Many is More: The Utility of Simple Reactive Agents with Predictive Mechanisms in Multiagent Object Collection Tasks

Matthias Scheutz and Paul Schermerhorn
Artificial Intelligence and Robotics Laboratory
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{mscheutz,pscherm1}@cse.nd.edu

Abstract

This paper investigates low-cost strategies for the multiagent object collection task*, in which multiple agents work together to collect a set of items distributed throughout an environment. Several agent architectures are examined, including simple reactive architectures, more complex deliberative architectures, and "predictive" versions of both of these that take other agents into account when choosing targets for collection. A series of "yardstick" experiments demonstrate that the simple agent types perform very well relative to agents that employ much more computationally expensive approaches. Subsequent large-scale simulations that substantially increase the number of agents and collection items demonstrate that both reactive strategies scale well to more realistic task sizes, with the predictive version performing significantly better than the non-predictive ones.*

1 Introduction

Suppose you need to design a control system for a given number of robots that will gather items in an environment as quickly as possible without any prior knowledge of their distribution and without being able to communicate, while keeping cost (in terms of components) and energy requirements of the controller (as it is running) as low as possible. Problems of this sort could arise in a variety of civil and military logistics (e.g., when supplies are flown in and dropped from planes, which units should retrieve which supplies, and in which order?), as well as more mundane situations (e.g., when a shipper needs to pick up multiple packages throughout a city, which truck should pick up which packages, and in which order?). What kind of control system would achieve the best performance/cost ratio? Observations of simple reactive agents compared to complex deliberative agents indicate that in certain circumstances complex deliberative planning and prediction mechanisms are not necessary to achieve good performance in such collection tasks [24]. However, without some sort of coordination mechanism, agents may work at cross-purposes, hurting the overall performance of the multiagent system.

In this paper we will investigate different agent architectures for *multiagent object collection tasks*, where multiple agents together need to collect objects in the environment. In particular, we are interested in architectures with high performance-cost ratios. The cost of an architecture will be measured in terms of the computational effort to compute an agent's actions at any given time, while the performance will be evaluated in terms of the time it takes the agents to collect all items. The rest of the paper is organized as follows: first, we define the multiagent object collection task and discuss some potential solutions and their tradeoffs. Next we explain what we mean by "reactive", "deliberative", and "predictive" control, and then introduce the four different agent architectures we intend to study: reactive, reactive-predictive, deliberative, and deliberative-predictive. After describing the experimental setup, we report the results from several extensive sets of experiments with these agents that show that the simpler reactive agent types perform well relative to the more expensive deliberative types and to the best possible performance. We then present the results of large-scale follow-up simulations of reactive agent types, in which the numbers of agents and items to be collected are significantly higher; these large-scale simulations show that the performance of the simpler agent types scales well to more realistic task sizes. Finally, we analyze the results and discuss their implications for behavior coordination in multiagent tasks.

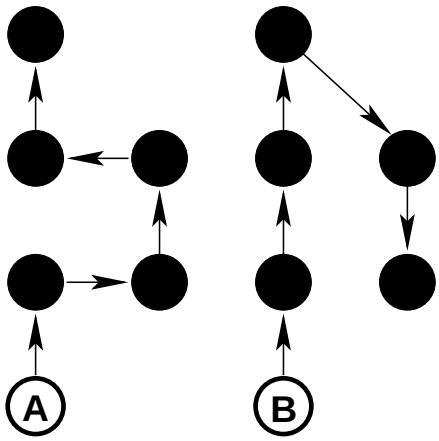

Figure 1. Optimal paths for MOCT and Traveling Salesman problems: A solves the MOCT optimally in less time than B, because B solves the TSP optimally and drops the last leg of the tour.

2 The Multi-Agent Object Collection Task

The *multiagent object collection task* (MOCT) consists of A randomly placed agents that need to collect C items, also randomly-placed in an environment. The challenge is to collect all C items in as little time as possible, while avoiding B obstacles. All agents travel at a uniform constant speed, allowing a direct comparison of collection strategies between agent types.

A concrete example of a MOCT in a real-world setting is the assignment of scientific tasks to a group of extraplanetary rovers. Landing rovers is an inexact science, so the placement of a group of rovers on a planet's surface cannot be determined ahead of time. The solution must be calculated after the agents have landed and their initial positions are determined. Also, rovers have tended to travel the terrain seeking and executing scientific goals without needing to return to a base station. They operate until their batteries run out, then the mission is over. This corresponds to the MOCT ending when all items have been collected without bringing the items back to a depot. NASA's ASPEN (Automated Scheduling and Planning Environment) project generates command sequences for multiple rovers to achieve multiple high-level scientific goals [20, 9]. The problem is treated as a multi-Traveling Salesman Problem, and a greedy insertion method is used to construct tours. It is possible to obtain a solution to the MOCT from a solution to the TSP by removing the edge between the penultimate vertex in the tour and the start/end vertex. However, there is a difference between optimal solutions to the MOCT and optimal solutions to the TSP. Figure 2 depicts two agents' solutions to the MOCT. Agent A solves the problem optimally, traveling a distance of 5 units. Agent B, however, solves the TSP optimally, but leaves off the final step of the solution. Because B assumes from the outset that it will return to its starting position, it includes the longer diagonal step in its path, requiring a distance of $4 + \sqrt{2}$ units to accomplish the MOCT. Hence, in order to use a TSP approximation for the MOCT, it will be necessary to employ a heuristic that does not assume a return to the initial position in the costruction of the path.

2.1 Solutions to the MOCT

Given the initial positions of the agents, items, and obstacles present in the environment, it is possible to solve a MOCT optimally at the beginning by exhautively enumerating all combinations and choosing the one with the shortest time to completion. Each agent can then be given its assignment of objects to collect (including the order in which they should be collected) and the ojects can be collected in the shortest possible time. However, the optimal solution is computationally intractible for even modest numbers of agents and items. Furthermore, in dynamic environments where the items can move from one location to another or new items can be generated during the collection task, the solution may need to be recomputed during the course of executing the solution. Limiting agents' sensory ranges also effectively turns the problem into a dynamic one, since a new solution will need to be computed whenever another item is sensed. What is needed are flexible *online algorithms* (rather than *offline* algorithms) that provide good (although not necessarily optimal) solutions in dynamic environments at a reasonable cost. We are particularly interested in online algorithms that have a low computational cost, while showing a reasonable performance compared to the optimal solution. Note that while recomputing the optimal solution each time the environment changes is an online algorithm that yields an optimal solution, this possibility is again practically infeasible.

A first pass at a solution to the collection task would be for each agent to always attempt to collect the item that is closest to it. The intuition behind this "greedy" strategy is that going far out of the way to collect an item costs more than collecting the nearby one, making the greedy algorithm attractive. Especially in multi-agent environments, however, this strategy can lead to a great deal of wasted effort. If an agent moves toward the closest item, only to have another agent collect the item just before the first agent arrives, the first agent's effort is wasted.

This situation can be avoided by coordinating agents' actions. For example, by communicating their intentions to one another, agents will be able to reduce or eliminate altogether the wasted effort arising from situations where two or more agents attempt to collect the same resource: for

every agent knows what its goal is and communicates it to all other agents. In the case of conflicting goals, additional mechanisms are required to determine the agent that should eventually collect the item and reassigning other items to the remaining agents. And even if optimal group behavior is not achievable (e.g., because of the nature of the task or the computational requirements of the distributed algorithm), there is at least the benefit that agents will *know* what other agents are up to (always assuming they tell the truth, of course). However, communication can be expensive, and, furthermore, it does not in itself solve the problem of who *should* retrieve the resource in question, it is merely a tool to be used by whatever control mechanism does solve the problem.

In the absence of communication, reasoning about other agents' intentions based on observations can be a good substitution. If, for example, every agent in a group constantly observes the behavior of other agents, knows how other agents chose their behavior, and chooses its actions based on its predictions, then the agent group may be able to converge to (several) stable, coordinated behaviors. In the solution explored below, agents use predictions of other agents' actions in order to implicitly coordinate their efforts on the collection task.

