Cost-balanced Cooperation Protocol in Multi-agent Robotic Systems

Fang-Chang Lin and Jane Yung-jen Hsu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

Abstract

This paper proposes a cooperation protocol based on the cost-balanced strategy for the Object-Sorting Task in multi-agent robotic systems. The protocol coordinates agents for carrying objects to destinations efficiently and effectively. Each agent autonomously makes subjective optimal decision, then the coordination algorithm resolves their conflicts by balancing the load, which is measured in terms of ***cost****. Since coordination can be performed simultaneously with agent movement, it incurs very little overhead. The protocol is efficient because every agent runs the same algorithm to obtain the common results without further communication. Implementation of the protocol is realized on a distributed modular agent architecture for design simplicity, flexibility, and reactivity. Experimental results have shown that 1) the protocol has better performance than a previously proposed help-based cooperation protocol, 2) the protocol is flexible, and 3) the protocol can effectively utilize the agent power to achieve linear and superlinear speedup in most cases.*

1. Introduction

For designing multi-agent robotic systems, parallel performance and cooperation requirements of tasks are the main interests and consideration. Some tasks can be executed in parallel for better performance. Tasks such as painting a wide wall or cleaning rooms can be partitioned into several subtasks, each of which is then assigned to an agent. If all agents work independently, i.e. no resource contention or goal conflicts, the performance is linearly speeded up. Otherwise, if there is resource contention (e.g. short of paintbrushes or brooms) or conflicts (e.g. agents have different favorite colors), the performance speedup will be sublinear. Besides, some other tasks require explicit cooperation among the agents, e.g. conferences, moving a heavy equipment, etc. This paper focus on a specific multi-agent task, the Object-Sorting Task, which has the characteristics of both parallelism and cooperation.

There has been much research in multi-agent robotic systems. Some of them proposed mechanism to create the foundation of multi-agent robotic systems. For example, Fukuda's CEBOT system [5] showed the self-organizing behavior of a group of heterogeneous robotic agents; Asama et al. proposed the ACTRESS architecture [4] for connecting equipment, robots and computers together to compose autonomous multi-agent robotic systems by designing underlining communication architecture; Wang proposed several distributed functional primitives for distributed robotic systems [11]. Other researchers worked on solving multi-agent tasks, e.g. Mataric addressed the problem of distributing a task over a collection of homogeneous mobile robots [9]; Arkin et al. assessed the impact on performance of a society of robots in a foraging and retrieval task when simple communication was introduced [2,3]; Alami et al. coordinated the multiple-robot navigation with a plan-merging paradigm for the task of transporting containers in harbors [1]; Lin and Hsu provided a fully distributed cooperation protocol for the Object-Sorting Task in multi-agent robotic systems [7,8].

Either centralized or distributed approaches can be employed to solve a multi-agent task. A centralized model uses a powerful agent to plan and schedule the subtasks for every agent. However, for tasks with NP complexity, the centralized approach is impractical. Furthermore, the control agent must be powerful enough to achieve satisfactory performance. High complexity of system design, high cost and low reliability are the other drawbacks of centralized approaches.

On the other hand, a distributed approach decreases design complexity and cost, while increasing reliability. Agents are autonomous and equal. An agent plans for itself and communicates with the others in order to accomplish the global task. Because every agent interacts directly with the environment, it is reactive. However, each agent has only local knowledge of the task and the environment. Hence, it cannot make the best decisions of the global task alone. Furthermore, negotiation or social cooperation rules for conflict resolution are required to coordinate among them. A distributed approach for dealing with multi-agent tasks with a modular and reactive agent architecture was proposed in [7]. Conflicts are re-

0-8186-7267-6/96 $05.00

solved by designing social rules into the *help-based cooperation protocol* (HCP) that coordinates agent actions.

In order to improve the performance of HCP, a *cost-balanced cooperation protocol* (CCP) has been developed. It has the advantages of both centralized and distributed approaches. The Object-Sorting Task (OST) is used to demonstrate the approach. Section 2 introduces and discusses the OST, and summarizes the HCP. The CCP and its implementation are described in Sections 3 and 4. Simulation and experimental results are shown in Section 5.

