many possible techniques that can be useful.

Plan Merging

We begin by considering the problem of having multiple agents formulate plans for themselves as individuals, and then having to ensure that their separate plans can be executed without conflict. Assume that the assignment of goals to agents has been done, either through task-sharing techniques, or because of the inherent distributivity of the application domain (such as in a distributed delivery (DD) task, where different agents are contacted by users to provide a delivery service). Now the challenge is to identify and resolve potential conflicts.

We begin by considering a centralized plan coordination approach. Let us say that an agent collects together these individual plans. It then has to analyze the plans to discover what sequences of actions might lead to conflicts, and to modify the plans to remove the conflicts. In general, the former problem amounts to a *reachability analysis* – given a set of possible initial states, and a set of action sequences that can be executed asynchronously, enumerate all possible states of the world that can be reached. Of these, then, find the subset of worlds to avoid, and insert constraints on the sequences to eliminate them.

In general, enumerating the reachable state space can be intractable, so strategies for keeping this search reasonable are needed. From the planning literature, many assumptions about the limited effects of actions and minimal interdependence between agents' goals can be used to reduce the search. We will look at one way of doing this, adapted from Georgeff [16] next.

As is traditional, assume that the agents know the possible initial states of the world, and each agent builds a totally-ordered plan using any planning technology. The plan is comprised of actions a_1 through a_n, such that a_1 is applicable to any of the initial states, and a_i is applicable in all states that could arise after action a_{i-1}. The state arising after a_n satisfies the agent's goal.

We represent an action as a **STRIPS operator**, with preconditions that must hold for the action to take place, effects that the action has (where features of the world not mentioned in the effects are assumed unaffected), and "during" conditions to indicate changes to the world that occur only during the action. The STRIPS assumption simplifies the analysis for interactions by allowing us to avoid having to search through all possible interleavings of actions; it is enough to identify specific actions that interact with other specific actions, since the effects of any sequence is just the combined effects of the sequence's actions.

The merging method thus proceeds as follows. Given the plans of several agents (where each is assume to be a correct individual plan), the method begins by analyzing for interactions between pairs of actions to be taken by different agents. Arbitrarily, let us say we are considering the actions a_i and b_j are the next to be executed by agents A and B, respectively, having arrived at this point through the asynchronous execution of plans by A and B. Actions a_i and b_j can be executed in parallel if the preconditions, during conditions, and effects of each are satisfiable at the same time as any of those conditions of the other action. If this is the case, then the actions can commute, and are essentially independent. If this is not the case, then it might still be possible for both actions to be taken but in a stricter order. If the situation before either action is taken, modified by the effects of a_i, can satisfy the preconditions of b_j, then a_i can precede b_j. It is also possible for b_j to precede a_i. If neither can precede the other, then the actions conflict.

From the interaction analysis, the set of unsafe situations can be identified. Clearly, it is unsafe to begin both a_i and b_j if they do not commute. It is also unsafe to begin a_i before b_j unless a_i has precedence over b_j. Finally, we can propagate these unsafe interactions to neighboring situations:

- the situation of beginning a_i and b_j is unsafe if either of its successor situations is unsafe;
- the situation of beginning a_i and ending b_j is unsafe if the situation of ending a_i and ending b_j is unsafe;
- the situation of ending a_i and ending b_j is unsafe if both of its successor states are unsafe.

To keep this safety analysis tractable, actions that commute with all others can be dropped from consideration. Given a loosely-coupled multi-agent system, where agents mostly bring their own resources and capabilities to bear and thus have few opportunities to conflict, dropping commuting actions would reduce the agents' plans to relatively short sequences. From these simplified sequences, then, the process can find the space of unsafe interactions by considering the (exponential) number of interleavings. And, finally, given the discovered unsafe interactions, synchronization actions can be added to the plans to force some agents to suspend activities during regions of their plans that could conflict with others' ongoing actions, until those others release the waiting agents.

Plan synchronization need not be accomplished strictly through communication only. Using messages as signals allows agents to synchronize based on the completion of events rather than reaching specific time points. But many applications have temporal features for goals. Manufacturing systems might have deadlines for fabricating an artifact, or delivery systems might have deadlines for dropping off objects. For these kinds of applications, where temporal predictions for individual tasks are fundamentally important, the formulation of distributed plans can be based on scheduling activities during fixed time intervals. Thus, in these kinds of systems, the individual planners can formulate a desired schedule of activities assuming independence, and then plan coordination requires that the agents search for revisions to their schedules to find non-conflicting times for their activities (which can be accomplished by DCHS (see 3.4.3)). More importantly, different tasks that the agents pursue might be related in a precedence ordering (e.g. a particular article needs to be dropped off before another one can be picked up). Satisfying these constraints, along with deadlines and resource limitation constraints, turns the search for a workable collective schedule into a distributed constraint satisfaction problem (see Chapter 4).

A host of approaches to dealing with more complex forms of this problem exist, but are beyond the scope of this chapter. We give the flavor of a few of these to illustrate some of the possibilities. When there are uncertainties about the time needs of tasks, or of the possibility of arrival of new tasks, the distributed scheduling problem requires mechanisms to maximize expected performance and to make forecasts about future activities [30]. When there might not be feasible schedules to satisfy all agents, issues arise about how agents should decide which plans to combine to maximize their global performance [12]. More complex representations of reactive plans and techniques for coordinating them based on model-checking and Petri-net-based mechanisms have also been explored [20, 27, 37].

Iterative Plan Formation

Plan merging is a powerful technique for increasing parallelism in the planning process as well as during execution. The synchronization and scheduling algorithms outlined above can be carried out in centralized and decentralized ways, where the flow is generally that of (1) assign goals to agents; (2) agents formulate local plans; (3) local plans are exchanged and combined; (4) messaging and/or timing commitments are imposed to resolve negative plan interactions. The parallels between this method of planning and the task-sharing style of distributed problem-solving should be obvious. But just as we discovered in distributed problem solving, not all problems are like the Tower of Hanoi; sometimes, local decisions are dependent on the decisions of others. This raises the question of the degree to which local plans should be formulated with an eye on the coordination issues, rather than as if the agent could work alone.

One way of tempering proposed local plans based on global constraints is to require agents to search through larger spaces of plans rather than each proposing a single specific plan. Thus, each agent might construct the set of *all* feasible plans for accomplishing its own goals. The distributed planning process then consists of a search through how subsets of agents' plans can fit together.