While such coordination mechanisms lead to improved performance, the processing cost incurred by their architectural mechanisms can be substantial, as we will demonstrate below. Hence, the gain in task performance may not be worth the loss in processing efficiency. Often, sufficient performance in a multiagent collection task can be achieved either without explicit coordination or with coordination mechanisms that do not involve predicting other agents' behavior, e.g., avoiding coming too close to another agent. Below we describe agents that pursue their goal of acquiring items from the environment without taking other agents into account, in effect working independent of one another. Yet, they tend to avoid other agents, and in particular they avoid crowded regions. This simple rule causes them to display a certain degree of coordinated behavior, and more importantly, allows them to perform very well relative to the low complexity of architectural mechanisms that control their behavior.

Several questions ensue: When are such simple mechanisms sufficient? In what environments do they reach a performance peak for a given number of collectible items as measured in terms of number of agents and the time it takes to complete the task? Does the performance depend on the ratio of agents and collectible items?

To begin to answer these questions, we will study four different agent types: (1) purely *reactive agents* with no predictive ability, (2) *deliberative agents* with planning and information storage facilities, (3) *reactive-predictive agents* with the ability to predict the actions of other agents, and (4) *deliberative-predictive agents* with both deliberative and predictive capabilities. By contrasting the performance of these four agent types, we demonstrate that the more computationally intensive architectures do not always provide as great an advantage as one might expect, particularly when considerations of computational cost are taken into account. In addition, the results presented here provide insight into the usefulness of different types of control mechanisms in the context of agent cooperation.

3 Reactive, Deliberative, and Predictive Architectures

Agent architectures play an important role in the understanding of the development of natural and artificial systems [23, 22]. They can be thought of as blueprints of control systems, where different functional components and their interconnections are depicted (e.g., see [21] for a more detailed definition of "agent architecture"). Since we would like to understand (1) what kinds of components (and arrangements thereof) are required to produce particular kinds of behaviors, and (2) what the relative tradeoffs of different control systems (and their implementations) are, we first need to define what we mean by "reactive" and "deliberative" control, or more to the point, what "reactive" and "deliberative architectures" are.

3.1 Reactive Architectures

Unfortunately, there seems to be a wide range of definitions of "reactive" that differ in substance (e.g., "reactive" as "stateless" versus "reactive" as "tight sensor-motor coupling"). Hence, it seems that "reactive" is best defined in opposition to "deliberative", i.e., as "not deliberative", which puts the burden on a definition of "deliberative". Since we are interested in demarcating an intellectually interesting difference, rather than trying to say what "deliberative" *really means*, we will construe "deliberative" as "being able to produce and use representations of hypothetical past or future states or as yet unexecuted actions (or sequences of such actions)". Note that according to this (negative) definition of "reactive", reactive architectures may make use of simple representations of the state of the world and/or the agent. But these representations will not explicitly encode goals, hypothetical states of the world or sequences of possible actions. And while we may be able to ascribe intentional states such as beliefs and desires to a reactive agent, the agent architecture contains no explicit representation of these states. For example, an agent which exhibits a behavior that could be described as "avoiding obstacles" can be said to have a goal of "avoiding collisions", even though this goal is not explicitly represented in the agent's control system.

3.2 Deliberative Architectures

As mentioned before, a *deliberative* architecture is one in which there is some consideration of alternative courses of action before an action is taken. Hence, there is need for the capacity to represent counterfactual states referring to hypothetical past or future states or as yet unexecuted actions (or sequences of such actions), in which at least some of the basic operations of the architecture are to produce, read, and write such counterfactual states. Such states include goals (descriptions of states to be achieved), plans (sequences of unexecuted actions), states describing the imagined consequences of performing an action in the current state or some hypothetical state, partial solutions generated during planning or problem solving, the hypothetical states of the agent's beliefs generated during belief revision, and many others. We further require that such states should be influential in the production of actions, in the counterfactual sense that, had the (counterfactual) state not been generated, the agent would have chosen a different action to execute.[1]

To represent counterfactual states, a deliberative agent requires a reusable working memory for the construction and comparison of hypothetical states and some means of deriving the consequences of actions performed in these states. At its simplest, this might be a set of memories of the consequences of performing the action in similar states in the past. The use of a common working memory limits the number of alternative courses of action that can be considered in parallel, and hence the degree of parallelism possible within a deliberative architecture.

All other things being equal, a deliberative architecture must be slower and require more resources than a reactive architecture which encodes a solution to any specific goal solvable by the deliberative architecture, since the generation of alternatives will take time. However, a deliberative architecture will typically be more space efficient than an equivalent reactive architecture, even though it will often require more space than a reactive solution to any given problem instance, since it can solve a *class* of problems in a fixed amount of space, whereas a reactive architecture requires space proportional to the number of problems. We can view this as an example of the standard space-time tradeoff, though in this case there is also the time required to code or evolve all the reactive solutions.

[1] Note that this definition implies no commitments as to whether the states and operations are fine grained, e.g., dealing with partial plans or alternative solutions and their generation and comparison, or whether the states and operations are "coarse grained", e.g., a single "plan" operator which takes a goal and a description of the current state and returns a plan with the rest of the fine-grained states and operators buried in the implementation of the architecture and invisible to the agent program and the agent state. Both cases have at least one counterfactual state and one operator that takes a non-counterfactual state and returns a counterfactual state.

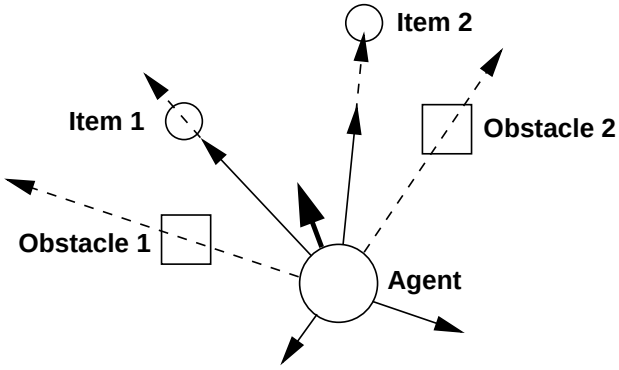

Figure 2. Schema-based navigation in reactive architectures: dotted arrows represent the raw perceptual schemas, while solid arrows represent the weighted schemas. The bold arrow is the sum of the individual weighted schemas, and indicates the direction the agent will travel.

4 Agents: Architectures and Behavioral Dispositions

In the experiments reported in this paper, we employ four different kinds of agents, *reactive*, *deliberative*, *reactive-predictive*, and *deliberative-predictive* agents, where the architectures of the"predictive" agents are extensions of their non-predictive counterparts.

All agents are standardly equipped with exteroceptive "vision" and "touch" sensors. *Vision* is used to detect items and other agents and *touch* to detect (1) impending collisions with agents and (2) items that are within reach for collection. In addition, the touch sensor is connected to a global collision avoidance system, which triggers an automatic reflex-like action pattern, which the agent cannot suppress, to move it away from other agents.

On the effector side, agents have motors for locomotion and turning, and a mechanism for collecting. When agents come to a halt on top of an item, its collection mechanism suppresses the motors for locomotion until the item is collected, which takes one simulation cycle.

While different agents may have different short-term goals at any given time (e.g., reaching an item faster than another agent, or avoiding collisions with other agents), there are two long-term goals that are common to all of them: (1) *collection* (i.e., to acquire as many items as possible), and (2) *survival* (i.e., to avoid colliding with other agents in the environment). In the following, we will briefly describe the employed architectures and behavioral dispositions of each agent kind.

The Reactive Agents All reactive agents process sensory information and produce behavioral responses using a motor schema-based approach [3] (see Figure 4). Let $Ent = \{c, b, a\}$ be an index set of the three types of objects: *items*, *obstacles* and *agents*. For each object type Ent, a force vector F_t is computed, which is the sum, scaled by $1/|v|^2$, of all vectors v from the agent to the objects of type t within the respective sensory range, where $|v|$ is the length of vector v. These *perceptual schemas* are mapped into motor space by the transformation function

$$T(x) = \sum_{t \in Ent} g_t \cdot F_t(x) \quad (1)$$

where the g_t are the respective gain values of the perceptual schemes. The gain values simply scale the effect of sensory input, providing a means by which to prioritize certain inputs (e.g., if collecting items is especially important, the item gain value could be higher than the agent gain value, so that sensing an item has a greater impact on the direction chosen than sensing other agents). These gain values are initialized to values determined to be reasonable via a series of experiments, and are kept constant throughout the life of a reactive agent.