2. The Object-Sorting Task (OST)

The OST was formally defined and discussed in [8], which showed that the OST is a NP-complete problem for finding the optimal performance. This section describes its definition, complexity, and problems. Finally, the previous proposed HCP is summarized.

2.1 Definition

Let $O=\{o_1,...,o_M\}$ be a set of stationary objects that is randomly distributed in a bounded area. Every object, $o_i=(l_i,d_i,n_i)$, is associated with an initial location l_i, a destination location d_i, and the number n_i of agents for movement. An object o_i can be moved only if there are at least n_i agents available to move it. Let $R=\{r_1,...r_N\}$ be the set of agents and n_{max} be the maximal number of agents to move any single object, an object-sorting task can be completed only if N is not less than n_{max}. Agents search for objects and move them to their destinations. When all the objects have been moved to their destinations, the task is finished.

Typical application examples of OST are foraging and retrieval tasks, explosives detection and handling, *automatic guided vehicle* dispatching for components transportation in manufacturing systems, surveillance, etc. In this research, the following assumptions were made. The agents are *homogeneous* mobile robots with the basic capabilities for *navigation*, *obstacle avoidance*, *object recognition*, and *object handling*. The agents have no prior knowledge about the environment, nor the other agents. Finally, the communication system is *reliable*, and the agents communicate with the others by broadcast or point-to-point channel.

There are many operation models associated with different strategies to do the task, e.g. an object is assigned to the agents at random, movement of the objects are controlled by a predefined precedence, actions of the agents and objects are all central-controlled, etc. The performance evaluation is based on the *cost* to accomplish the task. Generally speaking, an agent searches for objects, coordinates its object schedule with the other agents, and cooperates with the other agents to move objects. So, an agent spends its cost on search, coordination and cooperation. The cost of the applied operation model is the maximum cost among all agents' costs.

Algorithm ***Optimal-OST.***

1. FOR each permutation of objects $o_1,...,o_M$ DO
   (1) Let the object sequence is $s_1,...s_M$.
   (2) FOR each $s_i=(l_i,d_i,n_i)$ in $s_1,...s_M$ DO
      a) Let *Comb(r)* represent all the r-combinations from the N agents $\{r_1,...r_N\}$.
      b) For each combination of *Comb*(n_i), assign the agents to the object s_i.
   (3) For each assignment of agents for $s_1,...s_M$, calculate its cost.
2. The optimal solution is the object sequence with agent assignment, which has minimal cost.

Finding an optimal solution of OST is a high complexity problem. Algorithm *Optimal-OST* describes this search process. It lists all object sequences, then assign agents to the objects for each sequence. There are $M!$ object sequences from M objects, and $C(N,n_i)$ agent assignments for each object o_i, where $C(k,r)$ is the number of r-combinations from k. Hence, the search space is $M!(\Sigma C(N,n_i))$, $1 \le i \le M$. Let $E(C(N,i))$ be the mean of $C(N,i)$, $1 \le i \le N$. The search space becomes $M!(E(C(N,i)))^M$. Table 1 lists the order of search spaces for different number of objects when N=10 agents.

***Table 1**: The order of search spaces for N=10 agents*

M	10	20	30	40	50	100
Order	10^{26}	10^{58}	10^{92}	10^{128}	10^{164}	10^{358}

2.2 Problems

The OST consists of two kinds of work. One is the *search* work: searching for objects, another is the *object processing* work: carrying objects to destinations. The former has search problem while the latter has coordination and deadlock problems. Moreover, load balance, cost balance and task termination are their common problems. To solve the OST, these problems need to be addressed.

1. *Search*. All the objects must be found in order to attack the task. The most intuitive way is to let agents search the entire area so that they can find all the objects. How do they search efficiently ?
2. *Coordination*. How do the agents coordinate for assigning themselves to a found object ? That is, for a given object, which agents should work together to move it ? and who makes the decision ?

