JOURNAL OF ORGANIZATIONAL COMPUTING, 4(1), 41–83 (1994)

Computational Organization Theory: Autonomous Agents and Emergent Behavior

Michael J. Prietula
Kathleen M. Carley
Carnegie Mellon University

A computational organization theory is the articulation of an organization theory in the form of a computer program. We describe an example of this approach to studying organizational phenomena through the use of simulated autonomous intelligent agents, present a detailed description of such a model, and demonstrate the application through a series of experiments conducted with the model. The model, called Plural-Soar, represents a partial instantiation of a cognitively motivated theory that views organizational behavior as emergent behavior from the collective interaction of intelligent agents over time, and that causal interpretations of certain organizational phenomena must be based on theoretically sufficient models of individual deliberation. We examine the individual and collective behavior of the agents under varying conditions of agent capabilities defined by their communication and memory properties. Thirty separate simulations with homogeneous agent groups were run varying agent type, group size, and number of items in the order list an agent acquires. The goal of the simulation experiment was to examine how fundamental properties of individual coordination (communication and memory) affected individual and group productivity and coordination efforts under different task properties (group size and order size). The specific results indicate that the length of the item list enhances performance for one to three agent groups, but with larger groups memory effects dominate. Communication capabilities led to an increase in idle time and undesirable collective behavior. The general conclusion is that there are subtle and complex interactions between agent capabilities and task properties that can restrict the generality of the results, and that computational modeling can provide insight into those interactions.

distributed artificial intelligence, computational organizational theory

This research was supported through a grant from the National Science Foundation, Division of Information, Robotics and Intelligent Systems, IRI-9111804. The initial implementation of Plural-Soar was completed by Johan Kjaer-Hansen. We gratefully acknowledge the contributions of Allen Newell and the rest of our colleagues in the Soar group at Carnegie Mellon University to this research.

Correspondence and requests for reprints should be sent to Michael J. Prietula, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213

1. INTRODUCTION

Organizations are comprised of individual agents. Organization theory attempts to explain the behavior of individual agents and groups of agents within the context of an organization, and also the aggregate behavior of organizations with respect to their environment [1]. Research in organization theory has taken many perspectives and has incorporated a wide variety of methods, where the variation may be determined by factors such as the unit, time frame, or level of analysis of the researchers [2]. Researchers studying the (macroeconomic) behavior of an organization in a global market over a 10-year period employ different theories, methods, and data than do researchers studying the (psychological) behavior of an organization of individuals working together for one hour to solve a particular problem.

A recurring issue, however, in organization research is how to account for the role played by the individual agents which comprise it; that is, to what extent is it necessary to have an understanding of individual behavior to account for organizational phenomena. Gioia and Sims [3] comment on the situation:

> At their essence organizations are products of the thought and action of their members. Yet organizations also seem to take on a life of their own. An apparent paradox is thus created. For some, organizations are entities guided by thinking individuals who shape the actions of the organization. For others, the distinction between organizations and their members is blurred enough that treating the two as independent entities is problematic. (p. 1)

On one hand, individual agents are characterized in organization theory by their inherent functional limits. Simon [4] has argued influentially for an alternative to the rational expectations model of individual agent decision making, bounded rationality, where individual agents do not (indeed, cannot) consistently engage in optimal search, but incorporate aspiration levels below optimality to yield satisficing behavior. A corollary to this thesis involves the concept of *procedural rationality* in which the rationality of an agent's behavior is determined situationally by the particular decisions to be made by an agent and the computational mechanisms (limited knowledge and processing capabilities) an agent can bring to bear in making those decisions [5,6]. Thus, even if an agent's goals were consistent with rational expectations, the functional limits of the agent would invariably constrain deliberation and choice except under the most simple decision contexts. As such, bounded rationality has directly or indirectly influenced a variety of organization theorists [7,8,9].

On the other hand, three developments have precipitated another look at the role of the individual agent in organization theory. First, there has been a general, sometimes subtle, movement toward linking macro/group/social phenomena with individual behavior. This has occurred in sociology and social psychology [10–14] as well as in aspects of organization theory itself [15–17]. This tension seems to be, in part, similar to the duality (and the many attempts at unification) in physics where general relativity speaks to large-scale events

and predictions, while quantum mechanics addresses the mechanisms on extremely small scales [18]. Yet, as Simon [19] has suggested, if we elect to align our views within a perspective from natural science, perhaps our alignment should be with biology and not with physics. People are not particles, organizations are not planets, and markets are not solar systems. Biologically influenced views of organizational development, evolution, and mortality borrow terms and concepts from behavioral, evolutionary, molecular biology, and ecology [20–22]. But these are views based on analogy and metaphor, and not directly derivative from those biologies. People are not genes, organizations are not organisms, and markets are not nature. Is this a problem? The search for a resolution of the Large and the Small is both diverse and interesting. This article derives from this movement and seeks to link the micro to the macro.

A second development concerns advances in cognitive science which have generated models of individual deliberation and decision making that go well beyond the constraints of bounded rationality [23,24]. The focus has moved from determining how decisions are made within a particular situation, to how decisions are made within a particular architecture, given specific knowledge within a particular situation. As Simon and Kaplan [25] define it:

> The fundamental design specifications of an information-processing system are called its *architecture*. The components of the architecture represent the underlying physical structures but only abstractly . . . . The amount of detail incorporated in an architecture depends on what questions it seeks to answer, as well as how the system under study is actually structured. (p. 7)

The nature of the architecture determines physical, organizational, and processing constraints on deliberation at some level of abstraction. Thus, decisions and knowledge are investigated within the context of a specific and unchanging architecture that remains constant. In addition, the nature of the particular architecture proposes specific hypotheses on the nature of deliberation and how those hypotheses interact with the knowledge and decisions made. As will be explained in Section 3, the specific architecture we selected permits comparisons about *cognitive deliberation effort* across agents and tasks.

Although various components of cognitive science perspectives have crept into the jargon of organizational researchers and theorists [26,27], none have incorporated a broad cognitive model as the fundamental component of an organization theory. For many macro-level organization theories, such a model is largely irrelevant, given their theoretical goals and level of abstraction. For other theorists, models of human deliberation are essential as their views are based on assumptions of how humans reason and how that reasoning ensues within the context of an organization. Yet, even such microtheorists tend not to use cognitive models of general intelligence. In this paper we take advantage of a specific cognitive model of general intelligence and examine the organization as a collection of intelligent agents.

Third, as in science in general, there is a broad and increasing incorporation of modeling and theory development through computer simulation. The use of

simulation in organization science is not new. Simulations have had a significant impact on organizational theory [28–30] and have been used to investigate issues such as organizational learning and turnover [15,31], firm structure and organizational decision making [32–35], and social structures [36]. However, our research differs from these in two ways. First, we simulate an organization through the distinct simulation of the *individual* agents comprising the organization. Second, the *cognitive sophistication* of simulated agents we employ is higher than most and is an instantiation of an architecture representing a general, unified theory of cognition. Though interesting work on organizational theory has been *influenced* by the cognitive or AI analogies [37], we directly *include* a theory of human deliberation. The specific theory we incorporate is that proposed by Newell [38] and realized in a cognitive simulation called Soar [39].

In this article we begin to explore aspects linking group phenomena with individual behavior. The type of organizational theory underlying this approach is called a *computational organization theory* in that a theory of organizational behavior can be articulated and investigated (in whole or in part) through the construction of an assemblage of intercommunicating computer-simulated agents. In the program lies the theory. The specific theory that we employ, ACTS theory [40] argues that the fundamental component (and thus foundation of causal interpretation) of organizational behavior resides with the appreciation (and therefore specification) of a goal-directed, but cognitively restricted, socially situated intelligent agent capable of learning and communication with a dynamic organizational environment. Implicit in this perspective is a strong proposal: The architectural mechanisms by which individual agents perform organizational tasks are precisely the *same* mechanisms by which they perform all cognitive tasks requiring deliberation. The cognitive mechanisms the agents bring to bear on the task do not change, rather, the environment (i.e., the task, the socio-organizational setting) and the agent's responses to the environment (i.e., application and acquisition of knowledge) changes. Organizational behavior is thus viewed as *emergent behavior*—the collective patterns of individual agents acting (and interacting) within the structure of a specific organizational environment and in the context of performing tasks.

As the properties of the individual agent (e.g., individual goals, knowledge, preferences, model of the situation), the properties of the task (e.g., prescribed goals, task structure), and organizational environment (e.g., organizational structure, communication structure, social setting) impose constraints on individual and, therefore, collective behavior, it is important to understand (and, ideally, manipulate) those properties in order to understand and explain how they account for variation of group phenomena.[1] The emergent behavior of even small groups can be difficult to explain causally, as Schelling [41] points out:

[1] There are, of course, organizational or group tasks in which the properties of the individual agents do not matter and when certain collective patterns cannot be altered. For example, consider Schelling's [41] descriptions of "musical chairs" and "trying to get rid of Canadian quarters." In the former, one person will always be left chairless when the music stops. With the latter, each individual agent can, at some time, pass the quarters on to another individual, but the collective cannot "pass them on"—they still remain in the system. Thus, it is important to understand how task and socio-organizational constraints interact with agent properties as well as research goals. An observation that has not gone unnoticed in cognitive psychology [42,43].

> And sometimes the results are surprising. Sometimes they are not easily guessed. Sometimes the analysis is difficult. Sometimes it is inconclusive. But even inconclusive analysis can warn against jumping to conclusions about individual intentions from observations of aggregates, or jumping to conclusions about the behavior of aggregates from what one knows or can guess about individual intentions. (p. 14)

This, in effect, is a reductionist approach to understanding how aggregates of individual agents can account for collective behavior. Prediction and explanation are the grails of a scientific endeavor; furthermore, the corresponding theory must be commensurate with the prediction and explanation in scale and scope. That is, if a theory is proposed (or ascribed to) which attributes as the causal mechanism of group (organization) behavior to properties of individual agents, one should ensure that the theory (as well as any investigative methods) encompasses the particulars (and the variations) of the relevant properties of individual agents. In other words, the explanation cannot be applied to a finer resolution than the theory or the data.[2] But the micro and macro perspectives are not in opposition, as Staw [17] observes:

> reductionism is not just a way to pick up additional variance missed by macroscopic models. Psychological theories can strengthen and add theoretical substance to macro models by providing the underlying rationale or missing process mechanism. This conclusion does not deny the utility of macro models, but simply recognizes them as an interim solution. (p. 810)

The psychological theory we engage for the individual model of deliberation is information processing theory, a perspective somewhat familiar to organization science [46]. However, as we have noted, we rely on what is called a "unified theory of cognition"—a general theory which posits a minimal set of plausible, coherent mechanisms that can account for a variety of cognitive phenomena, such as problem solving, decision making, memory, learning, language, perception, and attention.

The importance of a unified theory of an intelligent agent for organizational science is two-fold. First, an agent placed in an organizational (group) situation is faced with a set of constraints (goals, tasks) in which a broad set of cognitive phenomena may be required. The hypotheses often actually investigated in organizational research of individual agents are not those of an agent's cognitive architecture, but of an agent's *architectural responses* to a real (or experimentally constructed) socio-organizational setting. For example, manipulations to the agent's environment (e.g., method of intercommunication, organizational structure of the group, task assignments, goal specificity, cooperation levels) are made and subsequent behavioral (or attitudinal) variables are examined. What a unified theory brings to the table is a theoretical explanation of how those

[2] In addition, there are experimental implications. In many studies, individual behaviors (or properties) are inferred from an analysis of group data, rather than from detailed observations of individuals. Such inferences can potentially lead to misleading and incorrect attributions and causal explanations [44,45].

responses may occur and what they might be. Are the responses the result of deliberate selection from the agents or are the agents constrained by critical aspects of the task (or their architecture or knowledge, or other agents' architectures or knowledge)? In a sense, it provides a strong theoretical constraint on the specification of agent-cognitive alternatives, thus permitting somewhat more control on the investigative degrees of freedom to explain behavior.

Further unified theories are sufficiently complex that they typically need to be realized as a computer program. Thus utilization of a unified theory automatically puts us in the realm of computational theories. The articulation of the theory in the form of a computer program affords a formalism and methodology which imposes a rigor and discipline proven useful in cognitive science and many other disciplines—the simulation of complex systems [47].

Of interest is how organizational environment changes affect the behaviors of the individual agents and how such changes alter emergent organizational (collective) activity. Thus, explanations of collective action arise out of investigating the intersection of the environment properties (the task and the socio-organizational setting, which may include communicating with, and reasoning about, other agents) and the agent properties (e.g., architectural properties, knowledge, preferences). Tasks and socio-organizational settings can operate to severely restrict the behavior available to an agent and, in effect, *the type of agent it becomes* in a particular situation. Thus even though the complexities of modeling human problem solving are enormous, the requisite components of modeling a human in a restricted task and socio-organizational situation are much less so. What is important is accounting for the fundamental regularities of the cognitive architecture, the nature of the knowledge available within the architecture, and the environmental constraints (task, socio-organizational) imposed on the behavioral alternatives in a particular setting.