A collision detection mechanism invokes an agent's retreat reflex whenever it detects an impending collision with an obstacle or another agent (all collisions are fatal). The reflex works by inserting a very strong vector leading away from the site of the near collision. This vector is included for a random number of cycles between 5 and 15, and has the effect of moving the agent directly away from the object or agent. The reflex works well in most cases, although it is possible to fail in some situations (e.g., it may be possible to retreat into another obstacle in some circumstances).

Reactive agents always behave in the same way, given that their gain values are *constants*: their positive g_c makes them employ a greedy collection strategy (most of the time a "collect nearest" strategy [25]), whereas their negative g_b and g_a values make them avoid obstacles and other agents. The effect of g_a on the reactive agents' behavior is to establish implicitly a "ranking" of who gets to collect an item first if multiple agents attempt to collect the same item: whoever is closest will be more strongly attracted to the item than repelled by the other agents, and hence be able to get to collect the item, whereas the other agents will be repelled more by the presence of agents than they are attracted to the item, and hence will move away. In a sense, g_a implements a simple "coordination" strategy, if only one that is "negatively" determined.

The Deliberative Agents Deliberative agents have several components that allow them to manipulate representations of collectible items in the environment. Most importantly they have a route planner that can determine which item is closest to them and how they can best get to it. It is first and foremost this ability of being able to represent entities in the environment that opens up further possibilities such as storing and retrieving representations, using them in planning and plan execution, etc. None of these possibilities are available to reactive agents, which have access to sensed objects only in a holistic manner (via agglomerated force vectors).

The planner of the deliberative agents (based on a simplified version of the A^*_ϵ algorithm [19]) is given a list of items known to the agent (i.e., stored in the agent's memory), and returns a *plan*, which is a list of headings and distances, of how to get to the nearest reachable item. The plan is then passed to a *plan execution mechanism*, which ensures that plan steps are executed. When other agents cross a deliberative agent's route and the reflex is triggered, "re-planning" is initiated, and the agent will continue by executing the new plan. Re-planning is also performed if the item chosen by the agent has been collected by another agent in the meantime. A further difference between deliberative and reactive agents is that, while the schema-based mechanism of the reactive agents will not pick out the most direct route to an item (because of the influence of other items and agents), and may even move *away* from the nearest goal item (because of a cluster of objects further away in the opposite direction, or a cluster of agents in the direction of the nearest item), deliberative agents will find the nearest item and plan a route directly to it (while avoiding other agents), thus saving time and energy.

The Prediction Extension In many cases agents will pursue the same goal, which reduces the overall efficiency of the agent group: an agent should not waste time moving toward the same item another agent is pursuing if the second agent is closer to the item than the first is. The function of the prediction extension is to make an "educated guess" as to which goals other agents might be pursuing, so as to eliminate those items as possible goals. The prediction extension was added to both kinds of agent architectures, creating two new classes of agents: reactive-predictive and deliberative-predictive. The algorithm implemented by the prediction component, which is used to eliminate common items (i.e., items that more than one agent is likely to pursue), is given in Figure 3. It ensures that no two agents will ever be pursuing the same goal item.

The Reactive-Predictive Agents The extension is integrated into the reactive agent by applying the algorithm `Eliminate-Items` to the appropriate type in Ent_k before its force vector is computed. This effectively functions as a "perceptual filter" that prevents those perceptual schemas from being instantiated that correspond to items

```
FUNCTION EliminateItems(agentlist,itemlist)
for all A ∈ agentlist do
   closest =infinitely-far-item
   for all I ∈ itemlist do
      if dist(A, I) < dist(A, closest) then
         closest = I
      end if
   end for
   if dist(A, closest) < dist(Me, closest) then
      remove(closest, itemlist)
   end if
end for
return(itemlist)
```

Figure 3. The algorithm for Eliminate-Items.

that might be pursued by other agents. Consequently, the agent cannot be attracted to those items any longer and will simply ignore them on its way to pursuing other items. Note that reactive-predictive agents do not entertain or manipulate representations of other agents' goals, nor do they "reason" about other agents' intentions.

The Deliberative-Predictive Agents The prediction extension is integrated into the deliberative agent by applying `Eliminate-Items` to the list of items passed to the planner, which in turn selects its goal. The chosen goal is then the closest item to the deliberative-predictive agent that is not the goal of any other agent (i.e., no other agent is closer to it and not closer to any other item). The planner proceeds as in a normal deliberative agent, and the plan executes to pursue the uncontested goal, until a conflict is detected, i.e., a situation in which the current goal might have been chosen by another closer agent, in which case replanning will be triggered. In sum, the agent ignores items that it believes other agents will pursue and obtain before it can. Note that the deliberative agent does maintain representations of other agents' intentions, i.e, of its beliefs about other agents' intentions, to avoid having to process the intentions of all other agents again and again at every cycle. That way intentions only need to be processed again when items are collected or collisions occur.

5 Experimental Method and Prerequisites

Ideally, we would like to compare the performance of the above agent kinds with the best possible solutions for each configuration. In practice, however, this is not feasible, because the cost of computing the optimal solution is too high for configurations with large numbers of agents, items, and obstacles. Our approach to the problem, therefore, is to perform a series of "yardstick" experiments for different numbers of agents, obstacles, and items, for which the optimal solution can be practically computed. This allows us to see the relationship between the different agent types and how they compare to the optimal performance. We then perform much more extensive experiments that do not include the optimal solution (or even the deliberative agent performance, because they quickly become too expensive, as well) in order to demonstrate that the collection strategies implemented by the reactive agent types scale well to much larger numbers of entities.

However, before we can begin the yardstick experiments, we must determine reasonable parameters for the gain values of the schema-based architectures of the reactive agents. Therefore, we conducted experiments to map out performance in a "parameter space" for both reactive and reactive-predictive agent types. The idea is to find the gain settings that provide the best performance (across all numbers of agents tested), and use those settings for comparison with deliberative and optimal agent types. For experiments in 0-obstacle environments, the gain value for items was varied from 10 to 50 in steps of 10, providing five datapoints for each agent type. For 5-obstacle environments, the gain for items was varied from 10 to 100 and the gain for obstacles was varied from 0 to -50, both in steps of ten providing 60 datapoints for each agent type. The values we use in both the yardstick and extended experiments are the results of this exhaustive search of parameter-space (a total of 650 experiments, each consisting of 38 experimental runs). Note that once we find the best values for the MOCT, we do not need to perform this preliminary step again; the values will be valid for all instances of the MOCT.

5.1 The Simulation Setup

The simulation environment consists of a continuous, limited two-dimensional surface of 800 by 800 units, populated with collectible items and the different kinds of agents, where the objects to be collected are randomly distributed in a subregion of 720 and 720. Agents always move at a constant speed until they come to a collectible item, which they then pick up. When all items in the environment have been gathered, the simulation ends and the number of cycles executed is recorded. The simulation is ended prematurely if agents fail to collect all items within 10,000 cycles, in which case they "fail the task".

Each experimental run is started with a fixed number of agents (A) and a fixed number of items ($C = 10$) placed at random locations within the environment and is finished as soon as the last item is collected. The performance measure used (and depicted in the figures below) is the time it took the agents to collect all items. To be able to compare different agent kinds, the same initial conditions are used for all agent kinds for each run (i.e., the initial locations of agents and the locations of collectible items are the same).

Agents	Reactive g_c					Reactive-Predictive g_c				
	10	20	30	40	50	10	20	30	40	50
1	1948.34	1948.29	1948.29	1948.32	1948.29	1949.84	1949.84	1949.84	1949.84	1949.84
2	1389.39	1388.21	1398.63	1399.0	1403.74	934.083	934.25	934.25	934.306	934.167
3	974.921	976.368	953.368	962.974	964.684	608.703	608.865	608.892	608.892	597.73
4	922.947	869.921	902.29	900.553	905.974	532.278	532.306	531.972	532.083	531.889
5	770.816	756.053	769.263	762.868	759.026	433.595	421.919	419.541	421.27	420.919

Table 1. Parameter space results (average cycles to completion) for Reactive and Reactive-Predictive agents in 0-obstacle environments. The global best value of g_c is 20 for Reactive agents, 50 for Reactive-Predictive agents.