Ephrati and Rosenschein [11] have developed a **plan combination search** approach for doing this kind of search, where the emphasis is on beginning with encompassing sets of possible plans and refining these to converge on a nearly optimal subset. They avoid commitment to sequences of actions by specifying sets of propositions that hold as a result of action sequences instead. The agents engage in the search by proposing, given a particular set of propositions about the world, the changes to that set that they each can make with a single action from their plans. These are all considered so as to generate candidate next sets of propositions about the world, and these candidates can be ranked using an A* heuristic (where each agent can use its plans to estimate the cost from the candidate to completing its own goals). The best candidate is chosen and the process repeats, until no agent wants to propose any changes (each has accomplished its goal).

Note that, depending on the more global movement of the plan, an agent will be narrowing down the plan it expects to use to accomplish its own private goals. Thus, agents are simultaneously searching for which local plan to use as well as for synchronization constraints on their actions (since in many cases the optimal step forward in the set of achieved propositions might omit the possible contributions of an agent, meaning that the agent should not perform an action at the time).

An alternative to this approach instead exploits the hierarchical structure of a plan space to perform **distributed hierarchical planning**. By now, hierarchical planning is well-established in the AI literature. It has substantial advantages (as exemplified in the ToH problem) in that some interactions can be worked out in more abstract plan spaces, thereby pruning away large portions of the more detailed spaces. In the distributed planning literature, the advantages of hierarchical planning were first investigated by Corkill.

Corkill's work considered a distributed version of Sacerdoti's NOAH system. He added a "decompose plan" critic that would look for conjunctive goals to distribute. Thus, in a blocks-world problem (the infamous Sussman's Anomaly, for instance), the initial plan refinement of (AND (ON A B) (ON B C)) leads to a plan network with two concurrent paths, one for each of the conjuncts. The decompose-plan critic gives a copy of the plan network to a second agent, where each of the two agents now represents the goal it is to achieve as well as a parallel node in the network that represents a model of the other agent's plan. Then the agents proceed refine their abstract plans to successively detailed levels. As an agent does so, it can communicate with the other one about the changes that it expects to make to the world state, so that each can separately detect conflicts. For example, when an agent learns that the other is going to make block B not clear (it does not know the details of how) it can determine that this will interfere with stacking B on C, and can ask the first agent to WAIT on the action that causes that change until it has received permission to go on. This process can continue until a synchronized

1. Initialize the current-abstraction-level to the most abstract level.
2. Agents exchange descriptions of the plans and goals of interest at the current level.
3. Remove plans with no potential conflicts. If the set is empty, then done; otherwise determine whether to resolve conflicts at the current level or at a deeper level.
4. If conflicts are to be resolved at a deeper level, set the current level to the next deeper level and set the plans/goals of interest to the refinements of the plans with potential conflicts. Go to step 2.
5. If conflicts are to be resolved at this level:
   (a) Agents form a total order. Top agent is the current superior.
   (b) Current superior sends down its plan to the others.
   (c) Other agents change their plans to work properly with those of the current superior. Before confirming with the current superior, an agent also doublechecks that its plan changes do not conflict with previous superiors.
   (d) Once no further changes are needed among the plans of the inferior agents, the current superior becomes a previous superior and the next agent in the total order becomes the superior. Return to step (b). If there is no next agent, then the protocol terminates and the agents have coordinated their plans.

Figure 3.6 Hierarchical Behavior-Space Search Algorithm.

set of detailed plans are formed.

A variation on the hierarchical distributed planning approach is to allow each agent to represent its local planned behaviors at multiple levels of abstraction, any of which can suffice to resolve all conflicts. In this **hierarchical behavior-space search** approach to distributed planning, the outer loop of the protocol identifies a particular level of abstraction to work with, and whether conflicts should be resolved at this level or passed to more detailed levels. The inner loop of the protocol conducts what can be thought of as a distributed constraint satisfaction search to resolve the conflicts. Because the plans at various abstraction levels dictate the behaviors of agents to a particular degree, this approach has been characterized as search through hierarchical behavior space [9]. The algorithm is presented in Figure 3.6. Provided that there are finite abstraction levels and that agents are restricted in the changes to their plans that they can make such that they cannot get into cyclic plan generation patterns, the above protocol is assured to terminate. A challenge lies in the outer loop, in terms of deciding whether to resolve at an abstract level or to go deeper. The advantage of resolving a conflict at an abstract level is that it reduces the amount of search, and thus yields coordinated plans with less time and messaging. The disadvantage is that the coordination constraints at an abstract level might impose unnecessary limits on more detailed actions. At more detailed levels, the precise interaction problems can be recognized and resolved, while at

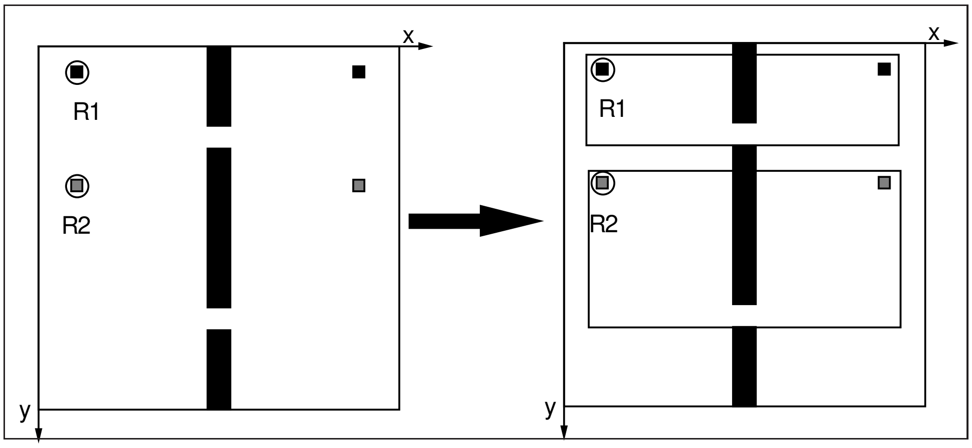

Figure 3.7 An organizational solution.

abstract levels more inefficient coordination solutions might work. The tradeoffs between long-term, simple, but possibly inefficient coordination decisions versus more responsive but complex runtime coordination decisions is invariably domain-dependent. The goal is to have mechanisms that support the broad spectrum of possibilities.