3. *Deadlocks*. When each agent autonomously selects an object and none of the selected object has enough agents for movement, a deadlock occurs. A coordination solution must have solved it.
4. *Load balance and cost balance*. How can the workload be distributed to the agents ? The smaller the maximum cost of the agents is, the better performance is. Intuitively, more balanced cost may reduce the maximum cost among the agents.
5. *Termination* . How do every agent realize the global task has been accomplished ?

2.3 Help-based Cooperation Protocol (HCP)

Our previous work proposed a help-based cooperation protocol for the OST. The protocol was implemented in a modular, reactive agent architecture as showed in Fig. 1. The search module, communication module and motion module are all finite state automata. They share the global state information and change the state information according to their state functions. The search module searches and identifies objects. The motion module performs the function of object movement alone or with other agents. The communication module communicates with the other agents in order to cooperate with them.

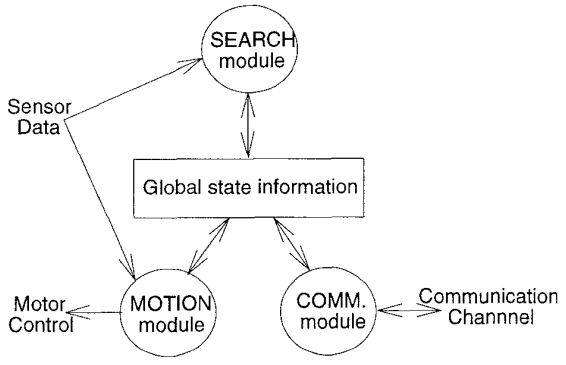

***Fig. 1**: Agent architecture of the Object-Sorting Task*

The working area is equally partitioned into disjoint N subareas, and each subarea is assigned to an agent. Advantages of the equally partition are load balance, cost balance, and parallel performance of the *search* work. Each agent exhaustively searches its subarea, requests help from the others once it has found a large object, and selects its partners. After there are enough agents arriving at the found object, they carry the object to its destination. The cooperation protocol defines how and when an agent requests help, how and when the other agents offer help, and deadlock handling in order to coordinate the agents for accomplishing the task. With this approach, the objects are partitioned into several parts. Each part of the objects is treated by a specified agent that requests help and determines which agents will be its partners for every object in this part of objects. Since each agent is autonomous, simultaneous selection of partners by several agents may cause a deadlock. Several deadlock handling schemes were also provided in the HCP.

The above approach utilized simultaneous subarea search for parallel performance, and help-based cooperation protocol for solving the coordination, deadlocks, and termination problems. The protocol was realized into each agent's state transition function so that agents reacted and cooperated quickly. Nevertheless, the performance of OST can be further improved. Object distribution affects the workload balance of subareas which may further affect the problem of cost balance. Unbalanced workload may cause some agents to finish their subareas before the others. Furthermore, subtask assignments and help-based strategies impose agents to request help and wait once finding a large object, which limited the global knowledge of agents. Hence, agents cannot make cost-optimal decisions. The CCP has been developed to address this issue. The results of CCP is better than HCP. In addition, parallel search performance is reserved and deadlock handling becomes natural and easy.

3. Cost-balanced Cooperation Protocol (CCP)

Like HCP, the CCP uses equal partition of the working area for parallel search. However, only the *search* work is partitioned instead of partitioning objects into subtasks. Instead of requesting help once an object is found, every object information is broadcast and saved when received. With the overall object information, each agent makes its subjective optimal schedule and coordinates with the others for the global schedule. In order to efficiently resolve conflicts, social rules are employed into a coordination algorithm from which decisions are obtained by each agent independently so as to minimize coordination overhead.

3.1 Coordination points

A coordination point specifies when the agents should start coordinating the object schedule for the found objects. The simplest way is to coordinate the object schedule after having searched the entire area. Since the CCP requires every agent to save the information of found objects, the storage demand expands with the increasing number of objects.