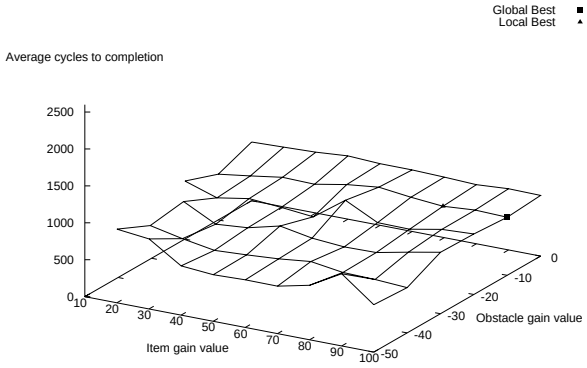

Figure 4. Reactive factor-space for $A = 5$, $C = 5$. The local best is the combination of g_c and g_b at which this particular value of A had the lowest average cycles to completion, whereas the global best represents the combination with the best performance over all values of A (1 through 5). For Reactive agents $g_c = 100$, $g_b = -10$ is the global best configuration.

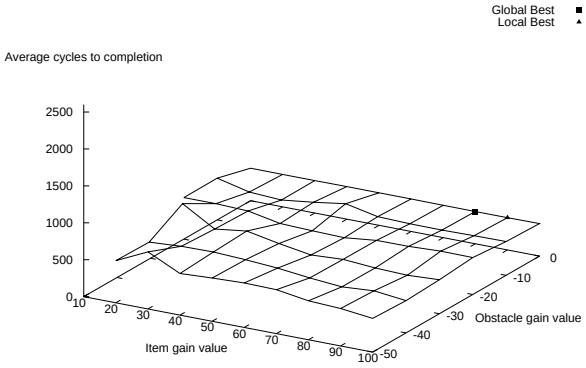

Figure 5. Reactive-Predictive factor-space for $A = 5$, $C = 5$. The local best is the combination of g_c and g_b at which this particular value of A had the lowest average cycles to completion, whereas the global best represents the combination with the best performance over all values of A (1 through 5). For Reactive-Predictive agents $g_c = 80$, $g_b = 0$ is the global best configuration.

All results are averaged over 38 runs to be able to average out effects of initial positions. So, for each value of A, 38 initial conditions are generated and each agent type is tested with A agents in each of the initial conditions.

5.2 Determining the Best Gain Values

Table 1 gives the results (in average cycles to completion) for the parameter-space experiments in 0-obstacle environments. These results indicate that the value of g_c (i.e., the item gain value) does not have a tremendous effect; the average cycles to completion is virtually the same across all values of g_c, indicating that for each number of agents, the strength of attraction to items does not play a major role. We identify the global best g_c for each agent type by finding the g_c with the minimum summed average cycles to completion for all values of A. For Reactive agents, the value is 20, and for Reactive-Predictive agents it is 50.

Figure 4 is representative of the results in five obstacle environents for normal reactive agents. It maps out the performance space for $A = 5$ given various combinations of g_c and g_b (the item gain and obstacle gain, respectively). Similar graphs could be made for $A = 1$ to 4, however these are omitted for space.

The first thing to note is that some data points are missing. For these combinations of g_c and g_b, none of the 38 experimental runs sucessfully completed within 10,000 cycles, so there were no results to take the average of. The

missing points correspond to agents with weakly positive item gains and strongly negative obstacle gains. In short, these agents were too strongly repelled by the obstacles in the environment and too weakly attracted to the items they were to collect to accomplish their goal. However, as the number of agents in the environments was increased, the number of gain value settings that failed to complete even one experimental run decreases; when $A = 5$ (Figure 4), only three combinations fail completely, whereas ten such combinations failed completely in the single-agent environments.

Each datapoint that is graphed depicts the average cycles to completion for the 38 experimental runs with those g_c and g_b, discarding runs that did not complete. Like the zero-obstacle results, aside from a few peaks and troughs, the results are very similar across all parameter settings, with a slight tendency to decrease as the gain values increase. "Local best" configurations were identified for each value of A (i.e., there is a local best for one agent experiments, one for two agent experiments, etc.), but for comparison with other agent types, a "global best" configuration was also identified. We identify the best-performing configuration based on two factors:

- Only configurations that completed more than 35 experimental runs for every value of A are considered in determining the global best. The idea here that the global best should be able to complete most of the time, otherwise it may be specialized to just those few environments in which it did complete.

- Among the remaining configurations, the minimum summed average cycles to completion for all values of A was taken to be the global best.

Thus, the global best is the configuration that had the best performance (minimum summed average cycles to completion for all A) of all configurations that completed more than 35 experimental runs for all numbers of agents. The combination identified as the global best for reactive agents in five obstacle environments is $g_c = 100$, $g_b = -10$.

Figure 5 presents similar results for reactive-predictive agent types. Once again, missing points are those gain combinations in which no experimental run completed. The local best is shown, along with the global best ($g_c = 80$, $g_b = 0$), chosen for comparison with the other agent types. Note that the obstacle factor can be 0 in this case, since agents still have collision avoidance mechanisms that will prevent them (in most cases) from running into obstacles. In this particular case it turned out that the collision avoidance mechanism alone was sufficient to deal with obstacles. Like the reactive results, there is a slight trend to higher gain values of both kinds, but no dramatic valleys that indicate a clearly superior configuration of g_c and g_b.

FUNCTION OptimalSolution($agentlist$,$itemlist$)
$optimaldistance = impossibly\_long\_distance$
$assignlist = gen\_all\_assign(agentlist, itemlist)$
for all $S \in assignlist$ **do**
  $worstdistance = 0$
  for all $A \in agentlist$ **do**
    $bestdistance = impossiblylongdistance$
    $permutationlist = gen\_all\_perms(S(A))$
    for all $P \in permutationlist$ **do**
      if $O = intervening\_obstacle(A, P)$ **then**
        $distance$ =MinObstPath(A, O, P)
      else
        $distance = calculate\_distance(A, P)$
      end if
      if $distance < bestdistance$ **then**
        $bestdistance = distance$
        $bestpath = append(A, P)$
      end if
      $move\_agent\_to(A, P)$
    end for
    if $bestdistance > worstdistance$ **then**
      $worstdistance = bestdistance$
    end if
    $append(worstpath, bestpath)$
  end for
  if $worstdistance < optimaldistance$ **then**
    $optimaldistance = worstdistance$
    $optimalpath = worstpath$
  end if
end for

Figure 6. The algorithm for OptimalSolution

5.3 The Optimal Solution

As a benchmark against which to compare the performance of the four agent types, a program that finds the best solution for a given distribution of agents, items, and obstacles was implemented. The premise for the optimal solution is an agent type that can communicate with its peers, can compute as much as it wants, and can see everything. At the beginning of the collection task, one agent is responsible for computing the best solution and communicating to each of its peers the set of items each is responsible for collecting along with the order in which they are to be collected.

The best solution is found by exhaustively enumerating all possibilities to find the one that yielded the shortest time to completion, as shown in Figure 6. Each assignment of items to agents is checked and for each assignment every collection order is checked. The algorithm computes the shortest Hamiltonian path in which some agent "visits" (i.e., collects) each item. As part of this computation, the algorithm checks if an obstacle is blocking the direct route, in which case it recursively considers paths going to the

Agents	Optimal		Reactive		Reactive-Predictive		Deliberative		Deliberative-Predictive	
1	1260.89	± 49.99	1948.29	± 121.36	1949.84	± 121.39	1486.82	± 63.63	1489.68	± 63.56
2	639.45	± 25.59	1388.21	± 114.08	885.0	± 56.33	1058.58	± 93.15	860.26	± 56.87
3	431.79	± 16.67	976.37	± 118.71	582.0	± 39.81	827.37	± 145.57	601.66	± 48.97
4	343.89	± 18.95	869.92	± 118.04	503.89	± 61.37	740.97	± 113.39	541.97	± 55.5
5	290.08	± 13.72	756.05	± 92.99	409.84	± 36.38	622.47	± 88.12	412.87	± 33.94

Table 2. Average cycles to completion (with confidence intervals for $\alpha = 0.05$) for 0-obstacle environments (A from 1 to 5, $C = 10$)

FUNCTION MinObstPath(A,O,P)
$R = move\_agent\_to(A, rightof(O))$
$rightdist = calculate\_distance(A, R)$
if $N = intervening\_obstacle(R, P)$ **then**
  $rightdist = rightdist + min\_obst\_path(R, N, P)$
else
  $rightdist = rightdist + calculate\_distance(R, P)$
end if
$L = move\_agent\_to(A, leftof(O))$
$leftdist = calculate\_distance(A, L)$
if $N = intervening\_obstacle(L, P)$ **then**
  $leftdist = leftdist + min\_obst\_path(L, N, P)$
else
  $leftdist = leftdist + calculate\_distance(L, P)$
end if
if $rightdist < leftdist$ **then**
  $return(rightdist)$
else
  $return(leftdist)$
end if

Figure 7. The algorithm for MinObstPath

left and to the right of the obstacle, taking the shorter one (Figure 7). For each assignment, the shortest collection order is then computed for each agent and its assigned items. Then the longest agent's completion time is chosen as the best time required by the current assignment to perform the task. The shortest completion time of any assignment is the best time overall. Note that it is not necessary to consider other agents as obstacles when sensory range is unlimited, because, while there may be cases in which agents' paths cross, there will not be cases in which they cross at the same time; if that were the case, it would be faster for the agents to "swap" their objectives, thus eliminating the cross. It is possible for agents to make conflicting decisions when sensory range is limited, and in that case a method for recovering would be required.