As a concrete example of this approach, consider the DD problem of two delivery robots making repeated deliveries between two rooms as in Figure 3.7 (left side). Since R1 always delivers between the upper locations, and R2 between the lower ones, the robots could each inform the other about where they might be into the indefinite future (between the locations, passing through the closest door). Their long-term delivery behaviors potentially conflict over that door, so the robots can choose either to search in greater detail around the door, or to eliminate the conflict at the abstract behavior level. The latter leads to a strategy for coordinating that statically assigns the doors. This leads to the permanent allocation of spatial regions shown in Figure 3.7 (right side), where R2 is always running around the long way. This "organizational" solution avoids any need for further coordination, but it can be inefficient, especially when R1 is not using its door, since R2 is still taking the long route. If they choose to examine their behaviors in more detail, they can find other solutions. If they consider a particular delivery, for example, R1 and R2 might consider their time/space needs, and identify that pushing their activities apart in space or time would suffice (Figure 3.8, top). With temporal resolution, R2 waits until R1 is done before beginning to move through the central door. Or the robots could use information from this more abstract level to further focus communication on exchanging more detailed information about the trouble spots. They could resolve the potential conflict at an intermediate level of abstraction; temporal resolution has R2 begin once R1 has cleared the door (Figure 3.8, middle). Or they could communicate more details (Figure 3.8, bottom), where now R2 moves at the same time as R1, and stops just before the door to let R1 pass through first. Clearly, this last instance of coordination is crispest, but it is also the most expensive to arrive at and the least tolerant of failure, since the robots have less distance between them in general, so less room to avoid collisions if they deviate from planned paths.

Of course, there are even more strategies for coordination even in a simple domain such as the distributed delivery task. One interesting strategy is for the robots to move up a level to see their tasks as part of a single, team task. By doing so, they can recognize alternative decompositions. For example, rather than decompose by items to deliver, they could decompose by spatial areas, leading to a solution where one robot picks up items at the source locations and drops them off at the doorway, and the other picks up at the doorway and delivers to the final destinations. By seeing themselves as part of one team, the agents can coordinate to their mutual benefit (they can cooperate) by searching through an enlarged behavior space.

Negotiation in Distributed Planning

In the above, we considered how agents can determine that conflicts exist between their plans and how to impose constraints on (usually when they take) their actions to avoid conflict. Sometimes, determining which agent should wait for another is fairly random and arbitrary. Exceptions, however, exist. A large amount of work in negotiation (see Chapter 2) is concerned with these issues, so we only touch on them briefly here.

Sometimes the selection of the agent that should revise its local plans is based on models of the possibilities open to the agents. For example, Steeb and Cammarata, in the air-traffic control domain, were concerned with which of the various aircraft should alter direction to decrease potentially dangerous congestion. Their agents exchanged descriptions indicating their flexibility, and the agent that had the most other options was asked to change its plan, in an early DAI application of the least-constrained agent heuristic (see Subsection 3.4.3 and Chapter 4 on DCSPs).

Of course, these and other negotiation mechanisms for resolving goals presume that agents are honest about the importance of their goals and their options for how to achieve them. Issues of how to encourage self-interested agents to be honest are covered elsewhere in this book (see Chapter 5). However, clearly agents have self-interest in looking for opportunities to work to their mutual benefit by accomplishing goals that each other need. However, although the space of possible conflicts between agents is large, the space of possible cooperative activities can be even larger, and introduces a variety of utility assessments. That is, while it can be argued that agents that have conflicts always should resolve them (since the system might collapse if conflicts are manifested), the case for potential cooperative actions is not so strong. Usually, cooperation is "better," but the degree to which agents benefit might not outweigh the efforts they expend in finding cooperative opportunities. Thus, work on distributed planning that focuses on

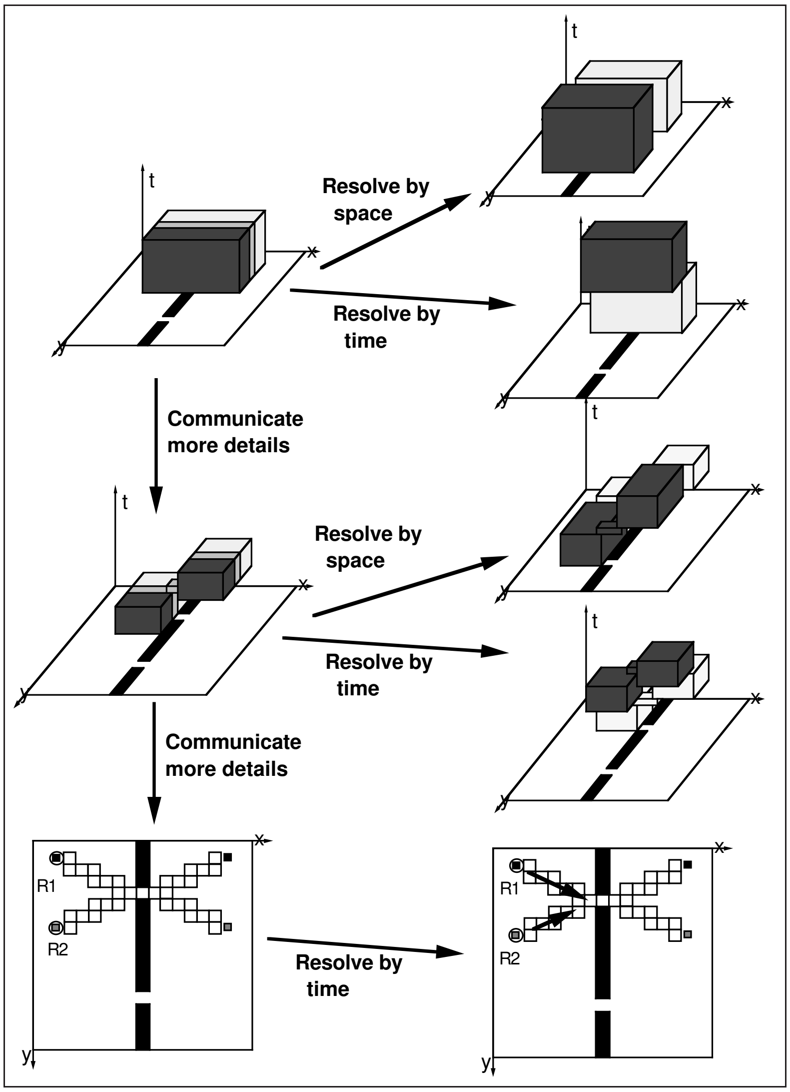

Figure 3.8 Alternative levels of abstraction.