The rest of the article is organized as follows. First, we describe the nature of the agents' organizational environment, as the organizational environment is essential in interpreting behavior [48,5]. The task involves agents iteratively retrieving orders to fill and searching a warehouse for the specific items. The organizational setting involves individual agents which act independently or cooperatively (a specific, but simple, manipulation controlling the agents' ability to communicate with each other). Next we describe the architectural substrate on which we built our individual agent models, Soar. Following this, we define the individual agent models and task simulation, Plural-Soar. We then examine the behavior of the Plural-Soar model under 30 different simulation studies where we systematically varied three parameters: the number of agents participating in the task, the number of items on the list an agent is to retrieve, and the capability of the agents to memorize locations of items and to communicate with other agents. We conclude by discussing the implications of the research for organization science.

2. THE TASK ENVIRONMENT: A WAREHOUSE

The Plural-Soar task environment is called the *warehouse task* [49]. Figure 1 presents the configuration of the warehouse. The warehouse is organized as a

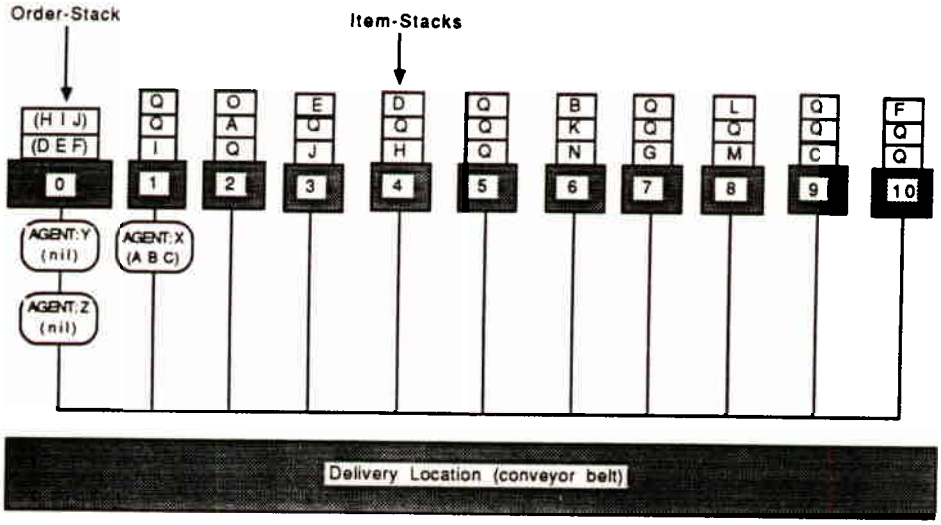

Figure 1. The warehouse task.

row of stacks of items. Agents queue up at a special stack (called the *Order-Stack*) to obtain an order and then proceed to search for the item or items listed on the order. An agent can "see" the contents of a stack only when it is at the specific stack location. When an agent finds an item on the list, the agent removes it from the stack (which may involve intermediate steps of removing other items stacked on it) and places it on the conveyor belt. When an agent has found all of the items listed on that order, the agent returns to the Order-Stack to obtain another order. Agents at the same location (i.e., the Order-Stack or an Item-Stack) cannot interact with the stack at the same time; rather, a first-in, first-out (FIFO) queue discipline is enforced (but agents in the queue can still "see" the items in the stack). Agents can manipulate stacks; they are allowed to move items from an Item-Stack to the stack on its immediate left or right without interference. There are no other physical constraints on agent movement (i.e., there is no other congestion in the model).

Agents can be configured to have a task memory and a communication capability.[3] If an agent has a *task memory* (or *memory* for short), it can remember the contents (i.e., the items) of any stack it encounters directly. When an agent does not know where a particular item is located, it engages in a brute force, uninformed, systematic search through the warehouse. With a memory, an agent may obtain an order and immediately determine through recall the expected location of an item. If an agent has the ability to *communicate*, it can ask other agents if they know where a particular item is located, or respond to

[3] We use the term *task memory* to denote the knowledge about the task/situation that the agent can recall once it physically changes its position or once time has elapsed. Task memory is a parameter we vary in our simulation experiments. It should not be confused with *working memory* or *long-term memory* or *preference memory*, which are parts of the underlying architecture of all agents in our study.

Table 1
Plural-Soar Warehouse Parameters

	Parameter	How Parameter Was Varied
AGENTS	Memory	Agent may have memory of stack contents it visits or no memory of the stack. Agent have memory of questions it was asked if it can communicate.
	Communication	Agent may communicate (ask, answer) with other Agents or not.
	Advice Preference	An Agent may believe its own item memory over any communication. An Agent may believe communcation or its own memory indifferently
WAREHOUSE	Order List	Number of items listed on an order was varied (1 and 3 items per order).
	Number	The number of Agents in the Warehouse was varied (1 to 5 agents).
	Number of Stacks	The number of stacks in the warehouse was kept constant (10).
	Number of Items	The number of items in the warehouse was kept constant (30). The number of requested items was kept constant (15).
	Distribution of Items	The initial distribution of items in the warehouse was kept constant.

another agent's request for location information. As an agent may obtain an order and determine, through asking, the expected location of an item, communication should reduce search effort.

Table 1 illustrates the key parameters that were varied in Plural-Soar. Though the warehouse task and the agents are simple, they collectively provide a rich environment in which to begin to model and examine important issues arising when organizations of autonomous, intelligent agents interact. By restricting the complexity of the task and the agents, the relationship between task properties and agent capabilities can be systematically explored from either an organizational perspective (e.g., coordination schemes, reward structures, organizational structures, group size, communication links, assignment responsibilities) or an individual perspective (e.g., knowledge, trust, strategies, memory, motivation). The key issue we focus on is that representational fidelity for the organization arises from the configuration of an organization comprised of individual agents each based on a general model of intelligence—Soar.

3. SOAR: THE BASE-LEVEL INTELLIGENT ARCHITECTURE

The fundamental architecture underlying Plural-Soar is Soar [39]. Soar is an architecture capable of exhibiting a broad range of cognitive phenomena, such as problem solving, decision making, memory, learning, language, perception, and attention [50]. Soar provides simple representational and control mechanisms that can be applied universally and recursively in a problem-solving task.

The power of soar is derived from its flexibility and, if directed to do so, its ability to modify its own behavior (i.e., learn).[4] We briefly describe Soar so that the reader can see the cognitive basis underlying the organizational agents in Plural-Soar.

At its heart, Soar is a physical symbol processing system [52]. A physical symbol processing system (or *symbol system* for short) is one which has the capacity to access, manipulate, associate, and impose systematic structure on symbols in order to build representational, interpretable symbolic structures.[5] A symbol system is equivalent to a universal computational system and, if properly designed, could satisfy the necessary and sufficient conditions for a generally intelligent agent as it affords the capacity to generate arbitrary response functions to the demands of the environment and the agent's perception of the environment [55]. The problem, of course, resides in the determination of what constitutes a "proper design." How can a system use symbols in a systematic and organized form such that general intelligence may be exhibited? In part, the design of an artifact is strongly influenced by the constraints imposed on aspects of its form or function. For Soar, two of the dominant constraints are [56]:

1. The number of distinct architectural mechanisms should be minimized.
2. All decisions are made through a combination of relevant knowledge at runtime.

The simplicity (and generality) of the architectural mechanisms and the context sensitivity in the application of knowledge within those mechanisms afford Soar important flexibility in its ability to adapt to complex problem-solving tasks—a strong theme in cognitive science [57]. The overall behavior of a Soar program can be described succinctly: the selection and application of *operators* to *states* within particular *problem spaces* to achieve *goals*. The *problem space* is the primary organizational structure of all knowledge in Soar. Goals are the primary reason for the existence of problem spaces; that is, problem spaces arise because goals cannot be directly resolved with the knowledge (as operators) contained within them. All problems spaces have an associated state which is a (possibly complexly linked) set of information represented as objects-attribute-value constructs. Within a problem space, operators manipulate that state until it matches a predetermined pattern defining the achievement of the goal and the resulting termination of that particular problem space.

[4] We will not describe Soar's learning mechanism in detail, as Plural-Soar was directed not to learn in the simulation experiments reported in this article. Details of Soar's learning mechanism can be found in [51].

[5] By establishing the fundamental processing system as symbolic, two goals are accomplished. First, the theory is grounded in a probable physical medium governed by laws of physical behavior. Second, the form of the system approximates the power of a Turing machine [53] in its flexibility and breadth of possible behaviors (subject to physical constraints), thus accounting for characteristic phenomena such as memory, perception, and behavioral adaptations. One alternative to this symbolic perspective is the somewhat more passive and *lower-level* connectionist approach. However, as Harnad [54] points out, it can be demonstrated that connectionist functionality is realized as a special case of a symbol system.

If Soar has difficulty in selecting or applying operators in a problem space (e.g., several operators are possible, but it is not clear which one is best), an *impasse* occurs indicating that the knowledge in the problem space is insufficient to resolve the goal. When an impasse occurs, Soar automatically generates a new goal (to resolve that impasse) and invokes (or creates) a new problem space. Subsequent impasses will result in subsequent (embedded) goals and problem spaces. In many cases, the user may define the major problem spaces in terms of the task and provide knowledge to resolve "task-related" impasses as part of the design of the knowledge base. Regardless of the reason, if any impasse occurs, the Soar architecture handles subgoaling and problem space bookkeeping automatically.

The characterization of a task in terms of goals, problem spaces, states, and operators is not new in cognitive science [58] and one such approach is called the Problem Space Computational Model (PSCM [59]). The realization of a PSCM into a Soar representation requires the translation of the specific PSCM into a set of Soar *productions*. Soar productions look like basic forward-chaining productions, but have a strict individual syntax and role which link the productions to components of the Soar architecture.[6] One consequence of this translation is that there is not a one-to-one correspondence between *operators* as defined at the PSCM level and *operators* as defined at the Soar level—several Soar productions must be written to realize a PSCM operator. The reason for this is that there are several types of decisions that must be made when one, for example, "applies an operator" and Soar bases its representation at a more refined level to address those decisions.

Accordingly, there are six basic production types that are used when building a Soar model (see figure 2). *Problem space proposal* productions are those which detect the conditions under which a new (different) problem space should be tried. *Initial state proposal* productions define the initial state of a problem space (when it is invoked) in terms of an arbitrarily complex set of list structures representing objects of interest for that space. The state contains information which is global to the task (generally a subset of the overall task representation) as well as information local to the particular problem space. *Desired state detection* productions define the goal state of the problem space and monitor the state to see if the goal has been achieved. Generally, within a problem space there may be either (a) multiple operators or (b) multiple instantiations of a single operator (or both) that may be useful to transform the initial state to the desired state. Consequently, Soar makes an explicit distinction between proposing, selecting, and implementing operators which change the problem state. *Operator proposal* productions monitor the state of the problem space and, when appropriate, fire

[6] Forward-chaining production systems are based on the "if <antecedent> then <action>" construct. If certain <antecedent> conditions hold, then <action> statements are engaged. There is a basic constraint that no production construct can directly reference any other production; rather, coordinated execution is based on all productions monitoring the same globally accessible data structure, called a blackboard or working memory, which holds the state variables to which the productions respond via their <antecedent> conditions and to which changes are made via their <action> statements [60].

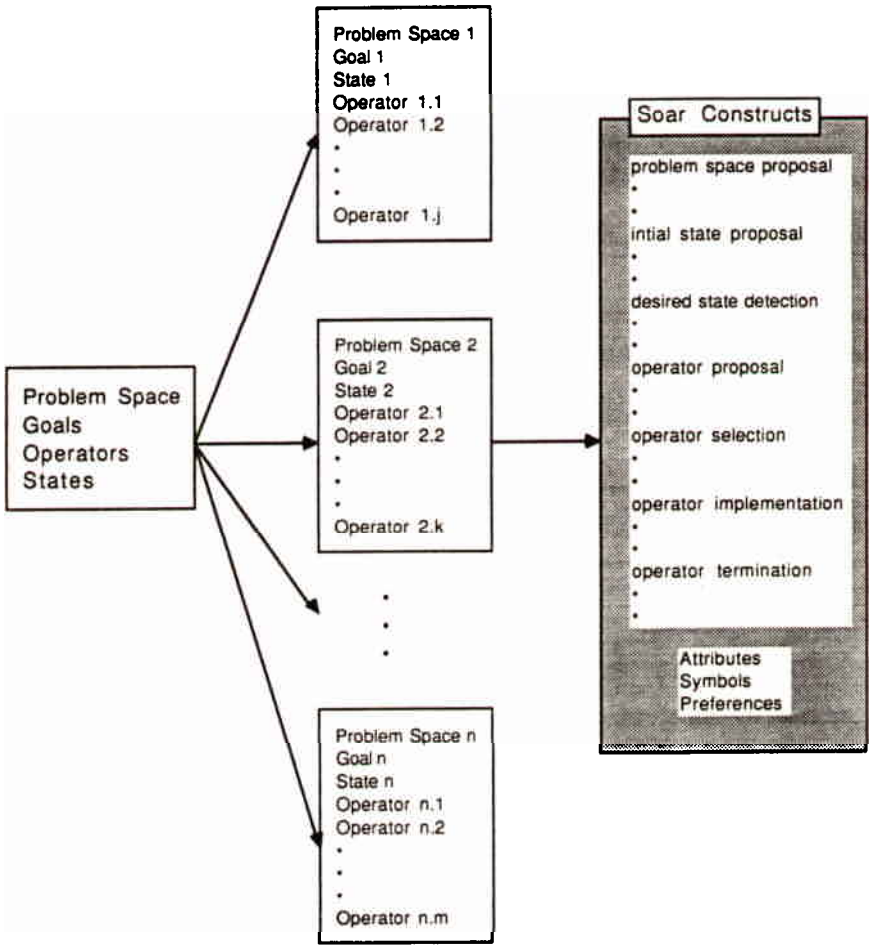

Figure 2. Problem spaces and Soar constructs.

to suggest specific operators to be considered. All potential proposals are made in parallel. Once these have been proposed, *operator selection* productions review and evaluate the proposals to see if one operator is preferred over the others. If one operator is selected, then *operator implementation* productions specific to that operator execute the particular manipulations to the state.[7] Table 2 presents examples of the six production types used in Plural-Soar as well as their English interpretation.[8]

[7] Note that if the set of *operator selection* productions fails to generate an unambiguous operator to implement, an impasse occurs and a new problem space is proposed and entered in a search for knowledge which can distinguish between the operators and thus resolve the impasse.