The "yardstick" results given below include the results from the optimal solution program. The numbers given are the average of the outcomes of the 38 initial conditions used by the other agents. Note that these results measure only time to completion and do not include the substantial computation time required to compute the best solution.

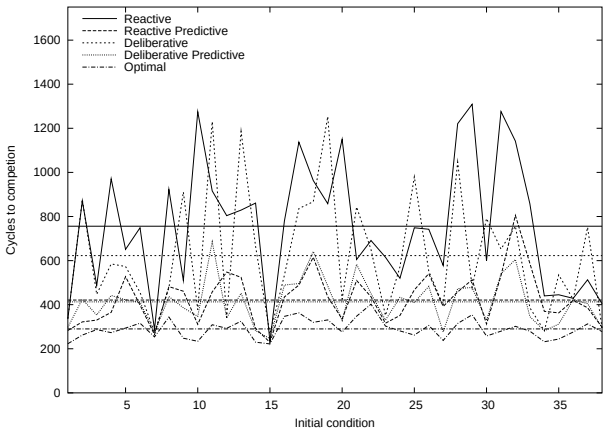

Figure 8. Five agent performance in zero obstacle environment (straight lines indicate average for each agent type, $C = 10$)

6 Experiments and Results

Here we present first the results of the "yardstick experiments", in which the performance of all four agent types are compared with the optimal performance. Then we present extended results of experiments with reactive agent types on large-scale MOCTs.

6.1 "Yardstick" Experiments, Results, and Analyses

Having found the best gain values for reactive and reactive-predictive agents, we can now proceed to compare all agent types. Table 2 compares the average performance (including confidence intervals for alpha of 0.05) of optimal agents, reactive agents with $g_c = 20$, reactive-predictive agents with $g_c = 50$, deliberative agents, and deliberative-predictive agents. Figure 8 graphs the five-agent performance of each agent type for all 38 experi-

Agents	Optimal		Reactive		Reactive-Predictive		Deliberative		Deliberative-Predictive	
1	1262.05	± 50.09	1953.21	± 112.05	2050.11	± 128.04	1503	± 64.91	1505.79	± 64.65
2	640.03	± 25.49	1317.66	± 115.39	908.74	± 58.55	1080.18	± 99.85	856.03	± 53.59
3	432.03	± 16.68	937.21	± 101.18	645.82	± 45.27	832.05	± 143.52	612.13	± 49.11
4	344.05	± 18.95	888.05	± 115.93	506.89	± 45.63	731.26	± 105.51	539.53	± 53.6
5	290.53	± 13.73	785.68	± 104.4	425.03	± 37.9	626.42	± 90.0	411.69	± 30.65

Table 3. Average cycles to completion (with confidence intervals for $\alpha = 0.05$) for 5-obstacle environments (A from 1 to 5, $C = 10$)

ments, making it possible to compare performance for each set of initial conditions. The average of the 38 experimental runs is also graphed for each agent type. When only one agent is allocated to the collection task (Table 2, row 1), the performance of both reactive agent types is virtually identical, as is the performance of both deliberative agent types. The small differences between reactive and reactive-predictive are due to the different values of g_c. Deliberative types perform significantly better than reactive types on average. In all multi-agent environments, on the other hand, reactive-predictive agents outperform normal deliberative agents on average. In fact, three, four, and five agent tests show reactive-predictive agents performing similarly to deliberative-predictive agents.

Turning to the individual results in Figure 8, we find that predictive types exhibit smoother behavior across all initial conditions; non-predictive agent types have a tendency to experience spikes in some environments. These spikes indicate the presence of "conflicts" during the corresponding experimental runs. When two agents arrive at an item at the same time, they enter a cycle in which they reflexively retreat from one another and return to the item to collect it. The distance that each agent retreats is determined probabilistically, so these conflicts eventually resolve themselves when one agent returns before the other does, but they may go through this cycle several times before resolution is accomplished. The predictive mechanism eliminates these conflicts altogether. By the time the number of agents allocated is five (Table 2, row 5), predictive agents' performance is fairly close to that of the optimal solution, whereas non-predictive agents averaged more than twice as many cycles to complete the task.

Table 3 compares the performance of optimal agents, reactive agents with $g_c = 100$ and $g_b = -10$, reactive-predictive agents with $g_c = 80$ and $g_b = 0$, deliberative agents, and deliberative-predictive agents in environments with five obstacles (note that the reactive-predictive agents perform well despite the fact that obstacles have zero weight; this is possible because of the retreat reflex described above). The initial conditions for these experiments are the same as for the zero-obstacle experiments,

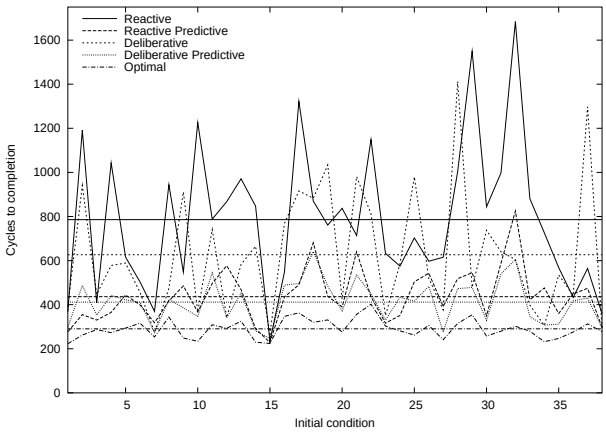

Figure 9. Five agent performance in five obstacle environment (straight lines indicate average for each agent type, $C = 10$)

with the exception of the added obstacles (i.e., the agents and the items are in the same initial positions). The difference between reactive and reactive-predictive agents in the one-agent tests is more pronounced, but is again due to the differing gain values. The individual test results are very similar to those in zero-obstacle environments; comparing Figure 9 with Figure 8, we see similarly shaped curves for each agent type, with spikes tending to occur in corresponding experimental runs, but of differing magnitudes. Similar comparisons can be made for all numbers of agents, but are not depicted here. Conflicts can be effected several ways by the introduction of obstacles into the environment. Obstacles can alter the timing of agents arriving at an item, causing an agent to take longer because it must navigate around an obstacle. This can lead to one agent arriving slightly later, eliminating a conflict that was present or introducing one that was not. Alternatively, an obstacle can prevent an agent from targetting a particular item because the path is too long or the repulsive force of the obstacle outweighs the attractive force of the item. If that item was the source of a conflict, the conflict has been eliminated. However, the agent may then target another item that does lead to a con-

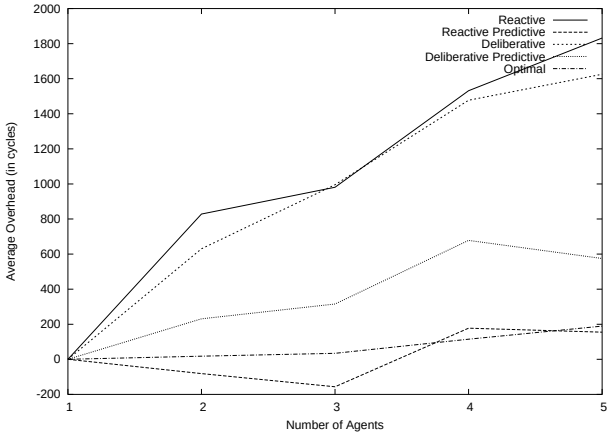

Figure 10. "Coordination" overhead for agents in zero-obstacle environments

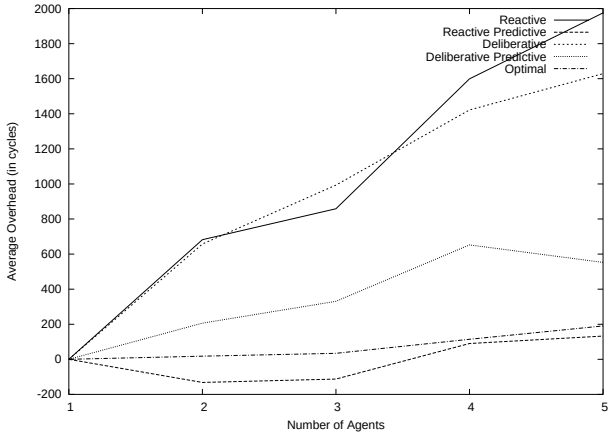

Figure 11. "Coordination" overhead for agents in five-obstacle environments

flict, introducing a new one. Overall, the average performance is similar to zero-obstacle results, with the exception that deliberative-predictive agents tend to hold onto a slight edge over reactive-predictive agents. Reactive-predictive agents, however, still perform significantly better than normal deliberative agents, on average.