planning for mutually beneficial actions even though they were not strictly necessary has been limited to several forays into studies within well-defined boundaries. For example, partial global planning (see Subsection 3.7.3) emphasized a search for generating partial solutions near partial solution boundaries with other agents, so as to provide them with useful focusing information early on (see Subsection 3.4.5 on communication strategies). The work of von Martial [32] concentrated on strategies that agents can use to exploit "favor relations" among their goals, such as accomplishing a goal for another agent while pursuing its own goal.

3.6 Distributed Plan Representations

Distributed problem solving, encompassing distributed planning, generally relies heavily on agents being able to communicate about tasks, solutions, goals, plans, and so on. Of course, much work has gone into low-level networking protocols for interprocess communication in computer science generally, which forms the foundation upon which the particular communication mechanisms for multiagent systems build. At a much higher level, general-purpose protocols for agent interaction have been developed over the years, ranging from the Contract Net protocol which we have already seen to a broader variety of languages based on speech acts, such as KQML and agent-oriented programming (see Chapter 2). With speech-act-based languages, sending a message can be seen as invoking a behavior at the recipient. For example, sending a message of the type "query" might be expected to evoke in the recipient a good-faith effort to generate an answer followed by sending a message of the type "response" back to the sender.

This is all well and good, but what should the query itself look like? And the response? Different kinds of information might be asked about, and talked about, very differently. For this reason, a high-level speech-act-based language usually leaves the definition of the "content" of a message up to the designer. For any application domain, therefore, one or more relevant content languages need to be defined such that agents can understand not only the intent behind a message, but also the content of the message. In general, the definition of content languages is difficult and open-ended. By restricting our considerations to distributed planning, however, there is some hope in developing characteristics of a sharable planning language.

A planning content language needs to satisfy all of the constituencies that would use the plan. If we think of a plan as being comprised of a variety of fields (different kinds of related information), then different combinations of agents will need to access and modify different combinations of fields. In exchanging a plan, the agents need to be able to find the information they need so as to take the actions that they are expected to take in interpreting, modifying, or executing the plan. They also need to know how to change the plan in ways that will be interpreted correctly by other agents and lead to desirable effects.

To date, there are few standards for specifying plans for computer-based agents. Some conventions certainly exist (such as the "STRIPS operator" format [14]), but these are usually useful only within a narrow context. In most distributed planning systems, it is assumed that the agents use identical representations and are built to interpret them in the same ways.

One effort for formulating a more general description of a plan has been undertaken by SRI, in the development of their Cypress system [46]. In a nutshell, Cypress combined existing systems for plan generation and for plan execution. These existing systems were initially written to be stand-alone; Cypress needed to define a language that the two systems could use to exchange plans, despite the fact that what each system did with plans was very different. In their formalism, an **ACT** is composed of the following fields:

- Name – a unique label
- Cue – goals which the ACT is capable of achieving
- Precondition – features of the world state that need to hold for the ACT to be applicable
- Setting – world-state features that are bound to ACT variables
- Resources – resources required by the ACT during execution
- Properties – other properties associated with the ACT
- Comment – documentation information
- Plot – specification of the procedure (partially-ordered sequences of goals/actions) to be executed

Of course, each of these fields in turn needs a content language that can be understood by the relevant agents.

Other efforts have sought planning languages grounded in temporal logics and operational formalisms such as Petri Nets and Graphcet [20, 27, 37]. By appealing to a representation with a well-understood operational interpretation, the planning agents are freed from having to use identical internal representations so long as their interpretations are consistent with the operational semantics.

3.7 Distributed Planning and Execution

Of course, distributed planning does not occur in a vacuum. The product of distributed planning needs to be executed. The relationships between planning and execution are an important topic in AI in general, and the added complexity of coordinating plans only compounds the challenges. In this section, we consider strategies for combining coordination, planning, and execution.

3.7.1 Post-Planning Coordination

The distributed planning approach based on plan merging essentially sequentialized the processes in terms of allowing agents to plan, then coordinating the plans, and then executing them. This is reasonable approach given that the agents individually build plans that are likely to be able to be coordinated, and that the coordinated result is likely to executed successfully. If, during execution, one (or more) plans for agents fail to progress as expected, the coordinated plan set is in danger of failing as a whole.

As in classical planning systems, there are several routes of recourse to this problem. One is **contingency planning**. Each agent formulates not only its expected plan, but also alternative (branches of) plans to respond to possible contingencies that can arise at execution time. These larger plans, with their conditional branches, can then be merged and coordinated. The coordination process of course is more complicated because of the need to consider the various combinations of plan execution threads that could be pursued. By annotating the plan choices with the conditions, a more sophisticated coordination process can ignore combinations of conditional plans whose conditions cannot be satisfied in the same run.

A second means of dealing with dynamics is through monitoring and replanning: Each agent monitors its plan execution, and if there is a deviation it stops all agents' progress, and the plan-coordinate-execute cycle is repeated. Obviously, if this happens frequently, a substantial expenditure of effort for planning and coordination can result. Sometimes, strategies such as repairing the previous plans, or accessing a library of reusable plans [42] can reduce the effort to make it managable.

Significant overhead can of course be saved if a plan deviation can be addressed locally rather than having to require coordination. For example, rather than coordinating sequences of actions, the agents might coordinate their plans at an abstract level. Then, during execution, an agent can replan details without requiring coordination with others so long as its plan revision fits within the coordinated abstract plan. This approach has been taken in the team plan execution work of Kinney and colleagues, for example [22]. The perceptive reader will also recognize in this approach the flavor of organizational structuring and distributed planning in a hierarchical behavior space: so long as it remains within the scope of its roles and responsibilities, an agent can individually decide what is the best way of accomplishing its goals. By moving to coordinate at the most abstract plan level, the process essentially reverses from post-planning to pre-planning coordination.