[8] In Soar 5.2 productions (the version of Soar used for this simulation), a symbol in brackets (e.g., <o>) is a *pointer* if it follows an identifier (e.g., ^operator <o>), and an *identifier* for a specific object if it is in the identifier field (i.e., the second field) of an augmentation [e.g., (operator <o>)].

Table 2.
Examples of Plural-Soar production types

Production Type	*Plural-Soar Example*	*English Interpretation of Production*
Problem Space Proposal	(sp top-goal*warehouse-task (goal <g> ^impasse no-change ^attribute operator ^object <sg>) (goal <sg> ^operator <o>) (operator <o> ^name warehouse) → (goal <g> ^problem-space <p>) (problem space <p> ^name warehouse-ps))	If there is an operator no-change impasse while applying the warehouse operator, then propose the warehouse-ps problem space in which to resolve the impasse.
Initial State Proposal	(sp warehouse*propose*state*initial (goal <g> ^problem-space <p> ^impasse no-change ^attribute operator ^object <sf>) (problem space <p> ^name warehouse-ps) (goal <sg> ^state <ss> ^operator <so>) (operator <so> ^name warehouse ^desired <d>) → (goal <g> ^state <ss> ^desired <d>))	If there has been an operator no-change impasse detected with the warehouse operator, it needs to be applied in a subgoal in its own problem space. The initial state is set to that of the super-state (i.e., that state of the problem space in which it was invoked).
Desired State Detection	(sp warehouse*detect-success*finish (goal <g> ^problem-space <p> ^state <s> ^desired <d>) (problem-space <p> ^name warehouse-ps) (desired <d> ^order <order> ^orders-to-take <off>) (state <s> ^order <order> ^orders-to-take <off>) → (state <s> ^success <d>))	If the Agent is in the warehouse-ps and there is a desired state indicated by the values of the ^order <order> and the ^orders-to-take <off> augmentations and there is, in fact, state augmentations with values equal to those desired, then mark the state as achieving a desired state <d>.

Operator Proposal

```
(sp warehouse*propose*operator*where-is-item
  (goal <g> ^problem-space <p> ^state <s>)
  (problem-space <p> ^name warehouse-ps)
  (state <s> ^location-line <l> ^order-list <o1> ^agents <a>)
  (location <l> ^number 0)
→
  (goal <g> ^operator <o>)
  (operator <q> ^name ask-question ^what <o1> ^who <a>))
```

If the Agent is located at the Order-Stack (location 0) and the Agent has an order, propose an operator to ask the other Agents if they know where the item is located.

Operator Selection

```
(sp warehouse*compare*operator*ask question
  (goal <g> ^problem-space <p> ^state <s>
            ^operator <o1> + {<><o1><o2>}+)
  (problem-space <p> ^name warehouse-ps)
  (operator <o1> ^name ask-question)
  (operator <o2> ^name got-to-location ^reason search)
→
  (goal <g> ^operator <o1>><o2>))
```

If in the warehouse-ps problem space there are two operators proposed, ask-question and go-to-location, establish a preference for the former operator over the latter.

Operator Application

```
(sp warehouse*implement*operator*ask question
  (goal <g> ^problem-space <p> ^state <s> ^operator <o>)
  (problem-space <p> ^name warehouse-ps)
  (operator <o> ^name ask-question ^what <wh> ^who <w> -^asked)
→
  (write 2 (crlf) "Hey" <w> ", do you know where " <wh> "is?")
  (operator <o> ^asked t)
  (state <s> ^speak-link <sl> + &)
  (speak <sl> ^to-whom <w> ^message question ^object <wh>))
```

If in the warehouse-ps problem space there is an ask-question operator proposed, then apply that operator by augmenting the state with information that notes a particular question is being asked (^speak-link <sl>), to a particular Agent (^to-whom <w>) and about a particular item (^object <wm>)

There are three basic "memories" in the Soar architecture: long-term, working, and preference memories. First, there is *long-term* (or permanent) *memory*. All knowledge in long-term memory is encoded as if-then productions of the types just described. Each production in Soar has rigid left-hand side constraints: It must contain references to only goals, problem spaces, operators, states, or state augmentations. Knowledge in long-term memory is cued (activated) by matching the left-hand side pattern of the productions to the objects in working memory. Unlike most forward-chaining production systems, in Soar there is no conflict resolution—all long-term memory productions that match objects in working memory fire in parallel.

Second, there is *working memory*. All knowledge in working memory is represented as augmentations. An augmentation is an object (object-attribute-value triplet) with associated preference for that object. Working memory houses the temporary knowledge that represents the state unfolding as problem-solving ensues.

Third, there is a *preference memory*. When a long-term memory production fires, it does not change working memory directly; rather, the right-hand side of the production proposes preferences for changes which are kept in a preference memory. Soar then evaluates the preferences in preference memory and makes the change decisions according to the results of the evaluation. Changes to working memory are made only after all the preferences for a particular context have been evaluated.

The basic control cycle of soar is called a *decision cycle* and involves an elaboration phase and a decision phase.[9] In the *elaboration phase*, all relevant long-term memory productions are instantiated and the preferences for changes in working memory that do not involve any context objects (i.e., operators, states, or problem spaces) are processed immediately, such as those that propose operators or compare operator proposals. If the preference processing results in changes to be made to working memory elements (e.g., add or delete), the changes are then made. Such memory changes may trigger additional long-term memory productions, which would then trigger additional changes to working memory elements. This *instantiation-decision* loop continues until no more long-term memory productions can be fired—the system has reached quiescence. After soar has reached quiescence in the elaboration phase, the *decision phase* is entered where decisions (via the production preferences) regarding the context objects (operators, states, or problem spaces) are made. As a result of the decision phase, impasses and subgoals may be created or resolved, as selections for problem spaces, operators, and states can occur.

[9] In traces of its behavior, Soar reports activity in terms of *decision cycles* and the decision cycle is the metric of deliberation effort for Soar programs. Thus, when comparing two agents or two Soar programs, the number of decision cycles it takes to complete a task directly reflects the deliberation effort brought to bear in task solutions. The more decision cycles an agent or program takes, the more reasoning effort it has used in solving the problem.

The augmentations in working memory represent the fundamental state of the system and trigger potentially appropriate knowledge from long term memory. Every augmentation in working memory has a unique symbol which is its identifier. An augmentation for an object is given as:

(<object-class> <object-identifier> ^<attribute$_1$-name> <attribute$_1$-value>
^<attribute$_2$-name> <attribute$_2$-value>
.
.
.
^<attribute$_N$-name> <attribute$_N$-value>)

where the <object-class> describes the particular type of object, such as the goal, problem space, state, operator, or user-defined object in the state expressed as a constant, and the <object-identifier> is a unique identifier which determines the object to which one is referring. The <attribute$_1$-name> is generally any user-defined name for an attribute attached to a state (e.g., ^name, ^status, ^top-of-stack) while the value for that attribute is defined via the <attribute-value>, which can be either a constant (e.g., ^name top-goal) or an <object-identifier> (e.g., ^state <s>), where <s> is a symbol that links the attribute name ^state to another augmentation in working memory. Constants permit augmentations to have specific values that reflect attribute-value pairs of interest in the problem space (e.g., ^name Agent1). Object-identifiers permit dynamically defined links between augmentations that can reflect flexible and complex relationships between objects (e.g., ^top-of-stack <item>) or general between-object constraints on pattern matching in the left-hand side of rules (e.g., linking a goal object with a state object with an operator object).

Summarizing, Soar is a forward-chaining, symbolic production system that casts all tasks as a collection of interacting problem spaces with associated goals, states, and operators. Within a problem space, operators are applied to states in pursuit of goals. When specific types of difficulties arise in that pursuit, an impasse occurs and Soar generates a new goal (and associated problem space) to resolve the impasse. If directed to learn, Soar generates chunks as it resolves impasses. Chunks are productions created and added to long-term memory that link desired results of deliberation to working memory elements that led to preferences for the result. Most often, chunks are created in the context of a particular subgoal and problem space; however, chunks are created whenever a state is modified, which may involve changing preferences for any context objects (e.g., goals, operators, problem spaces) that are in consideration.

4. PLURAL-SOAR: A TASK-LEVEL KNOWLEDGE MODEL

The Plural-Soar model is comprised of the core Soar architecture with Common LISP routines to enable interagent communication between programs across the Ethernet. Each Plural-Soar agent is realized by a set of 125 Soar productions which define five basic task problem spaces and the relevant operators within

each problem space (see Figure 3). In Figure 3, problem spaces are denoted by the ovals, and the operators are indicated by operators emanating from the problem spaces. The (comparatively) low number of productions reflect an important aspect of cognitive economy realized by the Soar architecture—the semantics of task performance are defined by the task productions, but the semantics of architectural performance underlying the application of the task knowledge are in the purview of the Soar control mechanism and are not a component of the task knowledge.

The Soar architecture, in the form of the techniques it brings (e.g., goals, problem spaces, automatic subgoaling, chunking), handles much of what is minimally required in driving a general problem-solving (and learning) system. In the parlance of cognitive psychology, Soar's architecture affords a fundamental set of "weak methods" [61] that are universal, but not powerful within the context of a particular knowledge domain. The user defines particular task-specific knowledge that allows Soar to solve problems in a particular domain. The extent to which Soar relies on the weak methods alone, or can craft stronger, domain-specific search control depends on how the problem is specified to the Soar architecture. Thus, the user needs only to focus on domain knowledge representations and not on the control mechanism itself.

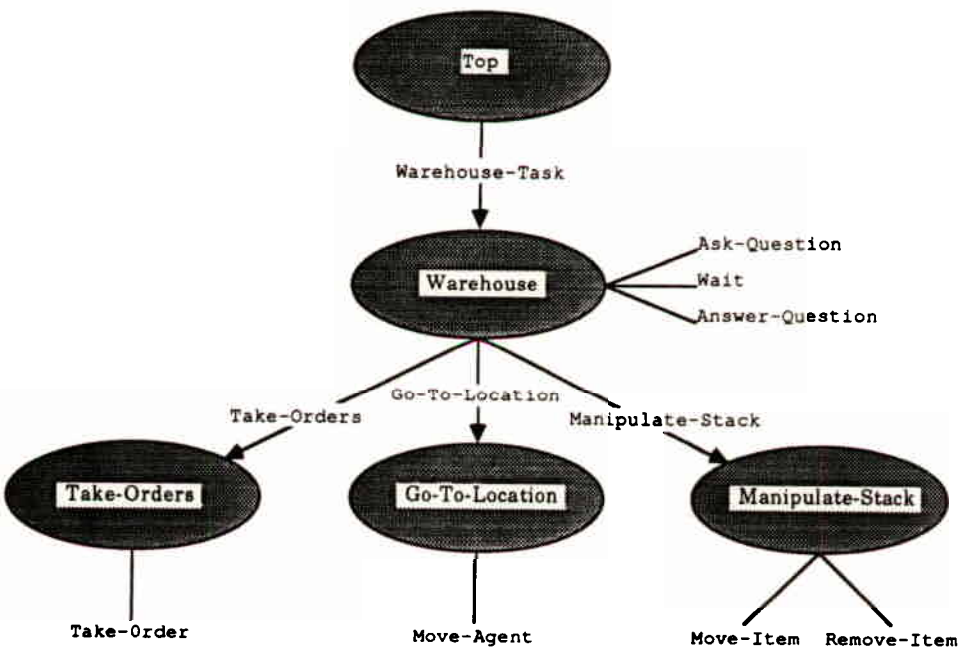

Figure 3. Agent problem spaces and operators.

4.1 Problem Spaces and Operators

The five problem spaces defining an agent reflect a specific (though not necessarily unique) decomposition of the warehouse task from the agent's perspective. In this case, all agents have the same fundamental representation, initial knowledge, and basic capabilities. For any given agent, Top is the top-level problem space proposed for doing the task and determining if the task is completed. This space is the first problem space invoked and components are set up in the Top state to handle such initializations and monitoring as: bookkeeping of moves and other statistics, establishing a basic agent preference for moving to the right when doing an uninformed search for an item in the warehouse, and creating an augmentation which indicates that there are orders to take for the agent and that the agent currently has no orders. In Top, the warehouse operator is proposed to do the overall task of fulfilling all of the orders. However, there is initially no knowledge in working memory to directly implement the warehouse operator. This causes an impasse to occur (the warehouse operator could not be directly applied) and the Warehouse problem space to be proposed.