One of the most interesting outcomes of this work is the emergence of what could be called "coordinated behavior" among independently acting agents using very simple predictive mechanisms. Of course, if one wants to characterize coordination as "the process by which an agent reasons about its local actions and the (anticipated) actions of others to try and ensure the community acts in a coherent manner" [14], then these agents' actions are not coordinated, as predictive-reactive agents do not reason about anything (after all, they are still reactive in the sense defined previously, but merely filter their perceptual input using a simple criterion).

Regardless of one's definition of "coordination", the experiments here demonstrate that reactive-predictive agents (i.e., reactive agents equipped with the "perceptual filter") perform as well as "coordinated deliberative agents" that reason about other agents' intentions and maintain representations of various environmental states.

It is interesting to look at the results also from another perspective, namely that of the "overhead" introduced by letting multiple agents cooperate. If one agent takes time t' to perform a given task, then two agents should roughly take half the time, three agents a third, and so forth. In general, A agents should take t'/A time to complete the task. Based on this intuition we define the notion of "cooperation overhead" (in a multi-agent task) as $t * A - t'$ (where t is the time it takes A agents to complete the task and t' is the time it takes one agent to complete it). Cooperation overhead thus defined is a measure of how efficiently additional agents are utilized in a multiagent task.

Given this definition, it turns out that reactive-predictive agents have a lower overhead than the predictive deliberative agents, sometimes even significantly lower (see Figures 10 and 11), even though their performance is in some cases slightly worse. This implies that increasing the number of reactive-predictive agents increases the performance relative to an increase in the number of predictive deliberative agents. Most notably, the reactive-predictive agents' overhead is, in fact, even better than the overhead of the optimal solution, which is yet another point to prove that the relative performance of individual reactive-predictive agents improves with the number of agents participating in the task.

Consequently, reactive-predictive agents are better at utilizing the multiagent system than any other agent kind. Furthermore, both predictive kinds are significantly better than the non-predictive kinds, which have a large overhead due to their lack of coordination, which demonstrates the effectiveness of the prediction mechanism.

6.2 Large-Scale Experiments, Results, and Analyses

We turn now to our investigation of the large-scale instances of the MOCT. These large-scale experiments represent the performance of reactive and reactive-predictive agents under more realistic conditions. The size of the environment is increased to 1600 by 1600, the number of agents is varied from 10 to 50, and the number of collection items is varied from 10 to 200. In addition, environments with 50 obstacles were tested. The reactive and reactive-predictive agents in 0-obstacle environments have $g_c = 20$ and $g_c = 50$, respectively. In the 50-obstacle environments, reactive agents have $g_c = 100$ and $g_b = -10$,

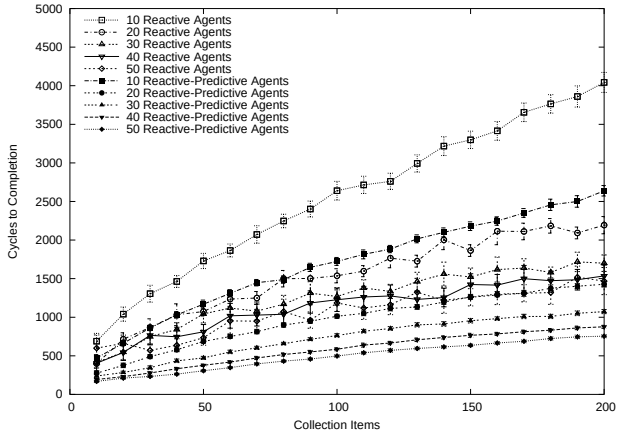

Figure 12. Results for reactive and reactive-predictive agents for extended tasks, no obstacles (A from 10 to 50, C from 10 to 200; errorbars depict 95% confidence intervals for $\alpha = 0.05$)

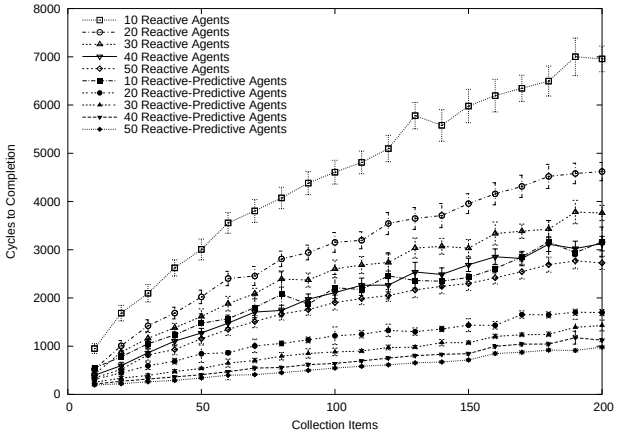

Figure 13. Results for reactive and reactive-predictive agents for extended tasks, 50 obstacles (A from 10 to 50, C from 10 to 200; errorbars depict 95% confidence intervals for $\alpha = 0.05$)

while reactive-predictive agents have $g_c = 80$ and $g_b = 0$. These are the values determined as the best overall in the parameter-space experiments. The remainder of the experimental setup is the same. Performance is again measured as number of cycles to completion, with a ceiling of 10,000, at which point the task is said to have failed (the same as in the yardstick experiments).

The optimal solution is not included for the large-scale experiments, for it is computationally intractable, as already mentioned. Furthermore, neither variety of deliberative architecture is examined here for two reasons: (1) the deliberative architectures are computationally very expensive because of the involved exponential planning mechanism, and more importantly (2) because the yardstick experiments showed that the performance of the reactive agents is comparable to that of the deliberative ones. Hence, the enormous computational overhead for running deliberative agents in these large scale environments is not necessary.

We now present the results of 400 experiments (each of which consisted of 40 experimental runs); as the number of entities in the environment increased, the amount of (wall-clock) time to complete the simulations became quite large, approaching 24 hours for 50 reactive-predictive agents collecting 200 items in environments containing 50 obstacles on a dedicated 2.4 GHz Pentium IV Linux system.

Figure 12 presents the results of these experiments for 0-obstacle environments. The relative performance of each agent type is similar to their performance in the "yardstick" experiments. For $C = 10$, it takes ten agents of each type roughly the same amount of cycles to complete the task on average as it took five agents in the "yardstick" experiments. This is because the size of the environment has quadrupled. Given the same number of agents, reactive-predictive agents perform significantly better than non-predictive reactive agents overall. In fact, only the tests with ten reactive-predictive agents performed worse than *any* number of normal reactive agents tested. Introducing obstacles into the environment only increases the advantage of reactive-predictive agents (Figure 13; note the difference in scale from Figure 12). Maneuvering around obstacles is expensive, especially for agents implementing schema-based architectures (deliberative agents plan efficient routes around obstacles, whereas reactive agents take a more "trial-and-error" approach). Since non-predictive reactive agents choose their targets poorly, many of their paths are wasted, including paths that require detours around obstacles. This increases the total amount of travel in the environment, leading to still more encounters with obstacles. Thus, the problem is self-reinforcing: the further an agent travels in the environment, the more it is penalized by obstacles, and hence the further it needs to travel.