3.7.2 Pre-Planning Coordination

Before an agent begins planning at all, can coordination be done to ensure that, whatever it plans to do, the agent will be coordinated with others? The answer is of course yes, assuming that the coordination restrictions are acceptable. This was the answer in organizational structuring in distributed problem solving, where an agent could choose to work on any part of the problem so long as it fit within its range of responsibilities.

A variation on this theme is captured in the work on **social laws** [39]. A social law is a prohibition against particular choices of actions in particular contexts. For example, entering an intersection on a red light is prohibited, as might be *not* entering the intersection on a green light. These laws can be derived by working from undesirable states of the world backwards to find combinations of actions that lead to those states, and then imposing restrictions on actions so that the combinations cannot arise. A challenge is to find restrictions that prevent undesirable states without handcuffing agents from achieving states that are acceptable and desirable. When overly constrictive, relaxations of social laws can be made [1].

Alternatively, in domains where conflict avoidance is not a key consideration, it is still possible that agents might mutually benefit if they each prefer to take actions that benefit society as a whole, even if not directly relevant to the agent's goal. For example, in a Distributed Delivery application, it could be that a delivery agent is passing by a location where an object is awaiting pickup by a different agent. The agent passing by could potentially pick up the object and deliver it itself, or deliver it to a location along its route that will be a more convenient pickup point for the other agent. For example, the delivery agents might pass through a "hub" location. The bias toward doing such favors for other agents could be encoded into cooperative state-changing rules [17] that require agents to take such cooperative actions even to their individual detriment, as long as they are not detrimental beyond some threshold.

3.7.3 Interleaved Planning, Coordination, and Execution

More generally, between approaches that assume agents have detailed plans to coordinate and approaches that assume general-purpose coordination policies can apply to all planning situations, lies work that is more flexible about at what point between the most abstract and most detailed plan representations different kinds of coordination should be done. Perhaps the search for the proper level is conducted through a hierarchical protocol, or perhaps it is predefined. In either case, planning and coordination are interleaved with each other, and often with execution as well.

Let us consider a particular example of an approach that assumes that planning and coordination decisions must be continually revisited and revised. The approach we focus on is called **Partial Global Planning** [8].

Task Decomposition – Partial Global Planning starts with the premise that tasks are inherently decomposed – or at least that they could be. Therefore, unlike planning techniques that assume that the overall task to be planned for is known by one agent, which then decomposes the task into subtasks, which themselves might be decomposed, and so on, partial global planning assumes that an agent with a task to plan for might be unaware at the outset as to what tasks (if any) other agents might be planning for, and how (and whether) those tasks might be related to its own as in the DVM task. A fundamental assumption in Partial Global Planning is that no individual agent might be aware of the global task or state, and the purpose of coordination is to allow agents to develop sufficient awareness to accomplish their tasks nonetheless.

Local Plan Formulation – Before an agent can coordinate with others using Partial Global Planning, it must first develop an understanding of what goals it is trying to achieve and what actions it is likely to take to achieve them. Hence, purely reactive agents, which cannot explicitly represent goals that they are trying to achieve and actions to achieve them, cannot gainfully employ Partial Global Planning (or, for that matter, distributed planning at all). Moreover, since most agents will be concurrently concerned with multiple goals (or at least will be able to identify several achievable outcomes that satisfy a desired goal), local plans will most often be uncertain, involving branches of alternative actions depending on the results of previous actions and changes in the environmental context in carrying out the plan.

Local Plan Abstraction – While it is important for an agent to identify alternative courses of action for achieving the same goal in an unpredictable world, the details of the alternatives might be unnecessary as far as the agent's ability to coordinate with others. That is, an agent might have to commit to activities at one level of detail (to supply a result by a particular time) without committing to activities at more detailed levels (specifying how the result will be constructed over time). Abstraction plays a key role in coordination, since coordination that is both correct and computationally efficient requires that agents have models of themselves and others that are only detailed enough to gainfully enhance collective performance. In Partial Global Planning, for example, agents are designed to identify their major plan steps that could be of interest to other agents.

Communication – Since coordination through Partial Global Planning requires agents to identify how they could and should work together, they must somehow communicate about their abstract local plans so as to build models of joint activity. In Partial Global Planning, the knowledge to guide this communication is contained in the **Meta-Level Organization** (MLO). The MLO specifies information and control flows among the agents: Who needs to know the plans of a particular agent, and who has authority to impose new plans on an agent based on having a more global view. The declarative MLO provides a flexible means for controlling the process of coordination.

Partial Global Goal Identification – Due to the inherent decomposition of tasks among agents, the exchange of local plans (and their associated goals) gives agents an opportunity to identify when the goals of one or more agents could be considered subgoals of a single global goal. Because, at any given time, only portions of the global goal might be known to the agents, it is called a partial global goal. Construction of partial global goals is, in fact, an interpretation problem, with a set of operators that attempts to generate an overall interpretation (global goal) that explains the component data (local goals). The kinds of knowledge needed are abstractions of the knowledge needed to synthesize results of the distributed tasks. And, just as interpretations can be ambiguous, so too is it possible that a local goal

1. For the current ordering, rate the individual actions and sum the ratings.
2. For each action, examine the later actions for the same agent and find the most highly-rated one. If it is higher rated, then swap the actions.
3. If the new ordering is more highly rated than the current one, then replace the current ordering with the new one and go to step 2.
4. Return the current ordering.

Figure 3.9 The Algorithm for PGP Plan Step Reordering.

can be seen as contributing to competing partial global goals.

Partial Global Plan Construction and Modification – Local plans that can be seen as contributing to a single partial global goal can be integrated into a partial global plan, which captures the planned concurrent activities (at the abstract plan step level) of the individuals. By analyzing these activities, an agent that has constructed the partial global plan can identify opportunities for improved coordination. In particular, the coordination relationships emphasized in PGP are those of facilitating task achievement of others by performing related tasks earlier, and of avoiding redundant task achievement. PGP uses a simple hill-climbing algorithm, coupled with an evaluation function on ordered actions, to search for an improved (although not necessarily optimal) set of concurrent actions for the partial global plan (see Figure 3.9). The evaluation function sums evaluations of each action, where the evaluation of an action is based on features such as whether the task is unlikely to have been accomplished already by another agent, how long it is expected to take, and on how useful its results will be to others in performing their tasks.