The Warehouse problem space is the main problem space for actually implementing the behavior of an agent. In the Warehouse problem space is the agent's basic state: a mental model of the warehouse task environment; the basic knowledge for deciding what actions to take in the warehouse; the operators which select and implement those decisions; the capabilities for memory and interagent communication (if the particular agent has these capabilities); and the desired state. The desired state (i.e., goal), from the agent's perspective, is to have "completely filled the order it is working on and no more orders available."

The mental model of the warehouse is comprised of static, dynamic, and statistical augmentations. *Static* augmentations are those representations of the state that are presumed stable, such as, the agent's name as well as the names of the other agents, and the locations of the Conveyor, Order-Stack, and Item-Stacks in the warehouse. *Dynamic* augmentations are those components of the mental model which change over time according to events, such as, if the agent is holding an order, whether the agent believes that there are more orders to take, whether or not an agent has asked (or answered) a question, the particular item the agent is seeking in the warehouse, and the particular direction (left, right) the agent should move using an uninformed search for that item.

Statistical augmentations keep running totals of the agent's moves, orders taken, items moved, time waiting, questions asked, and answers provided. In addition, an agent may have two memory capabilities: the capability of memorizing the items in the stacks it encounters and the capability of memorizing questions it has been asked by other agents.

There are six operators available in the Warehouse problem space:

1. Take-Orders. At the Order-Stack, get an order (or list of orders) from the stack to process.

2. Go-To-Location. Go to a specific location (Item-Stack) to either take an item from the stack or to manipulate the stack to access an item.
3. Manipulate-Stack. At a particular Item-Stack, obtain a desired item from the stack by moving interceding items to adjacent stacks.
4. Ask-Question. Ask other agents if they know where a particular item is located.
5. Answer-Question. Respond to other agents' questions about the location of a particular item.
6. Wait. Skip a turn because an operator cannot be implemented (generally because the agent is in a queue waiting to access either the Order-Stack or an Item-Stack).

In service of these operators, each agent is equipped with two primary input functions and two primary output functions which are Lisp routines that permit data to be passed between the state representations (i.e., agent's mental model) and the external (to Soar) environment. The two input functions are perceive-location and listen-communication. *Perceive-location* permits an agent to perceive its current location and surroundings, such as the items in a given Item-Stack (if it is immediately adjacent to that stack) and the "on top of" relationships between the items on the stack, and any agents in a queue in front of an Item-Stack (and the agent's relative location in that queue). *Listen-communication* permits an agent to hear if any other agent is addressing it with either a question about the location of a certain item or offering an answer to one the agent's own requests for item location information. The two output functions are move and speak. *Move* is general in the sense that it can move either the agent itself in the warehouse, move items of an Item-Stack where the agent is located, or acquire an item from the Order-Stack. *Speak* broadcasts a specific question to another agent (or agents) regarding the location of a specific item, or answering a question from another agent regarding the location of a specific item.

In the Warehouse problem space, there is enough knowledge to directly propose, select, and implement the Wait, Ask-Question, and Answer-Question operators, as these were viewed as basically straightforward operators which required little deliberation effort or variance in their proposals or implementations. The Wait operator is proposed when the agent is in a queue for either the Item-Stack or the Order-Stack, and keeps a running total of how many times (cycles) it was proposed. The Ask-Question operator can be proposed in two ways (expressed here in English rather than Soar syntax):

> If the agent is at the Order-Stack and the agent has an item to retrieve,
> Then propose to ask all agents if they know the location of the item.
>
> If an agent is at an Item Stack, and was motivated to go there through either its own memory or through communication with another agent, and the item is not at the Item-Stack,
> Then propose to ask all agents if they know the location of the item.

The Answer-Question operator is proposed as follows:

> If another agent has asked a question regarding an item, and the agent is at an Item-Stack, and the item of the question is in the Item-Stack,
> Then propose to answer the agent regarding the location of the item.

The implementation of the Wait, Ask-Question, and Answer-Question operators are essentially the adjustments of specific augmentations in the state and, in the case of the latter two operators, the direct invocation of the relevant input/output (I/O) support functions.

On the other hand, the primary effort and deliberation complexity was seen to be in the Go-To-Location, Take-Orders, and Manipulate-Stack operators; consequently, these were implemented in their own problem spaces. The Go-To-Location problem space can be proposed under several situations, such as the following:

> If the agent does not have an item to retrieve, and the agent is not at the Order-Stack,
> Then propose to go to the Order-Stack.

> If the agent has an item to retrieve and the location of the item is known in the agent's memory,
> Then propose to go to that location.

> If the agent has an item to retrieve and the location of that item has been communicated by another agent,
> Then propose to go to that location.

The Go-To-Location problem space has one operator: Move-Agent, which simply moves an agent one stack either left or right. Repeated invocation of this problem space (and this operator) results in movement across the warehouse, and responsibility for evaluating the results (and possible reinvocation) of the problem space is based on the operator proposal knowledge in the Warehouse problem space. If an agent has a memory capability, that knowledge is updated when the agent leaves an Item-Stack to which its attention has been focused (i.e., was exploring the Item-Stack for a particular item). An agent that is "moving by" an Item-Stack to a specific location (and not intentionally exploring Item-Stacks), does memorize any Item-Stacks it passes en route to its specific location goal. Finally, perception dominates memory; that is, previously generated knowledge about an Item-Stack is removed from memory when an agent re-encounters the Item-Stack.

The Take-Orders problem space is proposed under the following condition:

> If the agent does not have an item to retrieve, and the agent is at the Order-Stack, and there are no agents in front of the agent,
> Then propose to take an order from the Order-Stack.

There is one operator in the Take-Orders problem space: Take-Orders. If the agent does not have an order and there are orders on the Order-Stack, then the item on the top of the Order-Stack is taken by the agent. When this is accomplished, that item is removed from the Order-Stack and a count is updated on how many times the agent has taken a new order. The Manipulate-Stack problem space is proposed under the following condition:

> If the agent has an item to retrieve and the agent is at an Item-Stack and there are no agents in front of it
> Then propose to manipulate the Item-Stack.

There are two operators in the Manipulate-Stack problem space: Move-Item and Remove-Item. These two operators are themselves proposed under differing conditions. The Move-Item operator is proposed when the item the agent is looking for is not on the top of the Item-Stack. When this occurs, the agent first tries to move the topmost item to the stack on the right, but if there is no stack there, the agent will move it to the stack on the left. When the agent eventually reaches the desired item, it proposes the Remove-Item operator, which removes the item from the Item-Stack and places it on the Conveyor.

Finally, there is an important type of knowledge implemented in the Warehouse problem space called *search control knowledge* which will guide, under certain conditions, the selection of operators and problem spaces. There are three general conditions which would invoke this warehouse-task search control knowledge. First, there may be multiple operators proposed and one should be preferred over the other in a particular situation. Recall from Section 2 that operators are proposed, but the actual selection is accomplished through an analysis of the preferences active for the operator. An example of this type of search control would be:

> If there are acceptable preferences for two different proposed Go-To-Location operators and one was generated from memory or communication and one was generated for a brute force (i.e., uninformed) search,
> Then prefer the former operator over the latter.

Search control knowledge is used to address the difficulties which may arise when an agent is provided with information that will lead to conflicting goals. Search control is accomplished by predefining a set of preferences and the conditions under which those preferences operator. Setting preferences is a way of imposing social, cultural, and organizational norms on the agent and also a way of characterizing personality types. For example, in Plural-Soar we gave the agents a "lazy" personality. That is, as shown in the prior production, the agents had as a part of their long-term memory a preference for going to locations that had a chance of containing the items (based on socially communicated or direct experiential knowledge) rather than doing a brute-force, uninformed search.

When the agent has a particular item it is to retrieve, and has no knowledge of where it may be, it engages in a default brute-force search which systematically proceeds left to right across the set of Item-Stacks. Memory of where an item may have been seen, or whether an agent has communicated that it knows where an item is, both dominate the absence of knowledge. In effect, the agent trusts its own memory and information from other agents equally.

Second, multiple operators may be proposed, but it really does not matter which one to select; that is, it is not a problem and should not lead to an impasse. Random selection would be an acceptable tactic. An example of this type of search control would be:

> If there are acceptable preferences for two different proposed Answer-Question operators,
> Then be indifferent to the choice.

What this search control knowledge permits is multiple operators (i.e., answers) to be generated in response to a question. For example, an agent may have a list of items (rather than a single item) that it asks about. If another agent knows where more than one of those items is, multiple (and different) Answer-Question operators will be proposed. This knowledge allows that activity and does not require significant deliberation for answering questions.

Finally, there may be situations in which proposed operators should be rejected. For example this type of search control might be employed when an operator has completed its task:

> If an Ask-Question operator has been proposed and that question has been asked,
> Then reject the selection of the operator.

As was noted, various productions play Soar-specific roles in implementing task-specific knowledge and constructs such as the *task operators* described in this section. Furthermore, such constructs may be coded (realized) in many Soar productions. The Soar-specific roles reflect the fundamental components of the architecture described in Section 3—problem space proposal, initial state proposal, desired state detection, operator proposal, operator selection, and operator implementation.

4.2 Example Traces

The dynamic nature of the individual and aggregate agent behaviors can be shown through selections of actual (though abbreviated) traces of Plural-Soar runs. Figures 4 through 7 present traces of two agents, X and Y, in a two-agent organization where each agent has both memory and communication capa-

```
0   G: G1
1   P: P2 (TOP-PS)
2   S: S4 (TOP-STATE)
List:          ((A B C) (D E F) (G H I) (J K L) (M N O))
Location:      0        1         2         3      . . .    10
Stacks:                (Q Q I)   (O A Q)   (E Q J)  . . .  (F Q Q)
Agents  :    (Y X)     NIL       NIL       NIL      . . .    NIL
C. belt :
Initially the agent has no order list,
but believes that there are orders to take
3   O: O10 (WAREHOUSE)
4   ==>G: G11 (OPERATOR NO-CHANGE)
5      P: P12 (WAREHOUSE-PS)
6      S: S4 (TOP-STATE)
7      O: O13 (TAKE-ORDERS)
8      ==>G: G17 (OPERATOR NO-CHANGE)
9         P: P18 (TAKE-ORDERS-PS)
10        S: S4 (TOP-STATE)
11          O: O19 (TAKE-ORDER)   <— Agent X takes order list (A B C)
Order: C
Order: B
Order: A
Taking order
List:          ((D E F) (G H I) (J K L) (M N O))
Location:      0        1         2         3      . . .    10
Stacks:                (Q Q I)   (O A Q)   (E Q J)  . . .  (F Q Q)
Agents  :    (Y X)     NIL       NIL       NIL      . . .    NIL
C. belt :
12     O: O20 (ASK-QUESTION)
Hey Y, do you know where C is?     <— Agent X immediately asks for help
13     O: O21 (ASK-QUESTION)
Hey Y, do you know where B is?
14     O: O22 (ASK-QUESTION)
Hey Y, do you know where A is?
15     O: O23 (GO-TO-LOCATION)
16     ==>G: G32 (OPERATOR NO-CHANGE)
17        P: P33 (GO-TO-LOCATION-PS)
18        S: S4 (TOP-STATE)
19          O: O36 (MOVE-AGENT)  <— Agent X begins search for items (A B C)
Moving agent right: 0 --> 1
List:          ((D E F) (G H I) (J K L) (M N O))
Location:      0        1         2         3      . . .    10
Stacks:                (Q Q I)   (O A Q)   (E Q J)  . . .  (F Q Q)
Agents  :    (Y)       (X)       NIL       NIL      . . .    NIL
C. belt :
```

Figure 4. Plural-Soar trace, Agent X's perspective (decision cycles: 0–19).

bilities. Each agent takes an order containing three items from the Order-Stack, then must locate the items on the order and place them on the Conveyor. We will work through these traces to illustrate a Plural-Soar model in action.