On the other hand, coordination can start during subarea search under storage constraints or in order to reduce storage requirements. Coordination can be activated by a certain number of objects found, a part of area searched, or a fixed time interval if there is a common clock. The measurement is called *coordination interval*

(CI), e.g. coordination is activated every 100 time units (CI=100) or every 20 objects found (CI=20). The criteria of a certain number of objects found was employed in this research since it is more natural to the OST.

It is possible that more than one object are found at the same time. Hence, the actual number of objects found may be more than the CI when a coordination process begins. Agents must have the same object set for each coordination process. Under such circumstance, either the exactly CI number of higher priority objects or all found objects attend current coordination. Object ID can help to determine object priorities.

3.2 Object identification

In order to clearly identify objects during coordination, a consistent representation of objects is necessary. Generally, assigning a unique ID for each object is appropriate. However, various assignment methods should be suitably applied under different situations. Several methods are discussed:

1) Predefined attributes: Object IDs are predefined so that it can be identified by agents. An object can be identified from its outside look, size, symbol, or even initial location. For example, a two dimensional coordinate can determine the order of object IDs by comparing x-coordinate first then y-coordinate.
2) Coordinator: Where there is no way to specify predefined object IDs, this method can be applied. Every object found is reported to the coordinator, then the coordinator assigns its ID and broadcasts to the others.
3) Other more complicated distributed algorithms [6,10] concerning the order of discrete events may be employed when the coordinator method is unsuitable.

3.3 Coordination algorithm

Every found object is broadcast, and every agent stores all received object data. After reaching a coordination point or having searched the entire area, agents start coordination for determining the object schedule by a cost-balanced coordination algorithm. When the found objects have been scheduled, the current coordination process ends. The scenario continues until all the objects have been found and scheduled.

Coordination among agents are achieved by coordination strategies. There is no polynomial time algorithm for finding an overall optimal object schedule because the OST is a NP-complete problem. The utilized strategies are based on the balanced cost and greed. Algorithm *Coordination* depicts them. All agents invoke this algorithm to coordinate their individual object schedules.

At first, every agent selects its cost-optimal object and broadcasts its selection. Among all the selected objects, the best cost-optimal object is scheduled, i.e. enough agents are assigned to it. Finally, every assigned agent updates its cost and location. Agents repeat the process until all objects are scheduled.

Algorithm *Coordination.*

1. Every agent selects its current cost-optimal object, and broadcasts to the others. Let the selected objects be $s_1, ..., s_k$, $1 \le k \le N$.
2. Select the best cost-optimal object, *Opt*, among the selected objects $s_1, ..., s_k$.
   (1) FOR each object in $s_1 ... s_k$ DO
   - Let the current object be $o_i=(l_i,d_i,n_i)$.
   - FOR each agent r_j DO
     Let CA_j be the accumulated cost of r_j , and CR_j be the cost of r_j to reach o_i.
     $C_{ij} = CA_j + CR_j$
   - M_i = the n_i-th smallest cost from $C_{i1}, C_{i2}, ..., C_{iN}$

   (2) *Opt* = the object with the minimum M_i
3. Assign agents to *Opt*, and update data.
   (1) Let *Opt* be $o_p=(l_p,d_p,n_p)$.
   Assigned agents are the n_p agents with smaller cost C_{pi}, $1 \le i \le N$.
   (2) IF I am one of the assigned agents THEN
   - append *Opt* to my object schedule
     accumulate my cost with (M_p + the cost from l_p to d_p)
   - my location = d_p
4. Repeat from step 1 until all object attended in this coordination have been scheduled.

Obviously, this approach is cost balance because the selection strategy is to choose the agents with smaller cost. Besides, deadlock-free is guaranteed by the two facts: there is a consistent object order in every agent's object schedule and there are enough agents assigned to every object.

Since every agent selects one object for coordination at a time, the number of selected objects is at most N. For each selected object, it requires time complexity $O(N)$ to calculate all costs of agents. So, the cost for scheduling an object is $O(N^2)$. Because there are M objects, the complexity of algorithm *Coordination* is $O(MN^2)$ for overall consideration. However, the complexity is $O(MN)$ when CI=1 for general speaking.