Communication Planning – After reordering the major local plan steps of the participating agents so as to yield a more coordinated plan, an agent must next consider what interactions should take place between agents. In PGP, interactions, in the form of communicating the results of tasks, are also planned. By examining the partial global plan, an agent can determine when a task will be completed by one agent that could be of interest to another agent, and can explicitly plan the communication action to transmit the result. If results need to be synthesized, an agent using PGP will construct a tree of exchanges such that, at the root of the tree, partially synthesized results will be at the same agent which can then construct the complete result (see Figure 3.10).

Acting on Partial Global Plans – Once a partial global plan has been constructed and the concurrent local and communicative actions have been ordered, the collective activities of the agents have been planned. What remains is for these activities to be translated back to the local level so that they can be carried out. In PGP, an agent responds to a change in its partial global plans by modifying the abstract representation of its local plans accordingly. In turn, this modified representation is used by an agent when choosing its next local action, and thus the choice of

1. Initialize the set of partial task results to integrate.
2. While the set contains more than one element:
   (a) For each pair of elements: find the earliest time and agent at which they can be combined.
   (b) For the pair that can be combined earliest: add a new element to the set of partial results for the combination and remove the two elements that were combined.
3. Return the single element in the set.

Figure 3.10 The Algorithm for Planning Communication Actions.

local actions is guided by the abstract local plan, which in turn represents the local component of the planned collective activity.

Ongoing Modification – As agents pursue their plans, their actions or events in the environment might lead to changes in tasks or in choices of actions to accomplish tasks. Sometimes, these changes are so minor that they leave the abstract local plans used for coordination unchanged. At other times, they do cause changes. A challenge in coordination is deciding when the changes in local plans are significant enough to warrant communication and recoordination. The danger in being too sensitive to changes is that an agent that informs others of minor changes can cause a chain reaction of minor changes, where the slight improvement in coordination is more than offset by the effort spent in getting it. On the other hand, being too insensitive can lead to very poor performance, as agents' local activities do not mesh well because each is expecting the other to act according to the partial global plan, which is not being followed very closely anymore. In PGP, a system designer has the ability to specify parametrically the threshold that defines significant temporal deviation from planned activity.

Task Reallocation – In some circumstances, the exogenous task decomposition and allocation might leave agents with disproportionate task loads. Through PGP, agents that exchange abstract models of their activities will be able to detect whether they are overburdened, and candidate agents that are underburdened. By generating and proposing partial global plans that represent others taking over some of its tasks, an agent essentially suggests a contracting relationship among the agents. A recipient has an option of counter proposing by returning a modified partial global plan, and the agents could engage in protracted negotiations. If successful, however, the negotiations will lead to task reallocation among the agents, allowing PGP to be useful even in situations where tasks are quite centralized.

Summary – PGP fills a distributed planning niche, being particularly suited to applications where some uncoordinated activity can be tolerated and overcome, since the agents are individually revisiting and revising their plans midstream, such that the system as a whole might at times (or even through the whole task episode) never settle down into a stable collection of local plans. PGP focuses on dynamically revising plans in cost-effective ways given an uncertain world, rather than on optimizing plans for static and predictable environments. It works well for many tasks, but could be inappropriate for domains such as air-traffic control where guarantees about coordination must be made prior to any execution.

3.7.4 Runtime Plan Coordination Without Communication

While tailored for dynamic domains, PGP still assumes that agents can and will exchange planning information over time to coordinate their actions. In some applications, however, runtime recoordination needs to be done when agents cannot or should not communicate. We briefly touch on plan coordination mechanisms for such circumstances.

One way of coordinated without explicit communication is to allow agents to infer each others plans based on observations. The plan recognition literature focuses on how observed actions can lead to hypotheses about the plans being executed by others. While generally more uncertain than coordination using explicit communication, **observation-based plan coordination** can still achieve high-quality results and, under some circumstances can outperform communication-based distributed planning [18].

Another way of coordinating without explicit communication is to allow agents to make inferences about the choices others are likely to make based on assumptions about their rationality [36] or about how they view the world. For example, if Distributed Delivery agents are going to hand off objects to each other, they might infer that some locations (such as a hub) are more likely to be mutually recognized as good choices. Such solutions to choice problems have been referred to as **focal points** [13].

3.8 Conclusions

Distributed planning has a variety of reasonably well-studied tools and techniques in its repertoire. One of the important challenges to the field is in characterizing these tools and undertanding where and when to apply each. To some extent, the lack of specificity in the term "distributed planning" in terms of whether the process or the product or both of planning is distributed has hampered communication within the field, but more fundamental issues of articulating the foundational assumptions behind different approaches still need to be addressed. Until many of the assumed context and semantics for plans are unveiled, the goal of having heterogeneous plan generation and plan execution agents work together is likely to remain elusive.

The field of distributed problem solving is even more wide open, because the characterization of a "problem" is that much broader. As we have tried to emphasize, distributed plan formation and, in many cases, execution can be thought of as distributed problem solving tasks. Representations and general-purpose strategies for distributed problem solving are thus even more elusive. In this chapter we have characterized basic classes of strategies such as task- sharing and result-sharing. Ultimately, the purpose of any strategy is to share the right information about tasks, capabilities, availabilities, partial results, or whatever so that each agent is doing the best thing that it can for the group at any given time. Of course, exchanging and using the information that renders such choices can itself be costly, and opens the door to misinterpretation that makes matters worse rather than better. All of these considerations factor into the definition and implementation of a distributed problem strategy, but formulating such a strategy still has more "art" to it than we like to see in an engineering discipline.

Acknowledgements: The effort of compiling (and in some cases developing) the ideas in this chapter was supported, in part, by the NSF under PYI award 91-58473, and by DARPA under contract N66001-93-D-0058. I would like to thank my colleagues and my current and former graduate students, who have contributed the partial solutions assembled here.