7 Discussion

The results from the "yardstick" experiments indicate a performance ordering among the four agent types: normal reactive agents perform worst, followed by normal deliberative agents. The predictive versions of both agent types are the best, performing very similarly, although deliberative-predictive agents have a slight advantage in 5-obstacle environments. The performance of agents employing the predictive mechanism is clearly superior to the performance of their non-predictive counterparts. The fact that the reactive-predictive agents outperform even deliberative agents in

multi-agent environments is strong evidence that the prediction mechanism is effective; previous results [23] indicate that deliberative agents outperform reactive agents in most environments unless they are assesed a cost for the deliberative mechanisms. Analysis of the efficiency with which additional agents are used highlights the source of predictive agents' advantage in multi-agent environments.

However, absolute performance measures alone are insufficient for comparing and evaluating architectures; the computational cost of the architectures must also be taken into account. Determining the relative cost of the four agent types examined here is a matter of analyzing the computational complexity of the actions each must perform at each step. The reactive agents must sum the force vectors from each other agent A, any obstacles B that might be in the environment, and each (remaining) collection item C. Thus, their computational cost is linear ($O(|A| + |B| + |C|)$). The complexity of the predictive extension is ($O(|A| \cdot |C|)$), since the algorithm checks every collection item ($|C|$) for every agent A to see which is closest to each agent. The deliberative agents must update their memory of the relative locations of all items and obstacles in the environment ($O(|B| + |C|)$) at each time step. In addition, whenever a goal item is collected or a collision is immanent, the A^*_ϵ planner must be executed, which is of exponential complexity [19] both in time and memory. This yields the following increasing order of cost: non-predictive reactive agents, reactive-predictive agents, and deliberative agents of both types.

Taking both cost and performance into account, some tradeoffs become clear. First, given the fact that the cost of the predictive mechanism is dominated by the cost or the planner and the fact that the deliberative-predictive agents outperform normal deliberative agents by a significant factor, there appears to be no reason to deploy non-predictive deliberative agents. Deliberative-predictive agents appear to have greater "value" than their non-predictive counterparts without increasing the overall cost appreciably. Comparing the values of deliberative-predictive and reactive-predictive agents in these environments is also straightforward: their performance is similar, however, the deliberative-predictive agents pay much more in terms of computation to achieve that performance. Reactive-predictive agents are the better value here.

The difficult task is to determine where the relative value of the non-predictive reactive agents falls. The previous two comparisons were between alternatives of like cost and differing performance, and between alternatives of like performance and differing cost, respectively. Between non-predictive reactive agents and any of the other types, however, the comparison is of unlike cost *and* unlike performance. The performance is lower than any of the alternatives, but so is the cost. In the end, it is (1) the time-critical

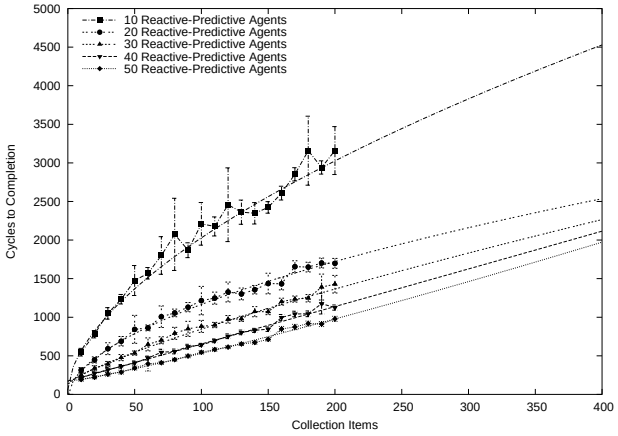

Figure 14. Projected performance (from 210 to 400 items) of reactive-predictive agents for extended tasks, 50 obstacles (A from 10 to 50, based on simulation results for C from 10 to 200)

nature of the specific application of the collection task and (2) the available computational resources on an agent that will determine the relative value of the non-predictive reactive agents. When time is very important (e.g., in the military example of finding supplies), the extra cost of the predictive mechanism may be worthwhile. When time is less important, but computational cost matters (e.g., imagine a group of room-cleaning robots trying to decide which mess to clean next), one may choose to sacrifice performance in favor of cost savings.

The results of the large-scale experiments indicate that the predictive mechanism allows reactive architectures to scale very well to more realistically demanding environments. Performance is worse in environments cluttered with obstacles, but overall the performance curves are very similar. As in the parameter-space experiments, these agents operate with unlimited vision ranges. This means that the number of items needing to be checked for each agent is very high, due to the simplistic nature of the prediction mechanism. While in the normal case where sensory range is limited, the number of items checked will be much more tractable, it is infeasible to extend the reactive-predictive results to more items or more agents when evaluating agents' best-case performance. To get an idea of how performance would scale for more items, we project performance curves based on the actual results obtained. In Figure 14, these results are extrapolated out to give projected performance values for higher values of C (up to 400). The projection was accomplished by fitting the data to the function $g(x) = (x/d)^e + f$ via nonlinear least-squares method. Table 4 gives the constant fits. The projected performance to more items continues to scale well, near linear in the worst

Agents	d	e	f
1	0.00027	0.59	64.16
2	0.00026	0.55	-27.8
3	0.025	0.79	135.75
4	0.22	1.0099	174.25
5	0.62	1.16	169.53

Table 4. Constant fits for $g(x) = (x/d)^e + f$ shown in Figure 14

case, providing further evidence for the scalability of the preactive-predictive architecture. Note that these results and projections are for a fixed-size environment; as the number of items increases, the density increases. Therefore, travel times do not increase as much as they would if the area were increased at the same time. Furthermore, one would expect the time required to complete the task to level off at some point, as the area becomes saturated with items and the agents are able to collect them just by moving in any direction.

While more complex and accurate predictive mechanism are certainly possible, the performance of the current simple version seems sufficient given its performance relative to the optimal solution in the "yardstick" experiments. Consequently, it is not obvious that further refinements would be worthwhile (to determine that, the tradeoff between the additional complexity of the prediction algorithm and the increased performance would need to be assessed).

The results we have obtained for the reactive-predictive agents look very promising and confirm our conjecture that simple mechanisms (without communication) might be sufficient for the coordination of multiple agents in a multi-agent task. In the above case, a simple reactive mechanism plus a perceptual filter performed as well as a complex deliberative mechanism. The extended results demonstrate that the predictive mechanism does scale well with the number of items, the number of agents, and the number of obstacles, as predicted.

8 Related Work

Currently, no online approximation algorithms are available for multi-object collection tasks. However, approximation algorithms exist for several related problems, most notably the Traveling Salesman Problems [7]. The *Traveling Salesman Problem* requires the salesman to make a Hamiltonian cycle of n cities, minimizing the cost (i.e., the distance traveled). Essentially, it is the shortest Hamiltonian cycle problem. A *Hamiltonian cycle* in an undirected graph is a simple cycle that visits all vertices in the graph.

There are many approaches to approximating a solution to the Traveling Salesman Problem [12, 15]. Constructing the tour by appending the best remaining node (by some heuristic, such as the nearest node to the current end of the tour) to the current path is one approximation. Insertion techniques attempt to find the best node to add to the tour at any location, not only at the current end of the tour. Once tours have been completed, they can be improved via heuristics that modify the tour in some way until no improvements are achieved. Other approaches include tabu search, simulated annealing, genetic algorithms, and neural networks. The reactive and deliberative agents described here can be viewed as implementing the greedy append method, with the predictive mechanism serving to improve the heuristic that determines which item is closest.

The *multi-Traveling Salesman Problem* requires some member of a sales team to visit each city. Makespan (i.e., the overall time to completion) is to be minimized, and salesmen can travel to cities in parallel. Most of the examples below approach the problem in this way.

Gavish and Srikanth provide a branch-and-bound optimal solution method for the multiple Traveling Salesman Problem starting and ending at the same city [11]. Due to the branch-and-bound technique, their technique was able to solve problems that the other techniques they reviewed would run out of space on, and their performance in time was as good as other techniques on Euclidean problems, one to two orders of magnitude better on non-Euclidean problems than other techniques.

Bugera describes in detail the characteristics of the *no-depot min-max 2-Traveling Salesman Problem* [4]. A no-depot Traveling Salesman Problem is one in which the multiple salesmen do not start and end at the same position; each has its own starting and ending position. This is similar to the MOCT, except that MOCT agents do not need to return to their original positions.

Delivery scheduling refers to the problem of deciding how best to order and assign delivery tasks to a finite set of delivery trucks. Similar problems include schoolbus routing and dial-a-ride scheduling. Wang et al describe a system that divides the delivery area into zones based on past performance and then solve each zone optimally [28]. The system is shown to be effective at solving long-distance delivery problems, where the cost of delivery can vary substantially from zone to zone.