Every agent runs the same coordination algorithm to obtain its object schedule. Thus, the algorithm can be implemented on a dedicated coordinator agent only. In this manner, every agent sends its decision to the coordinator instead of broadcast. The coordinator runs the algorithm and sends the results to the others for updating their schedules, costs, and locations.

There are several differences between HCP and CCP.

1. Task partition: HCP partitions objects and assign each part to an agent while CCP partitions the search task only.
2. Object assignment: An object is assigned to which agents is determined by request/offer protocol in HCP while by coordination protocol in CCP.
3. Handling object found: An agent deals with an object at a time (request help, select partners, then move object) in HCP while an agent broadcasts the object information in CCP when an object is found. For the CCP, coordination of the object schedule is delayed till the number of objects found having reached the CI.
4. Knowledge: HCP makes decision with local knowledge other than global knowledge in CCP.

4. Cooperation architecture

The cooperation architecture employed to realize cooperation protocols in multi-agent robotic systems is an agent architecture with embedded cooperation protocol. It is reactive, modular, and cooperative.

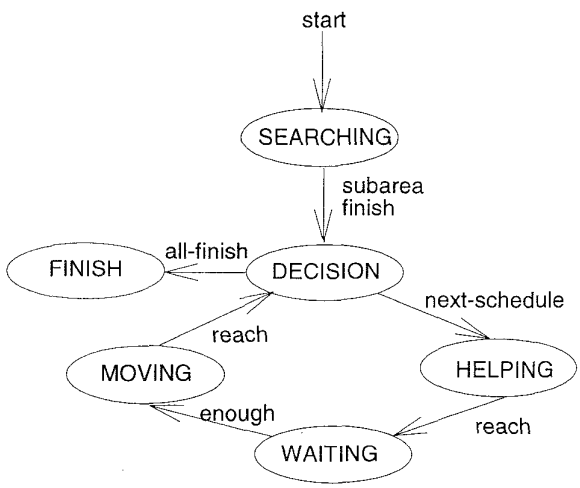

Fig. 2: State transition diagrams of coordination-based cooperation protocol

The agent architecture is a finite state automaton (FSA) composed of several functional modules that cooperate and coordinate each other through the state transition function. The FSA implements the cooperation protocol which coordinates the multi-agent robotic system. Every agent working on the architecture cooperates and coordinates with the others to achieve common goals. Every functional module is also a FSA which is a subset of the global FSA. Modularized decomposition simplifies the design complexity of each module.

The agent architecture of the OST is showed in Fig. 1 which have been utilized by the HCP and CCP approaches. The transition diagram of CCP is showed in Fig. 2 in which circles represent states and arrows represent transitions. At first, agents start searching in SEARCHING state, broadcast found object data. After finishing search, an agent broadcasts this message and enters DECISION state. During searching, agents will start a coordination process if they reach a coordination point, then exchange local optimal selections and run *Coordination* algorithm to obtain the overall object schedule. Coordination and object handling can perform simultaneously. In DECISION state, an agent picks its next object schedule and changes to HELPING state for going toward the target object. When an agent arrives at the object, its state becomes WAITING. If there are enough agents for object movement, they enter MOVING state and carry the object to its destination. They each repeat the movement cycle until all assigned object tasks having been accomplished.

If there is no assigned schedule when an agent in DECISION and not all objects are scheduled, the agent will wait for an assigned object task. This is for considering the cases when agents are in coordination or specially when the CI = M, i.e. coordination will start only if all the area has been searched. An agent enter FINISH state when there is no remaining object schedule.

The global FSA is further decomposed into three FSAs. The three functional modules execute concurrently and take care only its own subset transition functions. Fig. 3 shows their transition diagram.