3.9 Exercises

1. *[Level 1]* The ToH time complexity analysis that reduces the complexity to logarithmic time assumed that the number of levels was a function of the problem size. More realistically, an organization would be developed for a variety of problems, rather than on a case-by-case basis. Assume the number of levels is fixed (and so the ratio between hierarchy levels will vary with the problem size). Now what is the expected time complexity for the ToH in a distributed problem-solving scenario. What does this answer tell you?
2. *[Level 1]* Consider Contract Net without focused addressing (that is, announcements are broadcast).
   (a) Name a real-life example where task announcment makes much more sense than availability announcement. Justify why.
   (b) Now name a real-life example where availability announcement makes much more sense. Justify why.
   (c) Let's say that you are going to build a mechanism that oversees a distributed problem-solving system, and can "switch" it to either a task or availability announcement mode.
      i. Assuming communication costs are negligible, what criteria would you use to switch between modes? Be specific about what you would test.
      ii. If communication costs are high, now what criteria would you use? Be specific about what you would test.

3. *[Level 2/3]* We noted that task announcing can be tricky: If a manager is too fussy about eligibility, it might get no bids, but if it is too open it might have to process too many bids, including those from inferior contractors. Let us say that the manager has n levels of eligibility specifications from which it needs to choose one. Describe how it would make this choice based on a decision-theoretic formulation. How would this formulation change if it needed to consider competition for contractors from other managers?
4. *[Level 2]* A folk theorem in the organization literature is that, in human organizations, task decompositions invariably lead to clear assignments of subtasks to members of the organization. Give an example of where decomposition without look-ahead to available contractors can be detrimental. Give an example where biasing decomposition based on available contractors can instead be detrimental. Finally, give an algorithm for alternating between decomposition and assignment to incrementally formulate a distributed problem-solving system. Is your algorithm assured of yielding an optimal result? Is it complete?
5. *[Level 1]* Consider the pursuit task, with four predators attempting to surround and capture a prey. Define an organizational structure for the predators. What are the roles and responsibilities of each? How does the structure indicate the kinds of communication patterns (if any) that will lead to success?
6. *[Level 2]* In the problem of distributed meeting scheduling, let us say that the chances that a specific meeting time proposal will be accepted is p.
   (a) If each iteration of the scheduling protocol has an agent propose a specific time to the others, what is the probability that the meeting will be scheduled in exactly I iterations? What is the expected number of iterations to schedule the meeting?
   (b) If each iteration instead proposes N specific times, now what is the probability that the meeting will be scheduled in exactly I iterations? What is the expected number of iterations to schedule the meeting? What happens when N approaches 1? How about when N grows very large?
   (c) Based on the above, how would you choose a value for N to use in a distributed meeting scheduling system? What other considerations might need to be taken into account besides a desire to keep the number of iterations low?
7. *[Level 2]* Consider the following simple instance of the distributed delivery task. Robot A is at position α and robot B is at position β. Article X is at position ξ and needs to go to position ψ, and article Y is at position ψ and needs to go to ζ. Positions α, β, ξ, ψ, and ζ are all different.
   (a) Define in STRIPS notation, suitable for Partial Order Planning, simple operators Pickup, Dropoff, PickDrop, and Return, where Pickup moves the robot from its current position to a Pickup position where it then has the article associated with that position; Dropoff moves a robot and an article it holds to a dropoff position where it no longer has the article;

PickDrop combines the two (it drops off its article and picks up another associated with that position); and Return moves a robot back to its original position.

(b) Using these operators, generate the partial order plan with the shortest sequence of plan steps to accomplish the deliveries. Decompose and distribute this plan to the robots for parallel execution, inserting any needed synchronization actions. How does the use of multiple robots affect the plan execution?

(c) Using the operators, generate the partial order plan that, when distributed, will accomplish the deliveries as quickly as possible. Is this the same plan as in the previous part of this problem? Why or why not?

8. *[Level 2]* Given the problem of question 7, include in the operator descriptions conditions that disallow robots to be at the same position at the same time (for example, a robot cannot do a pickup in a location where another is doing a dropoff). Assuming each robot was given the task of delivering a different one of the articles, generate the individual plans and then use the plan merging algorithm to formulate the synchronized plans, including any synchronization actions into the plans. Show your work.

9. *[Level 2]* Consider the problem of question 7. Assume that delivery plans can be decomposed into 3 subplans (pickup, dropoff, and return), and that each of these subplans can further be decomposed into individual plan steps. Furthermore, assume that robots should not occupy the same location at the same time not just at dropoff/pickup points, but throughout their travels. Use the hierarchical protocol to resolve potential conflicts between the robots plans, given a few different layouts of the coordinates for the various positions (that is, where path-crossing is maximized and minimized). What kinds of coordinated plans arise depending on what level of the hierarchy the plans' conflicts are resolved through synchronization?

10. *[Level 2]* Assume that agents in the distributed delivery domain could be given delivery requests at any given time, and operate in a finite, fully shared delivery region. Describe social laws that can assure that no matter what deliveries are asked of them and when, the agents can be assured of avoiding collisions no matter where the pickup and dropoff positions are. You may assume that the world begins in a legal state. In what circumstances would using these laws be very inefficient?

11. *[Level 3]* Assume that distributed delivery robots are in an environment where delivery tasks pop up dynamically. When a delivery needs to be done, the article to be delivered announces that it needs to be delivered, and delivery agents within a particular distance from the article hear the announcement.

(a) Assume that the distance from which articles can be heard is small. What characteristics would an organizational structure among the delivery agents have to have to minimize the deliveries that might be overlooked?

(b) Assume that the distance is instead large. Would an organizational structure be beneficial anyway? Justify your answer.

(c) As they become aware of deliveries to be done, delivery agents try to incorporate those into their current delivery plans. But the dynamic nature of the domain means that these plans are undergoing evolution. Under what assumptions would partial global planning be a good approach for coordinating the agents in this case?

(d) Assume you are using partial global planning for coordination in this problem. What would you believe would be a good planning level for the agents to communicate and coordinate their plans? How would the agents determine whether they were working on related plans? How would they use this view to change their local plans? Would a hill-climbing strategy work well for this?