In Figure 4, each numbered line in the trace reflects a Plural-Soar "decision" (i.e., internal deliberated event or proposed external event, such as a physical

```
0   G: G1
1   P: P2 (TOP-PS)
2   S: S4 (TOP-STATE)

List:        ((A B C) (D E F) (G H I) (J K L) (M N O))
Location:    0        1       2       3     . . .   10
Stacks:           (Q Q I)  (O A Q)  (E Q J)  . . .  (F Q Q)
Agents  :  (Y X)   NIL      NIL      NIL     . . .    NIL
C. belt :

Initially the agent has no order list,
but believes that there are orders to take
3   O: O10 (WAREHOUSE)
4   ==>G: G11 (OPERATOR NO-CHANGE)
5      P: P12 (WAREHOUSE-PS)
6      S: S4 (TOP-STATE)
7        O: O13 (WAIT)   <— Agent Y waiting for Agent X to leave Order Queue
8      O: O14 (WAIT)
9      O: O15 (WAIT)
10     O: O16 (WAIT)
11     O: O17 (WAIT)
12     O: O19 (WAIT)
13     O: O20 (WAIT)
14     O: O22 (WAIT)
15     O: O24 (WAIT)
16     O: O26 (WAIT)
17     O: O27 (WAIT)
18     O: O28 (WAIT)
19      O: O29 (TAKE-ORDERS)      <— Agent X has left the Order Queue
20     ==>G: G33 (OPERATOR NO-CHANGE)
21        P: P34 (TAKE-ORDERS-PS)
22        S: S4 (TOP-STATE)
23          O: O35 (TAKE-ORDER)  <— Agent Y takes order list (D E F)
Order: D
Order: E
Order: F
Taking order

List:        ((G H I) (J K L) (M N O))
Location:    0        1       2       3     . . .   10
Stacks:           (Q Q I)  (O A Q)  (E Q J)  . . .  (F Q Q)
Agents  :  (Y X)   NIL      NIL      NIL     . . .    NIL
C. belt :
```

Figure 5. Plural-Sour trace, Agent Y's perspective (decision cycles: 0–23).

movement) resulting from a specific decision cycle.[10] Therefore, Figure 4 presents the first 19 decision cycles of a Plural-Soar run generated from Agent X's

[10] The Plural-Soar model presumes that the proposal of external physical events (e.g., move-left, move-right, ask-a-question) are indeed carried out without difficulty or interference (other than that handled by the agent). Consequently, the internal deliberations resulting in proposed external operators are taken as surrogates for physical behavior. Thus, counting the number of times the agent proposed to move left is taken as equivalent to the number of times the actual physical event occurred. How the external and internal perspectives are resolved in Soar is described by [62].

```
24      O: O36 (ASK-QUESTION)
Hey X, do you know where D is?     <— Agent Y immediately asks for help
25      O: O37 (ASK-QUESTION)
Hey X, do you know where E is?
26      O: O38 (ASK-QUESTION)
Hey X, do you know where F is?
27      O: O39 (GO-TO-LOCATION)
28      ==>G: G48 (OPERATOR NO-CHANGE)
29         P: P49 (GO-TO-LOCATION-PS)
30         S: S4 (TOP-STATE)
31           O: O50 (MOVE-AGENT)  <— Agent Y begins search for items (D E F)
Moving agent right: 0 --> 1
•
•
Moving agent right: 1 --> 2
•
•
37      O: O76 (GO-TO-LOCATION)
38      ==>G: G78 (OPERATOR NO-CHANGE)
39         P: P79 (GO-TO-LOCATION-PS)
40         S: S4 (TOP-STATE)
X tells me that E is at location 3  <— Agent X recalls question Agent Y
41         O: O83 (MOVE-AGENT)          asked about "E" and finds it at
Moving agent right: 2 --> 3             Item-Stack 3
•
•
•
42      O: O32 (WAIT)    <— Agent Y waits for Agent X at Item-Stack 3
43      O: O93 (WAIT)
44      O: O94 (WAIT)
45      O: O95 (MANIPULATE-STACK)  <— Agent X has left Item-Stack 3
46      ==>G: G99 (OPERATOR NO-CHANGE)
X tells me that D is at location 4   <— Agent X answers another question
47         P: P100 (MANIPULATE-STACK-PS)
48         S: S4 (TOP-STATE)
49         O: O103 (REMOVE-ITEM)
Moving item E onto conveyer belt  <— Agent Y moves "E" to the belt
List:          ((G H I) (J K L) (M N O))
Location:    0        1          2          3    • • •    10
Stacks:           (Q Q I)    (O A Q)    (Q J)  • • •  (F Q Q)
Agents  :           NIL        NIL        (Y)   • • •    NIL
C. belt :     A E
```

Figure 6. Plural-Soar trace, Agent Y's perspective (decision cycles: 24–49).

perspective. Other nonnumbered lines reflect events of specific productions which have fired in a particular decision cycle (the one immediately prior to it) that involve printing additional output (e.g., the state of the stacks, location of the agents, and so forth). Each of the numbered (decision cycle) entries is coded according to whether it is showing the active goal (G), problem space (P), state (S), or operator (O). These are the only types of constructs defined in Soar, so these are the only constructs reflected on the trace.

```
303     O: O826 (MANIPULATE-STACK)
304     ==>G: G829 (OPERATOR NO-CHANGE)
305        P: P830 (MANIPULATE-STACK-PS)
306        S: S4 (TOP-STATE)
307          O: O831 (REMOVE-ITEM)  <— Agent Y moves item "O" to belt
Moving item O onto conveyer belt
No more items on list.

List:         NIL
Location:     0        1         2        3    • • •    10
Stacks:             (Q Q Q)     (Q)      (Q)   • • •   (Q Q)
Agents  :     X       NIL       (Y)      NIL   • • •    NIL
C. belt :     AEDBCFIJHKLGNMO

308     O: O837 (GO-TO-LOCATION)
309     ==>G: G838 (OPERATOR NO-CHANGE)
310        P: P839 (GO-TO-LOCATION-PS)
311        S: S4 (TOP-STATE)
312        O: O841 (MOVE-AGENT)
Moving agent left: 1 <-- 2

List:         NIL
Location:     0        1         2        3    • • •    10
Stacks:             (Q Q Q)     (Q)      (Q)   • • •   (Q Q)
Agents  :     X        (Y)      NIL      NIL   • • •    NIL
C. belt :     AEDBCFIJHKLGNMO

313        O: O848 (MOVE-AGENT)
Moving agent left: 0 <-- 1
There are no more orders in warehouse  <— No More Items: Done!

Task statistics:          <— Summary Statistics for Agent Y
----------------
   Agent movements:    56
   Item movements:     20
   Orders taken:       3
   Questions asked:    3
   Questions answered: 1
   Cycles waited:      16
```

Figure 7. Plural-Soar trace, Agent Y's perspective (last decision cycles).

In Figure 4, Plural-Soar begins with a general overall goal (G1), the specification of the TOP problem space (P2), the top-level state (TOP-STATE) where initializations of the warehouse configuration occur. In this configuration, Location 0 is the Order-Stack and the List is the list of items to be retrieved by the agents. Note that this is a "list of lists" where each sublist is the object retrieved by an agent via the Take-Orders operator; therefore, when an agent gets an order from the Order-Stack in this case, it specifies *three* items to retrieve (i.e., the first order list contains the items A, B, C). The remaining Locations are the addresses of the Item-Stacks, and the Stacks line notes the items in the stacks at a particular Location. There are two agents in this run (X, Y) and they are both

initially located at the Order-Stack. There are no items initially on the Conveyor belt.

On the fourth decision cycle, the warehouse operator is proposed. However, as was described in the previous section, the knowledge to implement that operator is not in the problem space, so an impasse occurs and a subgoal is created to resolve that impasse (indicated by the indented arrow = =>). This results in a new problem space being proposed at decision cycle 5 (Warehouse). Eventually, the agent invokes the Take-Orders operator while in the Take-Orders problem space, and the first list is acquired (A, B, C) on decision cycle 11 and the new state of the system is printed. Once the agent has acquired the order, search control proposes that asking is preferable to searching, so the agent asks the other agent if it knows where the items are located (decision cycles 12–14) then proceeds (as no responses occurred) to engage in an uninformed, exhaustive search and moves to the nearest Item-Stack (decision cycle 19).

The abbreviated trace of Agent Y on the same problem (i.e., running at the same time) is given in Figures 5, 6 and 7. The first observation is the obvious series of Wait operators from decisions cycles 7–18. The reason for this is that Agent Y is in the Order-Stack queue behind Agent X. Meanwhile Agent X is, in addition to the basic set-up time and order taking and adjustments, asking if Agent Y knows were the items are, while still in the queue, thus effectively blocking Agent Y's progress. At decision cycle 19, Agent Y (Figure 5) can now propose to take an order from the Order-Stack as Agent X (Figure 4) must have moved out of the way. Agent Y acquires the list (D, E, F) and the resulting change to the Warehouse state is shown. Note that the displayed state of the Warehouse is the "true" state in that it includes all changes made by all other agents and the current state (location) of all other agents.

As Agent Y has the same capabilities and preferences as Agent X, Agent Y initially asks Agent X whether it is has seen any of the items on its list (Figure 6, decision cycles 24–26). Agent Y has a memory, but as it has not yet visited any of the Item-Stacks, it has no idea where any items are located. If it did know (or think it knew), it would not ask the question (see the prior section on search control). No answer is forthcoming, so it proposes to do an exhaustive (uninformed) search starting at the nearest Item-Stack (Stack 1) and then moves to Item-Stack 2. While at Item-Stack 2, Agent Y receives a message from Agent X telling is that one of its requested items (Item E) was seen at Location 3, so Agent Y moves to Location 3. The search now changes from an uninformed search to an informed search. However, Agent Y encounters Agent X still at Location 3, so it has to wait (Figure 6, decision cycles 42–44) before it can begin to manipulate the stack (decision cycle 45). Agent X has moved to Location 4 and has seen another item on Agent Y's requested list and tells Agent Y about it. However, Agent Y is focused on obtaining Item E and moves it immediately (as it is on the top of the Item-Stack at Location 3) to the Conveyor, where Agent X has already placed Item A (when it was at Location 2).

Both agents continue to search, communicate, and acquire new orders from the Order-Stack. The end of the task is shown starting with Agent Y at decision

cycle 307 (see Figure 7) when it moves the last item on its order to the Conveyor (Item 0) and returns to the Order-Stack (Figure 7, decision cycles 308–313). Agent X is already at the Order-Stack, but Agent Y soon discovers that there are no more lists on the Order-Stack to obtain. The task is completed.

Upon completion (all orders are filled), Plural-Soar prints statistics based on the particular agent it is modeling. For this problem, the relevant statistics for both agents are the following:

```
Agent Y:
Agent movements: 56
Item movements: 20
Orders taken: 3
Questions asked: 3
Questions answered: 1
Cycles waited: 16

Agent X:
Agent movements: 32
Item movements: 18
Orders taken: 2
Questions asked: 3
Questions answered: 2
Cycles waited: 0
```

In this case, it lists the number of "physical movements" of the agents (Y = 56, X = 32), the number of times the agents moved items "out of the way" to get to a specific item (Y = 20, X = 18), the number of order lists agents obtained from the Order-Stack (Y = 3, X = 2), the number of times the agents asked questions about specific items (Y = 3, X = 3), the number of times agents were asked where a specific item was and it knew (Y = 1, X = 2), and the total number of decision cycles the agents had to wait (Y = 16, X = 0). Other data can be obtained from the detailed traces of the individual programs running on the different machines distributed across the Ethernet.

In conclusion, Plural-Soar is a group of distributed, intelligent autonomous Soar agents which have memory capabilities, deliberation capabilities, and communication capabilities. The task they are performing is one of search within a warehouse for specific items to retrieve. Each agent is realized as a separate Soar program residing on its own workstation and all communication and perception of the primary warehouse state (i.e., making the actual moves and perceiving the actual warehouse states) is handled over the Ethernet. As there is only one, centralized "external warehouse" representation for the set of agents (i.e., one particular location on the Ethernet), the coordination of all updates and representational fidelity across the set of agents is easily controlled. Thus, it is easy to vary the number of agents, the capabilities of agents (e.g., communicative or not), and the nature of the warehouse (e.g., number of Item-Stack, distribution of items across the Item-Stacks, nature of the orders on the Order-Stack). In the next section, we describe a study which investigated specific variations and their effect on both individual and group performance.

5. SIMULATION STUDIES

A set of 30 simulations was conducted to examine how changes in fundamental properties of individual agents and task properties can lead to variations in emergent individual and group behaviors. The simulations explored the effects of three types of agents, five organizational sizes, and two types of order assignments. The three types of agents were: a *Basic Agent* (with no item location or communication capability), a *Memory Agent* with an item location memory, and a *Communicating Agent* with an item location memory plus the ability to communicate (including the ability to memorize communications to it). The third parameter varied was the number of items on the specific item list retrieved by an agent. When an agent takes an order from an Order-Stack, it is literally a list of items to access in the warehouse. We put either *one* or *three* items on the list. The total number of items was held constant under both conditions—15. Thus, in the first conditions the Order-Stack initially has a stack of 15 item lists, while in the latter conditions the Order-Stack has a stack of five item lists. No items were duplicated on the stack or within a list.

Regarding *agent capabilities*, the first two types of agents (Basic, Memory) reflect a difference in a fundamental capability of an agent—its short-term memory for the task. An agent with a (task) memory of item locations may be considered fundamentally different from an agent that does not have such a memory. Essentially, this additional capability enhances the individual capability, and, possibly, the *individual* productivity of each agent. However, these agents are similar in that neither can communicate to any other agent. The physical coordination mechanisms for these two types of agents are the same (and minimal): FIFO queue at the Order-Stack, parallel search and retrieval of items (with FIFO queues at the Item-Stacks). Thus, the only physical coordination mechanisms are located at the point of assignment (Order-Stack) and the points of activity (accessing the Item-Stacks). Furthermore, these coordination mechanisms are inflexible, deterministic, and externally imposed by the specific task constraints. However, the cultural coordination schemes are different. Agents with memory and agents with communication capabilities have additional cultural constraints in that they prefer to search their memory and communicate before they search the warehouse for each new item.

These agents will proceed as a collection of individual agents each attempting to pursue its private (and privately known) goal(s) and use its experience (as memory) in pursuit of its own goals. Lacking a communication capability, these agents cannot generate a communication-based coordination scheme for their collective behavior as they have no mechanism to exchange information. The addition of a communication capability (with a location memory) permits a given agent to share its experience and, consequently, a simple cultural coordination scheme for search is generated through the preferred mechanism to *ask* where an item is located and to *answer* if it knows where an item is located when asked. Thus, a given agent's perception and memory is functionally distributed throughout the set of communicative agents in the warehouse.

The *organizational size* was varied from one to five agents for each type of agent, a reasonable span for studies of small group behavior [63,64]. For each organizational configuration, all agents had equivalent capabilities. No organizations were configured with a heterogeneous group of agents.