An ant colony system has been proposed to solve the delivery problem in which demands for pickup are dynamically generated [18]. "Pheromone trails" created by simple search agents are used to identify efficient paths through the search space. The authors report performance improvements of, on average, 4.37% over local search methods on a suite of benchmarks, as well as success applying the technique to real-world situations.

Applegate and colleagues provide a distributed branch-and-bound solution to the multi-Traveling Salesman Prob-

lem [1]. This technique allowed them to verify the optimality of a solution to a newspaper delivery problem found earlier.

A cyclic transfer algorithm, in which demands are first assigned to agents and are then transfered from one agent to another closer agent, provides efficient assignments of tasks to salesmen [27]. This method yields paths up to 11% shorter than other algorithms on the classical vehicle routing problem.

These approaches apply to the delivery scheduling problem because it is a Traveling Salesman Problem, with additional constraints (e.g., truck capacity). The main difference between the multi-Traveling Salesman Problem and the problem being solved in the MOCT is that in the multi-Traveling Salesman Problem, the starting and ending vertices must be the same. What the optimal MOCT algorithm solves could be called the multi-shortest Hamiltonian path problem, with makespan minimized. A *Hamiltonian path* in a graph is a simple path that visits every vertex in the graph. Given an assignment of items to agents, the optimal algorithm solves the shortest Hamiltonian path problem for each agent. The graph is fully connected and undirected (with the exception of the edges connecting the agents with items initially). MOCT agents do not need to return to a base, hence their solutions may look quite different from the ones computed by the above methods (e.g., see Figure 2). Furthermore, often in other versions of the Traveling Salesman Problem a common source is assumed for all salesmen (i.e., a depot). The MOCT places agents randomly in the environment.

Similar differences can be found between the MOCT and job-scheduling tasks [10], printed circuit board assembly tasks [13], and flying probe schedulers [16].

The SHAC (Shared Activity Coordination) project [6] explores the role of communication in rover scheduling and cooperation. SHAC's emphasis on the need for communication as a means of coordination requires further investigation in light of the performance of non-communicating predictive agents presented above.

Finally, much interesting and useful work has been done in the area of foraging and cooperation, in both robotic and simulated environments (e.g., [17, 5, 8, 26, 2]); this research focuses on the utility of the predictive mechanism as an implicit cooperative mechanism.

9 Conclusions and Future Work

This paper proposes a solution to the multi-agent problem of collecting items in an environment in the shortest time possible. While the problem can be solved optimally by exhaustively searching every assignment of items to agents and every permutation of collection order within each assignment, the computational cost of such a solution makes it practically infeasible. Furthermore, offline algorithms such as this are not able to handle dynamic envionments in which items can move or new items can be generated; in such circumstances, the optimal algorithm would have to recompute the entire solution. The goal was to find an online solution with relatively low cost and respectable performance relative to the optimal solution in a static environment.

The mechanism we implemented incorporates a simple predictive component into reactive and deliberative architectures, allowing agents with the component to implicitly coordinate their actions to avoid duplicated effort. We conducted extensive sets of experiments to determine the effectiveness of prediction in both reactive and deliberative architectures and demonstrated that low-cost predictive reactive agents show a very high level of performance even in large-scale MOCTs, thus obviating the need for computationally expensive deliberative architectures. Moreover, the results suggest that communication (and all the computational overhead involved it) to achieve coordinated agent behavior might not be necessary for a large class of multiagent tasks (such as the MOCTs).

We are currently investigating different relaxations of the multiagent object collection task, in which we believe the full power of the simple mechanism will come to bear, such as (1) restricting the sensory range of agents (thus relaxing the condition on *complete information about the environment*), and (2) allowing new items to appear and old items to disappear during the collection task (thus making the environment more dynamic). Both extensions emphasize the need for online algorithms and preliminary results indicate that the proposed simple reactive-predictive mechanism shows even greater performance compared to other more expensive mechanisms.

References

[1] D. Applegate, W. Cook, S. Dash, and A. Rohe. Solution of a min-max vehicle routing problem. *INFORMS Journal on Computing*, 14(2):132–143, 2002.

[2] E. G. Araujo and R. A. Grupen. Learning control composition in a complex environment. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 1996.

[3] R. C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotic Research*, 8(4):92–112, 1989.

[4] V. Bugera. Properties of no-depot min-max 2-traveling-salesmen problem. In S. Butenko, R. Murphey, and P. Pardalos, editors, *Recent Developments in Cooperative Control and Optimization*. Kluwer Academic Publishers, 2003.

[5] J. Carmena and J. Hallam. Improving performance in a multi-robot task through minimal communication. In *Proceedings of the 7th Symposium on Intelligent Robotic Systems (SIRS)*, July 1999.

[6] B. J. Clement and A. C. Barrett. Continual coordination through shared activities. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (AAMAS '03)*, July 2003.

[7] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[8] A. Drogoul and J. Ferber. From Tom Thumb to the Dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, 1992.

[9] T. Estlin, G. Rabideau, D. Mutz, and S. Chien. Using continuous planning techniques to coordinate multiple rovers. *Electronic Transactions on Artificial Intelligence*, 4:45–57, 2000.

[10] A. Frangioni, E. Necciari, and M. G. Scutella. A multi-exchange neighborhood for minimum makespan machine scheduling problems. Technical Report TR-00-17, University of Pisa, Pisa, Italy, 2000.

[11] B. Gavish and K. Srikanth. An optimal solution method for large-scale multiple traveling salesmen problems. *Operations Research*, 34(5):698–717, 1986.

[12] B. Golden, L. Bodin, T. Doyle, and W. S. Jr. Approximate traveling salesman algorithms. *Operations Research*, 28(3):694–711, 1980.

[13] W. Ho and P. Ji. Component scheduling for chip shooter machines: a hybrid genetic algorithm approach. *Computers and Operations Research*, 30:2175–2189, 2003.

[14] N. R. Jennings. Coordination techniques for distributed artifi cial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. Wiley, 1996.

[15] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John-Wiley and Sons, Ltd., 1997.

[16] A. B. Kahng, G. Robins, and E. A. Walkup. Optimal algorithms ofr substrate testing in multi-chip modules. In J. D. Cho and P. D. Franzon, editors, *High performance design automation for multi-chip modules and packages*. World Scientifi c Publishing, 1996.

[17] D. McFarland. Towards robot cooperation. In *From Animals to Animats 3. Proc. of the Third International Conference on Simulation of Adaptive Behavior*, 1994.

[18] R. Montemanni, L. Gambardella, A. Rizzoli, and A. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. In *Second International Workshop on Freight Transportation and Logistics*, 2003.

[19] J. Pearl. A^*_ϵ—an algorithm using search effort estimates. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 4, pages 392–399, 1982.

[20] G. Rabideau, T. Estlin, S. Chien, and A. Barrett. Working together: Centralized command sequence generation for cooperating rovers. In *Proceedings of the IEEE Aerospace Conference (IAC)*, March 1999.

[21] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 1995.

[22] M. Scheutz. The evolution of simple affective states in multi-agent environments. In D. Cañamero, editor, *Proceedings of AAAI Fall Symposium*, pages 123–128, Falmouth, MA, 2001. AAAI Press.

[23] M. Scheutz and P. Schermerhorn. Steps towards a theory of possible trajectories from reactive to deliberative control systems. In R. Standish, editor, *Proceedings of the 8th Conference of Artificial Life*. MIT Press, 2002.

[24] M. Scheutz and P. Schermerhorn. Many is more but not too many: Dimensions of cooperation of agents with and without predictive capabilities. In *Proceedings of IEEE/WIC IAT-2003*. IEEE Computer Society Press, 2003.

[25] E. Spier and D. McFarland. Possibly optimal decision making under self-suffi ciency and autonomy. *Journal of Theoretical Biology*, 189:317–331, 1998.

[26] E. Stergaard, G. Sukhatme, and M. Mataric. Emergent bucket brigading - a simple mechanism for improving performance in multi-robot constrainedspace foraging tasks. In *Proceedings of the 5th International Conference on Autonomous Agents*, May 2001.

[27] P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

[28] H. Wang and D. Xue. An intelligent zone-based delivery scheduling approach. *Computers in Industry*, 48:109–125, 2002.