3.10 References

1. W. Briggs and D.J. Cook. Flexible social laws. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, August 1995.
2. Susan E. Conry, Kazuhiro Kuwabara, Victor R. Lesser, and Robert A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1462-1477, Nov. 1991.
3. Daniel D. Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*. PhD thesis, University of Massachusetts, December 1982.
4. Randall Davis and Reid Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63-109, 1983.
5. Keith Decker and Victor Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 210-216, July 1993.
6. Keith Decker and Victor Lesser. Designing a family of coordination mechanisms. In *Proceedings of the First International Conf. on Multi-Agent Systems (ICMAS-95)*, pages 73-80, June 1995.
7. Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Cooperation Through Communication in a Distributed Problem Solving Network. In M. Huhns, editor, *Distributed Artificial Intelligence*, Chapter 2, Pitman 1987.
8. Edmund H. Durfee. *Coordination of Distributed Problem Solvers*, Kluwer Academic Press, Boston 1988.
9. Edmund H. Durfee and Thomas A. Montgomery. Coordination as Distributed Search in a Hierarchical Behavior Space. *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Distributed Artificial Intelligence, SMC-21(6):1363-1378, November 1991.
10. Edmund H. Durfee, Patrick G. Kenny, and Karl C. Kluge. Integrated Premission Planning and Execution for Unmanned Ground Vehicles. In *Proceedings of the First*

International Conference on Autonomous Agents, pages 348-354, February 1997.

11. Eithan Ephrati and Jeffrey S. Rosenschein. Divide and conquer in multi-agent planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 375-380, July 1994.
12. Eithan Ephrati, Martha E. Pollack, and Jeffrey S. Rosenschein. A tractable heuristic that maximizes global utility through local plan combination. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 94-101, June 1995.
13. Maier Fenster, Sarit Kraus, and Jeffrey S. Rosenschein. Coordination without communication: experimental validation of focal point techniques. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 102-108, June 1995.
14. R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189-208, 1971.
15. Michael Fisher and Michael Wooldridge. Distributed problem-solving as concurrent theorem-proving. In *Proceedings of MAAMAW'97*, Lecture Notes in Artificial Intelligence, Springer-Verlag.
16. Michael Georgeff. Communication and Interaction in multi-agent planning. In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83)*, pages 125-129, July 1983.
17. Claudia Goldman and Jeffrey S. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 408-413, July 1994.
18. Marcus J. Huber and Edmund H. Durfee. An initial assessment of plan-recognition-based coordination for multi-agent teams. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 126-133, December 1996.
19. Toru Ishida, Les Gasser, and Makoto Yokoo. Organization self-design of distributed production systems, *IEEE Trans on Knowl. and Data Sys.*, DKE4(2):123-134.
20. Froduald Kabanza. Synchronizing multiagent plans using temporal logic specifications. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 217-224, June 1995.
21. Subbarao Kambhampati, Mark Cutkosky, Marty Tenenbaum, and Soo Hong Lee. Combining specialized reasoners and general purpose planners: A case study. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 199-205, July 1991.
22. David Kinney, Magus Ljungberg, Anand Rao, Elizabeth Sonenberg, Gil Tidhar, and Eric Werner. Planned Team Activity, *Preproceedings of the Fourth European Workshop on Modeling Autonomous Agents in a MultiAgent World*, July 1992.
23. Craig A. Knoblock. *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning.* Kluwer Academic Publishers, 1993.
24. Richard E. Korf. Planning as search: A qualitative approach. *Artificial Intelligence*, 33(1):65-88, 1987.
25. Susan E. Lander and Victor R. Lesser. Understanding the role of negotiation in distributed search among heterogeneous agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 438-444, August 1993.

26. Amy L. Lansky. Localized Search for Controlling Automated Reasoning. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 115-125, November 1990.

27. Jaeho Lee. *An Explicit Semantics for Coordinated Multiagent Plan Execution*. PhD dissertation. University of Michigan, 1997.

28. Victor R. Lesser and Daniel D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):81-96, 1981.

29. Victor R. Lesser and Lee D. Erman. Distributed interpretation: A model and an experiment. *IEEE Transactions on Computers* (Special Issue on Distributed Processing), C-29(12):1144-1163.

30. Jyi-Shane Liu and Katia P. Sycara. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 181-188, December 1996.

31. Douglas MacIntosh, Susan Conry, and Robert Meyer. Distributed automated reasoning: Issues in coordination, cooperation, and performance. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1307-1316.

32. Frank von Martial. *Coordinating Plans of Autonomous Agents*. Lecture Notes in Artificial Intelligence, Springer-Verlag, 1992.

33. Thomas A. Montgomery and Edmund H. Durfee. Search Reduction in Hierarchical Distributed Problem Solving. *Group Decision and Negotiation*, 2:301-317 (Special issue on Distributed Artificial Intelligence), 1993.

34. H. Edward Pattison, Daniel D. Corkill, and Victor R. Lesser. Instantiating descriptions of organizational structures. In M. Huhns, editor, *Distributed Artificial Intelligence*. London, Pittman.

35. M.V. Nagendra Prasad, Keith Decker, Alan Garvey, and Victor Lesser. Exploring organizational designs with TAEMS: A case study of distributed data processing. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 283-290, December 1996.

36. Jeffrey S. Rosenschein and John S. Breese. Communication-free interactions among rational agents: A probabilistic approach. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence volume II*, pages 99-118, Morgan Kaufmann Publishers.

37. Amal El Fallah Seghrouchni and Serge Haddad. A recursive model for distributed planning. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 307-314, December 1996.

38. Sandip Sen and Edmund H. Durfee. A contracting model for flexible distributed scheduling. *Annals of Operations Research*, 65, pp. 195-222, 1996.

39. Yoav Shaham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 276-281-380, July 1992.

40. Young-pa So and Edmund H. Durfee. Designing tree-structured organizations for computational agents. *Computational and Mathematical Organization Theory*, 2(3):219-246, Fall 1996.

41. John A. Stankovic, Krithi Ramamritham, and S.-C. Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real- time systems. *IEEE Transactions on Computers*, C-34(12):1130-1143, 1985.

42. Toshiharu Sugawara. Reusing past plans in distributed planning. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 360-367, June 1995.

43. Katia Sycara, Steven Roth, Norman Sadeh, and Mark Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-21(6):1446-1461.

44. Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1-23, 1993.

45. Keith J. Werkman. Multiple agent cooperative design evaluation using negotiation. In *Proceedings of the Second International Conference on Artificial Intelligence in Design*, Pittsburgh PA, June 1992.

46. D.E. Wilkins and K.L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):731-761, 1995.

47. Chenqi Zhang. Cooperation under uncertainty in distributed expert systems. *Artificial Intelligence*, 56:21-69, 1992.