Finally, the *warehouse configuration* consisted of ten Item-Stacks, as previously shown in Figure 1, with three items on each stack. No items appearing on the Order-Stack were duplicated in the warehouse and all requested items were in the warehouse (i.e., no missing items). Therefore, the content of the warehouse consisted of 15 items to be retrieved as specified on the Order-Stack and 15 interference (nonrequested) items. The assignment of the 30 items to the 10 stacks was random and held constant across all conditions. For all studies, agents initially faced the same initial warehouse configuration.

5.1 Method and Procedure

For each individual simulation, an organization of Plural-Soar agents was configured on a set of dedicated individual workstations (Digital Equipment Corporation 3100s) such that each agent ran on a single machine. All agents were connected via an Ethernet and accessed the Warehouse (and communicated with each other) through shared files on a distinguished machine which coordinated access and updates. Program traces and statistics were kept at the individual agent level (by the agent) and collected after each simulation. As was noted, 30 such simulations were conducted. Although this was not a stochastic study, minor random components were introduced in terms of arrival timings through network and specific machine timings.

5.2 Results and Discussion

The general results of the study are summarized in Table 3. A commonly encountered observation for typical, repetitive, noninterdependent organizational tasks is that as the size (i.e., the number of agents available to do the task) of the organization increases, the *total time* it takes to do the task declines, but at a decreasing rate [65]. In Figure 8, the total time taken for each task run was determined by the maximum time (in terms of total decision cycles) for any agent in the particular organization where B is a basic agent, BL is an agent with location (task) memory, and BLC is an agent with memory and communication (each type of agent handling either one or three items per order). Thus, BL3 describes the agents with memory only (no communication) that handle order lists with three items per order. As expected, there is an overall general decline in the total time as the number of agents increases for the task. However, as the size of the organization increases beyond three agents, the benefits are not as pronounced and interaction effects emerge when the different agent types are reviewed.

With a *minimal* number of agents (one or two agents), Basic Agents (no memory or communication capability) with single-item lists (one item per order)

Table 3.
Summary of major results.

Performance Measure	*General Findings for Organizational Performance Measure*
Total Time	Best performing group: Memory agents with 3-item list. Worst performing group: Memory/Communicating agents with 1-item list. Decreases as organizational size increases. 3-item list length generally results in lower total time.
Physical Effort	Best performing groups: Memory, Basic agents with 3-item list. Worst performing group: Memory with 1-item list. Total organizational physical effort decreases with increasing item list length. Average physical effort per agent decreases with organizational size. Average physical effort per agent decreases with large item list length.
Total Organizational Effort	Best performing group: Basic agents with 3-item list. Worst performing group: Memory/Communicating agents with 1-item list. Increases as organizational size increases. Reduced by increasing item list length.
Organizational Effort Saved	Best performing group: Memory/Communicating agents with 3-item list. Worst performing group: Basic agents with 1-item list. Increases with item-list length. Increases with agent complexity (Memory/Communication).
Absolute Gain in Effort	Best performing group: Memory/Communicating agents with 3-item list. Worst performing group: Basic agents with 1-item list. Increases with agent complexity (Memory/Communication).

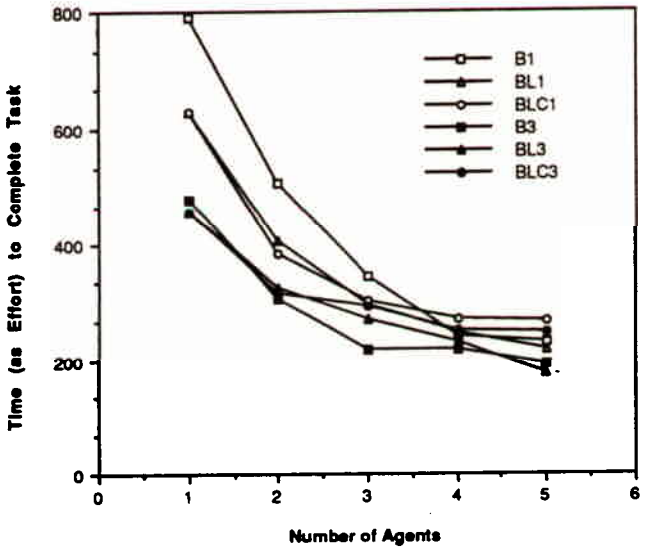

Figure 8. Total task time decreases as the organizational size increases.

take the longest time to complete the task. Agents with single-item lists that have memory and communication (for the two-agent condition[11]) capabilities perform comparatively better than the Basic Agent, and agents with communication capabilities perform slightly better than those which only have memory. Thus, for a small group of agents which process their orders one at a time, memory seems to account for the fundamental decrease in total task time, with communication helping slightly in the dyadic situation. When agents process three items at a time (a three-item list), the total task time is further reduced. The incremental benefit of adding location memory and communication is not as pronounced as with the agents processing single-item lists.

The reasons for these results can be determined by examining the interaction between agent capability and search within the context of the particular task environment. It is beneficial to have a memory, as this incrementally reduces the need to proceed back to the Order-Stack. Referring to the Warehouse configuration (prior Figure 1), consider two separate single-agent situations with Agents BL1 and BL3 which can take one- or three-item lists respectively from the Order-Stack. The first three items on the stack are A, B, and C. Agent BL1 takes the list, (A), and Agent BL3 takes the list, (A B C). Agent BL1 proceeds to Item-Stack 1, notes items Q and I there, then moves to Item-Stack 2 and finds item A. Agent BL3 removes item A to the Conveyor, then returns to the Order-Stack to retrieve the next item list, (B). Agent BL3 behaves like Agent BL1 until it places item A on the Conveyor. It immediately proceeds to search (to the right) for item B, memorizing the stack contents of the stacks it scans en route to finding item B in Item-Stack 6, processes item B, then continues to the right to search for item C, the last item on its order. After item C is found (in Item-Stack 9) and processed, Agent BL3 returns to the Order-Stack and gets the next order, (D, E, F). Although Agent BL3 had a memory, it had to do a brute force (uninformed) search for the items on the first order as it did not encounter any items it was looking for on the stacks. However, the following trace illustrates the next step for Agent BL3 starting at decision cycle 101:

```
101    ==> G: G254 (OPERATOR NO-CHANGE)
102        P: P255 (TAKE-ORDERS-PS)
103        S: S4 (TOP-STATE)
104        O: O256 (TAKE-ORDER) ← Agent BL3 gets next order list.
Order: F
Order: E
Order: D
Oh, I remember item E in location 3
Oh, I remember item D in location 4
```

Agent BL 3 recalls that it has seen two of the items before (items E and D), then proceeds directly to those locations. On the other hand, when Agent BL1

[11] Note that a single agent "organization" does not communicate, so the location memory and communication capabilities are equivalent for these conditions.

takes its next order list, (B), it simply begins another brute-force search and cannot exploit memory until the fourth item list, (D), is retrieved from the Order-Stack—almost 60 decision cycles later than Agent BL3. Furthermore, as it only can take on item, Agent BL1's memory is under utilized—it notes only the location of item D:

```
167    ==> G: G431 (OPERATOR NO-CHANGE
168        P: P432 (TAKE-ORDERS-PS)
169        S: S4 (TOP-SKATE)
170        O: O433 (TAKE-ORDER)
Order: D
Oh, I remember item D in location 4
```

Agent BL1 then proceeds to Item-Stack 4. Given that memory assists agents with single-item lists, why is there not a strong equivalent effect for three-item list agents in equivalent organizational sizes (one to three agents)? Agents without memory capabilities can only rely on brute-force search. List length contributes to a reduction not only in the set-up cost for a search (i.e., the return to the Order-Stack), but also adds a measure of randomness which may benefit the agent for particular item distributions (i.e., the search will start at different locations in the Warehouse). Therefore, it may be the case that the particular distribution of items in the Warehouse did not permit the differences between the two types of agents to emerge.

When all five agents are involved, the situation changes. Item list length dominates the effect of memory, and memory capability dominates the effect of communication. In fact, an organization comprised of five BLC agents (using either single-item or three-item lists) takes longer to complete the task than all other types of agents. Consequently, the total ask time is reduced by adding agents, increasing the order processing list size, and providing a location memory (with effects augmented with certain interactions), but not by adding a communication capability. Furthermore, the incremental benefit of communication can actually be negative as the group size grows beyond three agents.

Insight into these latter results can be obtained by examining the collective behavior of the agents and the particular task characteristics. We first examined the patterns of physical effort (i.e., agent and item movements only) in terms of two metrics: organizational (physical) effort and average agent (physical) effort. In Figure 9 the total *organizational physical effort* exerted by the set of agents shows that the dominant effect of item list length on physical effort. Thus, regardless of agent capability or organizational size, the number of total physical movements required by the organization to accomplish the task was largely determined by the number of items on the order list available to the agents. In Figure 10, the graph of *average physical effort per agent* shows these constant effort requirements are distributed among agents as the number of agents in the task increase—as the number of agents in the organization increases, the required physical effort of each agent decreases.

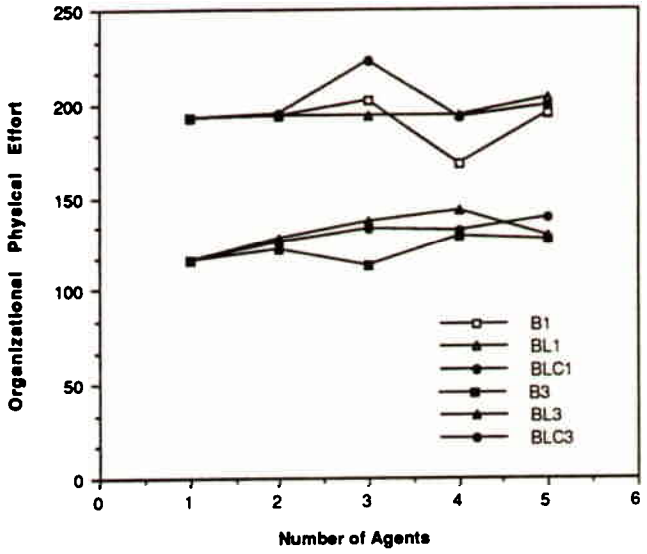

Figure 9. Item list length conditions determine two organizational physical effort levels.

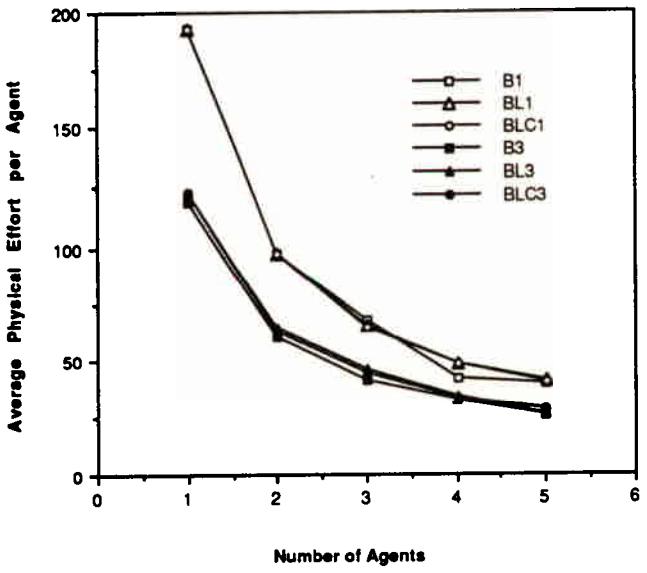

Figure 10. Average physical effort exerted by an agent decreases with organizational size.

However, there is more to the story than that. We define *total organizational effort* as the total number of decision cycles expended by the set of agents comprising the organization in the performance of a task. An examination of the total organizational effort indicates that there is a collective cost of adding agents to the task (Figure 11), an organizational finding reported in software development [66]. It appears, then, that item list length is the main effect, with additional effort being proportionally added with the number of organizational agents. The more agents, the more total effort. Furthermore, there appears to be an interaction between agent capabilities and organizational size on total organizational effort—agents that can communicate seem to incur a comparatively larger additional effort. This additional organizational organizational effort (beyond the requisite physical effort) could be considered as organizational cost.

What is the source of these costs? The main area where additional effort costs are incurred is found when the idle time is examined. In this task, idle time occurs when an agent proceeds to either an Item-Stack or an Order-Stack and encounters another agent in the way, whereupon the agent waits until access to the stack is unencumbered. Figure 12 shows the average idle time per agent for each organization. Note that when there are five agents in the organization, having the "busiest" set of agents, indicated in Figure 12 (three-item lists, no location memory, no communication capability), does not guarantee having the "most effective" set of agents, shown in prior Figure 8 (three-item lists, location memory) in terms of the reduction in total task time. Adding agents to the task increases the idle time.