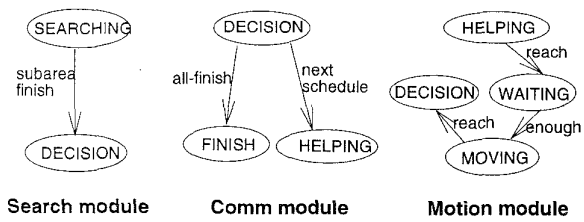

Fig. 3: State transition of functional modules

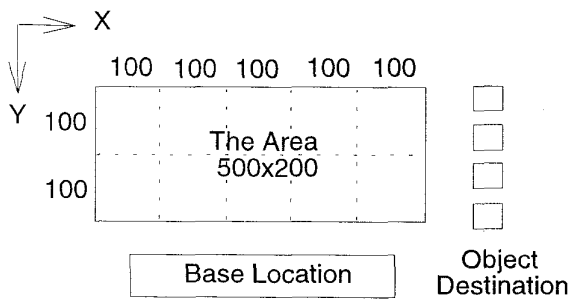

Fig. 4: The simulation map for 5×2 partition

5. Simulation

The object-sorting task was simulated in the simulator developed on Sun workstation with graphic user interface showing task execution. The simulator is a testbed for testing different operation models on the object-sorting task.

5.1 Simulation environment

The working area was equally partitioned into N subareas, and each subarea was assigned to an agent. The area is represented in a two-dimensional coordinate system, e.g. 500×200. The partition of the area is represented by a $m \times n$ notation, m in x-axis direction and n in y-axis direction. In the experiment, the area was 500×200. Fig. 4 shows the map used in the simulation for N=10 agents with 5×2 partition. The partitions for different number of agents N used in the experiment are shown in Table 2.

***Table 2**: Partition and the number of agents*

N	10	20	30	40	50
Partition	5×2	5×4	6×5	10×4	10×5

The performance was evaluated with the number of time steps. The following code fragment sketches the actions performed by the agent at each time step.

FOR each agent i ,
  do the search module of agent i;
  do the motion module of agent i;
  do the communication module of agent i.

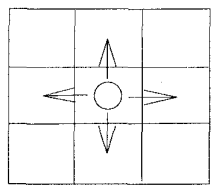

***Fig. 5**: Move directions*

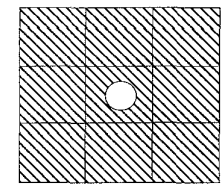

***Fig. 6**: Sensor ranges*

The area is a grid area, and the agents can only move to one of the four locations from a location in a time step as the Fig. 5. The agents can sense the objects located in the grids adjacent to the agents as the Fig. 6.

The agents may use any exhaustive search method to search their subarea. In this experiment, the row-major order method was used. They start search from the initial location, the (0,1) location of their subarea, to the right side boundary, and search the first three rows which are under sensor range. When they have reached the right side boundary, they search the second three rows and change the search direction from right to left. When they has reached the left side boundary, they change the search direction from left to right and start searching the next three rows. Agents repeat the search process until the subarea has been searched. Each agent will return to its base location after finishing its object schedule.

Each object was randomly generated for its destination, initial location, and the number of required agents. This experiment generated 10 sets of objects for each number of objects M using n_{max}=10. All object sets were run at N=10, 20, 30, 40, and 50. The number of objects used in the experiment are M= 1, 10, 20, 30, 40, 50, 100, and 200 objects.

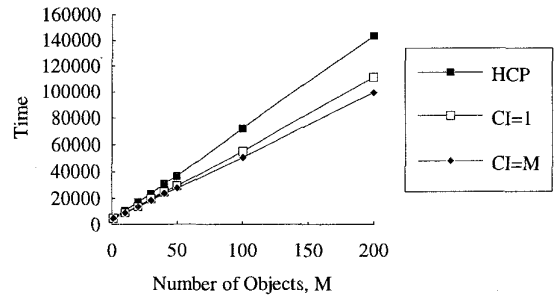

***Fig. 7**: Comparison of execution time for the HCP and CCP when N=10*

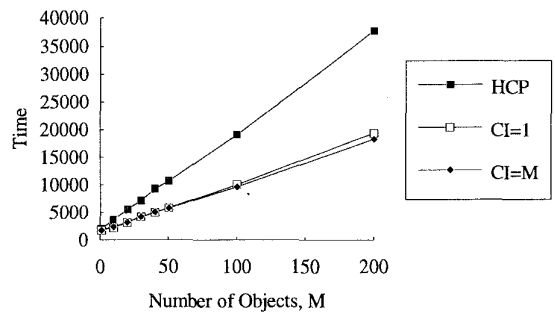

***Fig. 8**: Comparison of execution time for the HCP and CCP when N=50*

5.2 Simulation results

The experiment was performed by varying the number of agents N, and the number of objects M. For a given number M of objects, the execution time was the average of the execution time from the 10 generated object sets of M. The experiment included three parts: the performance