The idle times, however, are not distributed equally across agent types. The largest idle times are found with the communicating agents. One behavioral preference with this model is that agents prefer to ask where an item is, if it does

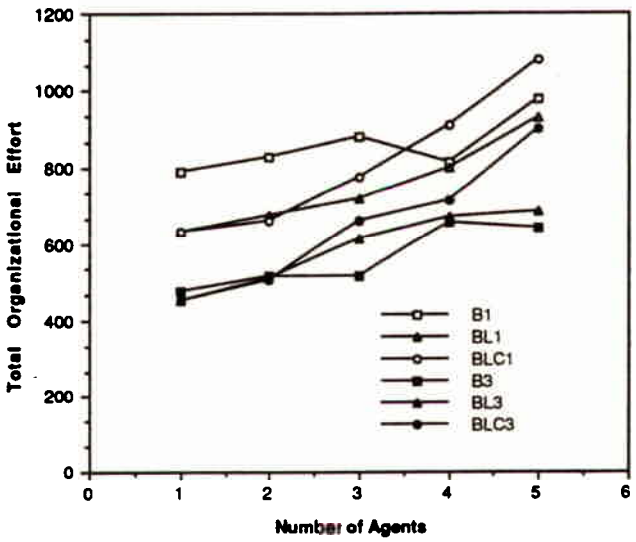

Figure 11. Total organizational effort increases as the organization size increases.

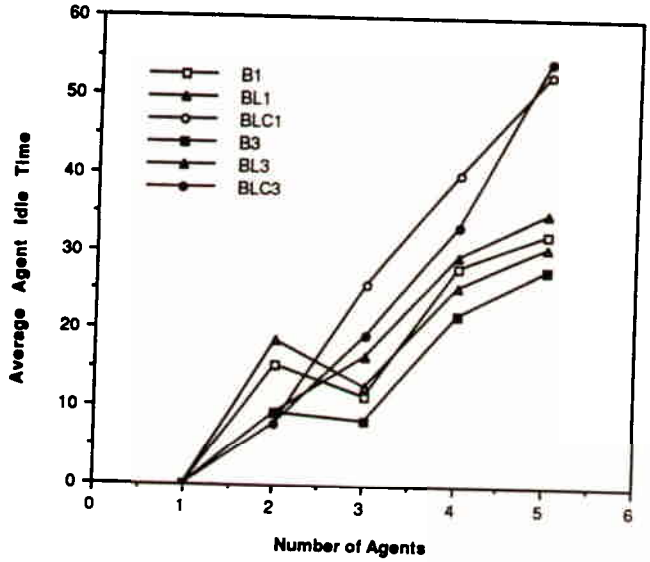

Figure 12. Average idle time increases as the size of the organization increases.

not know, rather than search. Thus when a communicating agent is at the Order-Stack, it acquires its order and promptly asks the other agents if they know where the items are located (recall Figure 4). Perhaps more importantly, such idle time delays the other agents from searching the Warehouse and encountering Item-Stacks to memorize. As a consequence, and perhaps further exacerbated by the low number of items to be retrieved, communicating agents are not that effective as the number of agents increase. This is apparent when the average number of questions asked by the agents is compared to the average number of answers (see Figure 13). As more communicating agents are placed in the organization, more questions are asked. However, the average number of questions answered does not grow nearly as fast because (a) the idle time increases as the queue length to the Order-Stack grows, thus keeping the agents from seeing the Warehouse, and (b) the distribution of the items in the Warehouse affects whether agents have seen them or not.

There is another item list length effect, with more questions being asked by the agents with longer lists and the answers supplied are somewhat more stable (Figure 13). More questions asked by the longer list length agents occur for two primary reasons. First, there is a larger "constant" in that agents immediately ask if any other agent knows where an item is located, so agents with larger lists initially would ask more questions. A second reason actually demonstrates an advantage of memory for the single-item list agents. A given single-item agent will access only one item in the Order-Stack, but ask only one question for each agent. Thus, this agent can get to the Warehouse quicker and begin to search. As the agent searches, and depending on the location of the items, it memorizes the contents of the Item-Stacks it approaches. When the agent returns to the Order-

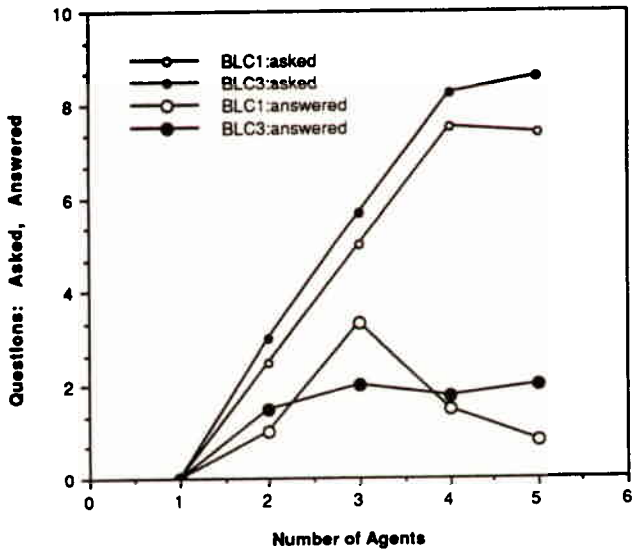

Figure 13. Average questions asked increases with organizational size, but average questions answered does not.

Stack, it may recall where it has seen the item, and does not require a question to be asked. As there are 15 items to be processed, five agents with single-item lists will return to the Order-Stack on the average of three times, while five agents who have three-item lists will not have to return at all.

We see what may be a boundary effect emerging with four- and five-agent organizational sizes, where the number of questions asked and answered are reaching an equilibrium for the task as the size of the organization reaches five. The only benefit of asking a question is if a response can be provided, otherwise, it does not reduce search and contributes to the total effort via increasing idle time. Part of the capability of answering a question is being able to recall experience with the task. Thus, experienced agents are better able to answer questions, but we have seen that the accumulation of idle time reduces experience. In Figure 14, we see that as the organizational size increases, the role of memory declines, as each agent, though "working less" is also "experiencing less" and cannot exploit memory and, consequently, communication.

Initially, agents cannot formulate a cognitive model of the warehouse as they are simply queued to begin the task. Subsequently, they are engaging the task with other agents and cannot generate sufficient experientially based memories, so their commonly shared cognitive models can be exploited through communication. A desirable individual preference (to ask) coupled with a specific group characteristic (communication) led to undesirable collective consequences (total task time not minimized, increase in total idle time) as well as a

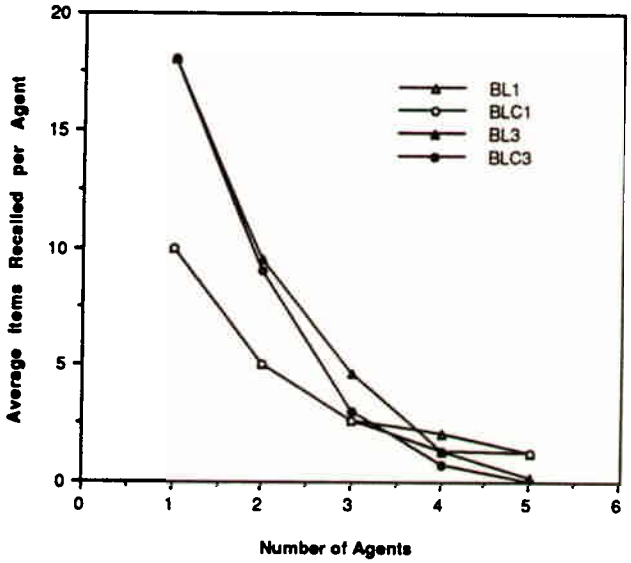

Figure 14. Use of memory declines as organizational size increases.

perhaps undesirable individual consequence (decrease in experiential memory, increase in individual idle time).

A final analysis examined the trade-offs between the effort (as time) saved to complete the task and the cost of added organizational overhead through the addition of agents. The time it takes to perform the task can be reduced by increasing the number of agents in the organization (see prior Figure 8). Figure 15 depicts the amount of organizational effort saved (i.e., decrease in time as measured by decision cycles) in performing the task through the addition of agents. Two observations can be made. First, all organizations experience declining rates of marginal productivity. Second, the largest effect on gain is accounted for the list length, with single-item lists yielding the highest gains. There is a secondary effect, in that agent complexity, within a given list length, accounts for similar orderings. For a given list length, agent complexity, such as memory and communication capability, reduces marginal productivity gains. Therefore, the highest marginal productivity gains are found with the simplest agents using the single-item lists.

Despite the comparatively better marginal productivity gains of simple agents and single-item lists, these organizations are constrained in an absolute sense, by the initial level of effort (prior Figure 8). Their rate of gain is offset by the high effort levels required of the single-agent solution. However, examining the reduction in task time is only one of the effects of increasing the size of an organization.

Reduction in task time by increasing the organizational size brings with it a cost of increased organizational effort (prior Figure 11). Consequently, we can

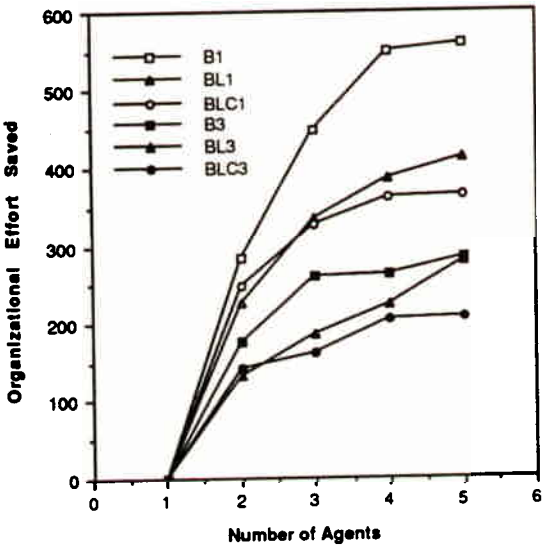

Figure 15. Most effort saved with simple agents using single-item lists.

define the "effort cost of time reduction" to show trade-offs across organizational sizes and types. Figure 16 shows the absolute effort gained (or lost) by increasing the size of the organization for each organizational type. In Figure 16, the initial effort of each organizational type is normalized at zero according to the base effort of one agent performing the task. The incremental effort of each additional agent added to an organization is calculated by subtracting the cost (increase in total organizational effort) from the benefits (reduction in time to complete the task).[12] Thus a positive effort value indicates a net gain to the organization (time reduction > effort added), a negative value indicates a net loss to the organization (time reduction < effort added), while a zero value indicates a direct trade-off. The highest net gain is found with simpler agents using single-item lists (B1, BL1), though a decline sets in when the organization expands to four or five agents. Complex agents (BLC1, BLC3) eventually incur a net cost to the organization, though at differing organizational sizes, but the decline begins at a lower organizational size (three agents). Organizations comprised of noncommunicating agents using three-item lists (B3, BL3) seem to initially decline with three or four agents, but then reverse the trend when the organizational size increases to five agents.

To summarize, the total time to complete the task decreased as the organizational size increased. For smaller organizations, the length of the item list accounts for much of the effort reduction. As the size of the organization increases, there is a general convergence with agent communication capabilities

[12] The metric and operations are mathematically meaningful as effort and time are both measured in the same units and scale (decision cycles).

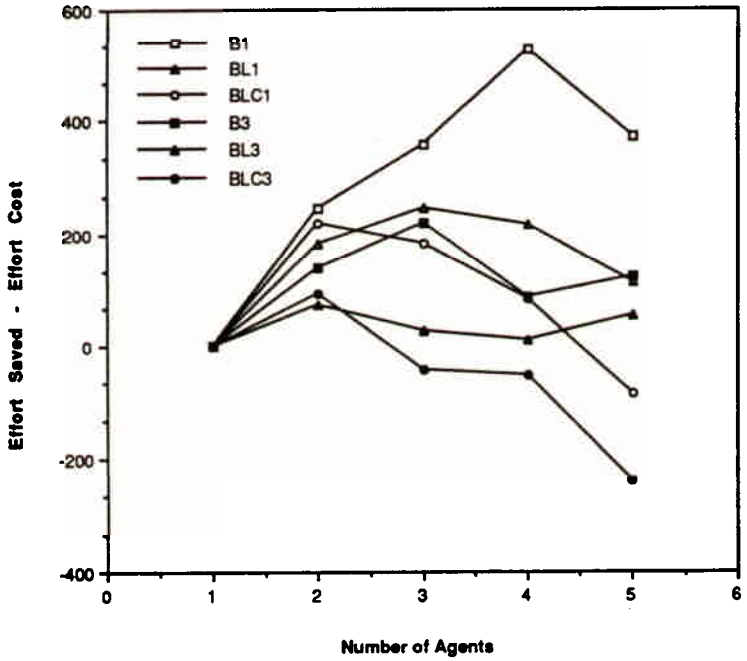

Figure 16. Most absolute gains in effort by simple, single-list agents.

slightly mediating the efforts of item list length. In addition, an increase in organizational size also brings with it an increase in wasted organizational effort. This wasted effort is primarily the result of increased idle time, especially with agents that can communicate. As idle time increases (with the increase in organizational size), the advantages of both location memory and communication erode. The highest marginal productivity rates are found with the simplest agents working with the smallest list size; furthermore, with larger organizational sizes, communicating agents actually incur a net cost in organizational effort.

6. CONCLUSIONS

The immediate goal of the research reported in this article was to demonstrate how an assemblage of autonomous cognitive models of agents performing a simple task could be used to study organizational behavior. The value of this approach can be found in the nature of the fundamental questions about behavior that may be addressed. When considering behaviors of groups of individuals, it is sometimes important to consider the degrees of explanatory freedom one has in attributing cause. Such degrees of freedom vary with the nature of the constraints imposed by the environment or the individuals or both. On one hand, intelligent behavior is based on the extreme flexibility and adaptability of the mind to bring to bear knowledge in the service of goals to perform tasks. We

are members of a larger family of universal Turing machines. Thus the degrees of explanatory freedom are enormous.