comparison between the HCP and CCP, the comparison of different *coordination intervals*, and the speedup effect of the CCP.

First, experimental results showed that the CCP is better than the HCP. Fig. 7 and 8 typically present the execution time comparison between the HCP and CCP under N=10 and 50 agents. The CCP results include both CI=1 and CI=M, i.e. coordination activated once finding an object or after having searched the entire area. No matter what value of CI is chosen, the CCP is always superior to the HCP. Through global coordination and cost-balanced strategy, the CCP can efficiently perform the object-sorting task.

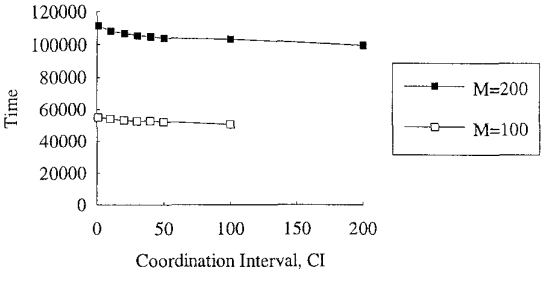

***Fig. 9**: Execution time for different coordination interval when N=10*

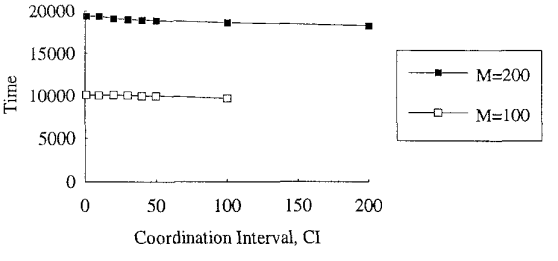

***Fig. 10**: Execution time for different coordination interval when N=50*

Second, the effect of different CI is showed in Fig. 9 and 10 where CI=1, 10, 20, 30, 40, 50, 100, and 200. Larger CI can make better decision due to holding more information for coordination, and smaller CI requires more coordination processes. Hence, larger CI has better performance than smaller CI generally. On the other hand, smaller CI requires less storage and is faster in generating the object schedule for movement. In addition, the difference between CI=1 and CI=M becomes smaller when there are more agents such as N=50 in Fig. 10. Hence, the CCP is flexible in choosing CI.

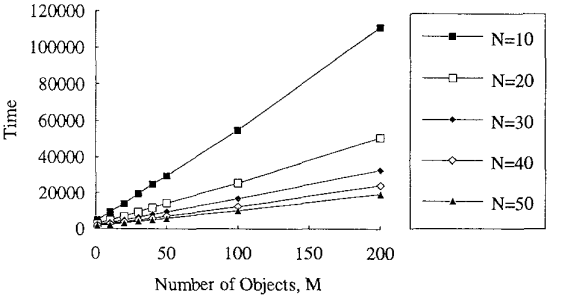

***Fig. 11**: The execution time for different number of objects when N=10,20,30,40, and 50*

When the generated object sets are applied to $N \geq 10$, the execution time decreases with the increasing number of agents as showed in Fig. 11. It shows that the protocol can effectively utilize the increased agent-power.