On the other hand, we are at the same time severely restricted in that there are limits on speed, depth of deliberation, and knowledge, as well as restrictions imposed by the environment (which includes the task and other agents) as it is initially presented and as the task unfolds. The degrees of explanatory freedom are reduced. What is difficult to assess, however, is where that reduction occurs and, therefore, what explanations are valid to account for the phenomena observed.

In our simple organizational models we learned that a particular task property, order length, dominated smaller organizational sizes in terms of time to perform the task. We also discovered that when an explicit coordination mechanism was introduced (i.e., the ability to communicate to each other on item locations), the organizational efficiency declined, as preferences to engage that mechanism caused idle times. Furthermore, the utility of memory declined as the organizational size increased. Increasing the number of complex agents had a decreasing return, to the extent that agent complexity "costs" something to the organization.

But what did we really learn about organizations? Even in this simple model it was apparent that agent properties and behaviors interacted with environmental and task constraints in complicated ways. However, the causes were unequivocal. For example, optimal individual behavior (asking about the location of an item) had suboptimal organizational effects (increase in idle time) which led to a decrease in individual experience (decrease in the number of stacks searched) that diminished the utility of both memory and communication.

The next steps, then, are to systematically vary the simulation in order to weave a picture of the sensitivity of agent properties (e.g., communication capabilities, knowledge, preferences, experience, goals, assumptions, trust, the ability to learn) to environmental properties (e.g., number of agents, type of agents, organizational structure, communication structure, coordination mechanisms, power relationships, warehouse structure) and how behavioral events emerge over varying time scales. It is those events, the collective conduct of agents acting and interacting in their environments over time, that *comprise* organizational behavior. It is the evolving experience, preferences, and knowledge of individual agents that *explains* organizational behavior.

In many organizational simulations, approximations to behavior can be obtained through normal discrete or continuous event simulation methods or other forms of modeling, as organizational roles and task constraints bound the behavior of an individual to the extent that the individual "becomes" a set of predefined, unambiguous, and immutable behaviors. At the extreme, the constraints of the task negate the power of deliberation and the available degrees of explanatory freedom—recall the musical chairs example of footnote 1. However, individuals are rarely that constrained—when "mind matters" the situation changes. Our solution is bottom-up in that explanatory degrees of freedom concerning individual agent behavior are subsumed within a theory of individu-

al deliberation. The explanatory degrees of freedom concerning the organizational environment are crafted with the task presented to the agents and the socio-organizational situation in which the agents are placed. The observations of how ideally we should meet our more macro, top-down organizational research colleagues is somewhere in the middle.

This research represents a first step toward more cognitively based computational organization theories which can be articulated and tested in the form of computer programs. When this is sufficiently accomplished, we can begin to explicate and link the *fundamental* properties of agents, organizations, and tasks, to the *emergent* properties of agents, organizations, and tasks. This research is a single, exploratory increment toward that goal.

REFERENCES

[1] R. Cyert and K. MacCrimmon, "Organizations," in *The Handbook of Social Psychology*, vol. 1, G. Lindzey and E. Aronson, Eds., 2nd ed. Reading, MA: Addison-Wesley, 1968.

[2] J. Pfeffer, *Organizations and Organization Theory*. Cambridge, MA: Ballinger, 1982.

[3] D. Gioia and H. Sims, "Introduction: Social cognition in organizations," in *The Thinking Organization: Dynamics of Organizational Social Cognition*, H. Sims, D. Gioia, and Associates, Eds., San Francisco: Jossey-Bass, 1986.

[4] H. Simon, "A behavioral model of rational choice," *Quarterly Journal of Economics*, vol. 69, pp. 99–118, 1955.

[5] H. Simon, "Rational choice an the structure of the environment," *Psychological Review*, vol. 63, pp. 128–138, 1956.

[6] H. Simon, "On how to decide what to do," *Bell Journal of Economics*, vol. 9, pp. 494–507, 1978.

[7] J. Galbraith, "Organizational design: An information processing view," *Interfaces*, vol. 4, no. 3, pp. 28–36, 1974.

[8] J. March, "Bounded rationality, ambiguity, and the engineering of choice," *Bell Journal of Economics*, vol. 9, no. 2, pp. 587–608, 1978.

[9] O. Williamson, *Markets and Hierarchies: Analysis and Antitrust Implications*. New York: Free Press, 1975.

[10] K. Carley, "A theory of group stability," *American Sociological Review*, vol. 56, pp. 331–354, 1991.

[11] J. House, "The three faces of social psychology," *Sociometry*, vol. 40, pp. 161–177, 1977.

[12] J. House and J. Mortimer, "Social structure and the individual: Emerging themes and new directions," *Social Psychology Quarterly*, vol. 53, no. 2, pp. 71–80, 1990.

[13] D. Morgan and M. Schwalbe, "Mind and self in society: Linking social structure and social cognition," *Social Psychology Quarterly*, vol. 53, no. 2 pp. 148–164, 1990.

[14] S. Sherman, C. Judd, and B. Park, "Social cognition," *Annual Review of Psychology*, vol. 40, pp. 281–326, 1989.

[15] K. Carley, "Organizational leaning and personnel turnover," *Organizational Science*, vol. 3, no. 1, pp. 20–46, 1992.

[16] K. Crowston, T. Malone, and F. Lin, "Cognitive science and organizational design: A case study of computer conferencing," *Human Computer Interaction*, vol. 3, pp. 59–85, 1987.

[17] B. Staw, "Dressing up like an organization: When psychological theories can explain organizational action," *Journal of Management*, vol. 17, no. 4, pp. 805–819, 1991.

[18] P. Renteln, "Quantum gravity," *American Scientist*, vol. 79, no. 6, pp. 508–527, 1991.

[19] H. Simon, "Rational decision making in business organizations," *The American Economic Review*, vol. 69, no. 4, pp. 493–513, 1979.

[20] J. Singh, Ed., *Organizational Evolution: New Directions*. Newbury Park, CA: Sage, 1990.

[21] J. Thompson, *Organizations in Action*. New York: McGraw-Hill, 1967.

[22] S. Winter, "Economic 'natural selection' and the theory of the firm," *Yale Economic Essays*, vol. 4, pp. 224–272, 1964.
[23] M. Posner, Ed., *Foundations of Cognitive Science*. Cambridge, MA: MIT Press, 1989.
[24] K. VanLehn, Ed., *Architectures For Intelligence*. Hillsdale, NJ: Erlbaum, 1991.
[25] H. Simon and C. Kaplan, "Foundations of cognitive science," in *Foundations of Cognitive Science*, M. Posner, Ed., Cambridge, MA: MIT Press, 1989.
[26] R. Hastie and D. Carlston, "Theoretical issues in person memory," in *Person Memory: The Cognitive Basis of Social Perception*, R. Hastie, T. Ostrom, E. Ebbeson, R. Wyer, D. Hamilton, and D. Carlston, Eds., Hillsdale, NJ: Erlbaum, 1980.
[27] R. Lord and R. Foti, "Schema theories, information processing, and organizational behavior," in *The Thinking Organization: Dynamics of Organizational Social Cognition*, H. Sims, D. Gioia, and Associates, Eds., San Francisco: Jossey-Bass, 1986.
[28] R. Axelrod, *The Evolution of Cooperation*. New York: Basic Books, 1984.
[29] M. Cohen, J. March, and J. Olsen, "A garbage can model of organizational choice," *Administrative Science Quarterly*, vol. 17, no. 1, pp. 1–25, 1972.
[30] R. Cyert and J. March, *A Behavioral Theory of the Firm*. Englewood Cliffs, NJ: Prentice-Hall, 1963.
[31] T. Lant and S. Mezias, "An organizational learning model of convergence and reorientation," *Organizational Science*, vol. 3, no. 1, pp. 47–71, 1992.
[32] R. Burton and B. Obel, "A computer simulation test of the M-form hypotheses," *Administrative Science Quarterly*, vol. 25, no. 3, pp. 457–466, 1980.
[33] S. Clearwater, B. Huberman, and T. Hogg, "Cooperative solution of constraint satisfaction problems," *Science*, vol. 254, pp. 1181–1183, 1991.
[34] A. Kumar, P.S. Ow, and M. Prietula, "Organizational simulation and information systems design: An operations level example," *Management Science*, vol. 39, no. 2, pp. 218–240, 1993.
[35] M. Masuch and P. LaPotin, "Beyond garbage cans: An AI model of organizational choice," *Administrative Science Quarterly*, vol. 34, pp. 38–67, 1989.
[36] S. Banerjee, "Reproduction of social structures: An artificial intelligence model," *Journal of Conflict Resolution*, vol. 30, no. 2, pp. 221–252, 1986.
[37] T. Malone, "Analogies between human organizations and artificial intelligence systems: Two examples and some reflections," in *Artificial Intelligence in Organization and Management Theory*, M. Masuch and M. Warglien, Eds., Amsterdam: North-Holland, 1992.
[38] A Newell, *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press, 1990.
[39] J. Laird, A. Newell, and P. Rosenbloom, "Soar: An architecture for general intelligence," *Artificial Intelligence*, vol. 33, no. 1, pp. 1–64, 1987.
[40] K. Carley and M. Prietula, "ACTS theory: extending the model of bounded rationality," in *Computational Organization Theory*, K. Carley and M. Prietula, Eds., Hillsdale, NJ: Erlbaum, in press.
[41] T. Schelling, *Micromotives and Macrobehavior*. New York: Norton, 1978.
[42] J. Jenkins, "Can we have a fruitful cognitive psychology?" in *Nebraska Symposium on Motivation*, vol. 28, J. Flowers, Ed., Lincoln, NE: University of Nebraska Press, 1980.
[43] A. Newell, "You can't play twenty questions with nature and win: Projective comments on the papers of this symposium," in *Visual Information Processing*, W. Chase, Ed., New York: Academic, 1973.
[44] W. Estes, "The problem of inference from curves based on group data," *Psychological Review*, vol. 53, no. 2, pp. 134–140, 1956.
[45] R. Siegler, "The perils of averaging data over strategies: An example from children's addition," *Journal of Experimental Psychology*: General, vol. 116, pp. 250–264, 1987.
[46] R. Lord and K. Maher, "Alternative information-processing models and their implications for theory, research, and practice," *Academy of Management Review*, vol. 15, no. 1, pp. 9–28, 1990.
[47] H. Simon, "Studying human intelligence by creating artificial intelligence," *American Scientist*, vol. 69, no. 3, pp. 300–309, 1981.
[48] H. Simon, *The Sciences of the Artificial*, 2nd ed. Cambridge, MA: MIT Press, 1981.
[49] K. Carley, J. Kjaer-Hansen, A. Newell, and M. Prietula, "Plural-Soar, A prolegomenon to artificial agents and organization behavior," in *Artifical Intelligence in Organization and Management Theory*, M. Masuch and M. Warglien, Eds., Amsterdam: North-Holland, 1992.

[50] P. Rosenbloom, J. Laird, A. Newell, and R. McCarl, "A preliminary analysis of the Soar architecture as a basis for general intelligence," *Artificial Intelligence*, vol. 47, nos. 1-3, pp. 289–325, 1991.

[51] J. Laird, P. Rosenbloom, and A. Newell, "Chunking in Soar: Anatomy of a general learning mechanism," *Machine Learning*, vol. 1, no. 1, p. 11–46, 1986.

[52] A. Newell, "Physical symbol systems," *Cognitive Science*, vol. 4, pp. 135–183, 1980.

[53] M. Minsky, *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ: Prentice-Hall, 1967.

[54] S. Harnad, "The symbol grounding problem," in *Emergent Computation*, S. Forrest, Ed., Cambridge, MA: MIT Press, 1991.

[55] A. Newell and H. Simon, "Computer science as empirical inquiry: Symbols and search," *Communications of the ACM*, vol. 19, no. 3, pp. 113–126, 1976.

[56] J. Laird, C. Congdon, E. Altmann, and K. Swedlow, *Soar User's Manual: Version 5.2*, Pittsburgh, PA: Soar Group, School of Computer Science, Carnegie Mellon University, 1990.

[57] J. Anderson, *The Adaptive Character of Thought*. Hillsdale, NJ: Erlbaum, 1990.

[58] A. Newell and H. Simon, *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

[59] A. Newell, G. Yost, J. Laird, P. Rosenbloom, and E. Altmann, "Formulating the problem space computational model," presented at 25th Anniversary Symposium, School of Computer Science, Carnegie Mellon University, 1990.

[60] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Reading, MA: Addison-Wesley, 1985.

[61] A. Newell, "Heuristic programming; Ill-structured problems," in *Progress in Operations Research*, vol. 3, J. Aronofsky, Ed., New York: Wiley, 1969.

[62] J. Laird, "Extending Problem Spaces to External Environments," Soar Working Paper, Artificial Intelligence Laboratory, University of Michigan, Ann Arbor, MI, 1991.

[63] A. Cohen, "Changing small group communication networks," *Administrative Science Quarterly*, vol. 6, pp. 443–462, 1962.

[64] M. Shaw, *Group Dynamics: The Psychology of Small Group Behavior*. New York: McGraw-Hill, 1981.

[65] A. Chandler and H. Daems, *Managerial Hierarchies*. Cambridge, MA: Harvard University Press, 1980.

[66] F. Brooks, *The Mythical Man-Month*. Reading, MA: Addison-Wesley, 1982.