Table 3 further lists the speedup ratio relative to the execution time unit of N=10. Let the execution time of N be C_N. The speedup ratio S_N is defined as:

$$S_N = (C_N / N)(10 / C_{10})$$

***Table 3**: The speedup ratio relative to N=10.*

Number of object, M	Number of agents, N				
	10	20	30	40	50
1	1	0.82	0.67	0.61	0.53
10	1	0.95	0.88	0.85	0.82
20	1	1.01	0.96	0.92	0.88
30	1	1.04	1.02	0.99	0.93
40	1	1.05	1.03	1.04	1.00
50	1	1.04	1.04	1.04	1.00
100	1	1.08	1.09	1.12	1.09
200	1	1.09	1.14	1.15	1.15

For most cases with M>30, the speedup is linear or superlinear. There is obvious improvement on objects movement because more agents make more objects movement in parallel. For example, assume there are three objects require 5, 6, and 7 agents respectively. They will be moved in sequence when N=10 while moved in parallel when N=20. The speedup ratio is 3. On the other hand, for small number of objects, e.g. M=1, more agents decrease the *search* time due to smaller subarea. But, there is no obvious improvement on object processing time since there is no significant increasing parallelism. Hence, the results also provide a useful reference when adding more agents for speedup. Where there is a superlinear speedup,

adding more agents can be beneficial. Under sublinear speedup condition, the benefit of adding more agents is not as significant.

6. Conclusion

In this paper, we provided a cost-balanced cooperation protocol (CCP) for coordinating multi-agent robotic systems to do the Object-Sorting Task. The protocol is 1) more efficient than the *help-based cooperation protocol*, 2) flexible in determining the criteria when a coordination process should start for the object schedule, and 3) effective in agent utilization.

By combining the centralized and distributed approaches, this paper has demonstrated a model for multi-agent tasks that offers efficient reactivity as well as the performance of global coordination. The cooperation architecture presented in this paper can be further generalized to handle more multi-agent tasks in distributed agent systems.

7. References

[1] R. Alami, F. Robert, F. Ingrand, and S. Suzuki, "Multi-robot Cooperation through Incremental Plan-Merging", *Proc. of IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 2573-2579.

[2] R. C. Arkin, T. Balch and E. Nitz, "Communication of Behavioral State in Multi-agent Retrieval tasks", *Proc. of 1993 IEEE International Conference on Robotics and Automation*, GA, May 1993.

[3] R. C. Arkin and J. D. Hobbs, "Dimensions of Communication and Social Organization in Multi-Agent Robotic Systems", *Proc. Simulation of Adaptive Behavior 92*, Honolulu, HI, Dec. 1992.

[4]. H. Asama, A. Matsumoto, and Y. Ishida, "Design of an Autonomous and Distributed Robot System: ACTRESS", *Proc. of IEEE/RSJ International Workshop on Intelligent Robots and Systems '89*, Tsukuba, Japan, Sept. 1989, pp. 283-290.

[5] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss, "Structure Decision Method for Self Organizing Robots Based on Cell Structure - CEBOT", *Proc. of IEEE International Conference on Robotics and Automation*, Scottsdale Arizona, 1989, pp. 695-700.

[6] L. Lamport, "The Mutual Exclusion Problem: Part II -- Statement and Solutions", *JACM*, Vol. 33, No. 2, Apr. 1986, pp. 327-348.

[7] F. C. Lin and J. Y.-j. Hsu, "Cooperation and Deadlock-Handling for an Object-Sorting Task in a Multi-agent Robotic System", *Proc. of IEEE Inter. Conf. on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 2580-2585.

[8] F. C. Lin and J. Y.-j. Hsu, "A Genetic Algorithm Approach for the Object-Sorting Task", *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, Vancouver, Canada, Oct. 1995.

[9] M. Mataric, "Minimizing Complexity in Controlling a Mobile Robot Population", *Proc. of 1992 IEEE International Conf. on Robotics and Automation*, Nice, 1992, pp. 830-835,

[10] J. Wang, "Establish a Globally Consistent Order of Discrete Events in Distributed Robotic Systems", *Proc. of 1993 IEEE ICRA*, GA, May 1993, pp. 853-858.

[11] J. Wang, "Operating Primitives Supporting Traffic Regulation and Control of Mobile Robots under Distributed Robotic Systems", *Proc. of IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May 1995, pp. 1613-1618.