Deadlock Handling in case of Mobile-agent System

Rashmi Priya

Abstract- This paper presents a deadlock-free cooperation protocol for an object-sorting task in a multi-agent system. First, the object-sorting task in a distributed robotic system is introduced and **a** cooperation protocol for the task along with the agent architecture is proposed. The agents are based on a homogeneous agent architecture that consists of search, motion, and communication modules coordinated through a global state. Second, the deadlock problem for the object-sorting task is addressed and several deadlock handling strategies are provided to guarantee the cooperation protocol **is** deadlock-free.

Keywords: Mobile Agents, Fault Tolerance, cooperation protocol, agent architecture

1. Introduction

A multi-agent robotic system uses multiple robots to solve problems by having them work in parallel. Space exploration, undersea construction, nuclear waste management, explosives detection and many other situations often require a multi-agent system that can achieve a common task by coordinating their behaviours.

Much research on multi-agent robotic system has begun to emerge. Fukuda's CEBOT system demonstrated the self-organizing behaviour of a group of heterogeneous robotic agents. Beni and Hackwood's research on swarm robotics demonstrated large scale cooperation in simulation. Brooks et al. developed the lunar base construction robots by using a set of reactive rules and based on the subsumption architecture. The work by Mataric addresses the problem of distributing a task over a collection of homogeneous mobile robots. The task performed by the robots is collective homing (moving toward a specified region). By using simple rules the robots can minimize interference caused from the collection of robots. Arkin has demonstrated that cooperation between robotic agents is possible even in the absence of communication. It simplifies the design of an agent because there is no communication between agents. On the other hand, it may be inefficient due to the lack of communication. Arkin et al. assessed the impact on performance of a society of robots in a foraging and retrieval task when simple communication was introduced. In a multi-agent robotic system, some tasks can be achieved by a single agent, e.g. cleaning up a region, searching for a target in an area, etc. System performance of these tasks can be improved if a task can be partitioned into many subtasks which can be done in parallel. Some tasks can only be done by a single agent, e.g. moving an object to cross a single-plank bridge on which only one agent is allowed at a time. Many other tasks in multi-agent system require cooperation among the agents and cannot be done by one agent alone, e.g. moving a large object which is not movable by any single agent alone. These tasks are multi-agent tasks. In a distributed environment, it is necessary to have a cooperation protocol that allows multiple agents to help each other in the problem solving process for multi-agent tasks. Furthermore, multi-agent tasks can be blocked if all the agents need and wait for help. This is so called deadlock. This paper addresses the cooperation and deadlock-handling for multi-agent tasks in multi-agent robotic systems. Section 2 defines an object-sorting task which is a multiagent task. The proposed system architecture and cooperation protocol for the task are summarized in Section 3. Deadlock problem is discussed in Section 4.

2. The object-sorting task

This section formally defines the object-sorting task. Let O={ol, ..., OM] be a set of stationary objects that is randomly distributed in a bounded area A. Every object, oi=(li,di,ni), is associated with an initial location Zi, a destination location di. and the number ni of agents for moving it. An object oi can be moved only if there are at least ni agents available to move it. Let R={rl, ... rN} be the set of agents and nm, 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 nmu. When, all the objects have been moved to their destinations, the. task is finished. For load-balancing consideration, this paper uses a uniform distribution model with a cooperation protocol. The bounded area is partitioned into disjoint subareas and each agent is assigned to a unique subarea. This paper further assumes the following: The agents are homogeneous mobile robots with the basic capabilities for navigation,

obstacle avoidance, object identification and object handling. The agents have no prior knowledge about the environment, or the other agents. So, the agents must search for the objects in their subareas. Finally, the agents communicate with the others by a broadcast or a point-to point channel.

3. Agent architecture and cooperation protocol

An agent architecture and cooperation protocol for the object-sorting task was proposed in [9], which is summarized in this section.

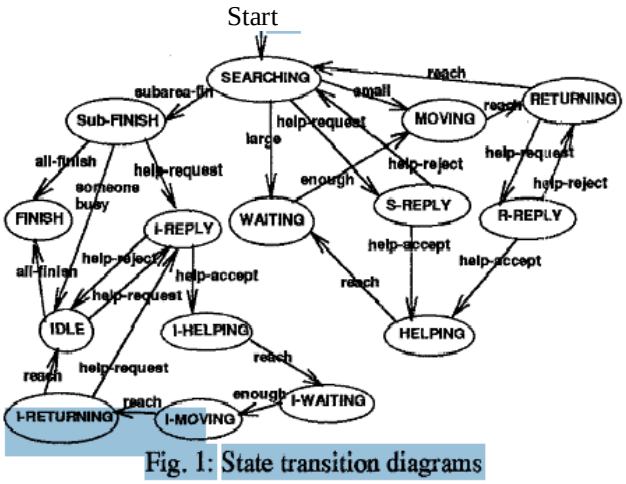

Fig. 1: State transition diagrams

The architecture of an agent contains search, motion and communication modules which are finite state automata and coordinated through a global state. The search module searches for the objects, identifies their destinations and required number of agents. The motion module performs the function of moving an object alone or with other agents to its goal. The communication module communicates with the other agents in order to cooperate with them. Fig. 1 shows the global state transition diagram. Each circle is a state and a thread represents the state transition driven by the event on the thread. For example, the search module searches for an object in the SEARCHING state and changes the state to MOVING if a small object is found, or to WAITING and broadcasts a help message if it finds a large object; the motion module changes the state from WAITING to MOVING and moves the object to its destination if there are enciugh agents to move the object; the communication module replies a will-help message when receiving a help message in SEARCHING, and
changes its state to S-REPLY. An agent comes into SUBFINISH state and broadcasts a sub-fin message after it has done its subtask, and enters IDLE state when receives busy from any agent whose subtask isn't finis h e dT. h e prefix I- is used to indicate an agent in a I- state has finished its subtask. An object is a small object if it can be moved by an agent alone; an object is a large object if it requires more than one agent to move. An agent broadcasting a help message when it finds a large object is called requiring-help agent. The other agents willing to help and replying to the requiring-help agent are called will-help agent. Multi-agent cooperation1 requires a communication architecture that allows an agent to communicate with the other agents in order to request for help or to offer help. Additionally, several strategies need to be considered as follows. In particular the strategies used for the object sorting task are described. 1) A when-help strategy determines when to help the other agents. An agent accepts a help message only when it is in SEARCHING, (I-)RETURNING or IDLE states, otherwise the message is queued for later processing.
2) A select-help strategy enables an agent to decide which requiring-help agent has; the most urgent need for help when there are many requiring-help agents. An agent selects the nearest to offer help.
3) A select-partner strategy let an requiring-help agent ri choose the suitable agents if there are more will-help
agents. The selected partners are called helping agents, and ri is called helped agent. An agent chooses the
nearest will-help agents as its partners.
4) A load-balancing strategy balances the system load among all the agents. It is considered in two parts. One
part is the equal partition of the area and the uniform distribution of the agents. Another part is embedded in
the select-partner strategy. When an agent needs help from n agents, there may be m will-help agents such
that m is greater than n. The load-balancing strategy enables the agent to select the first n nearest agents, who
will receive an accept message. The other will-help agents will be rejected by reject messages.
5) A deadlock free strategy handles the deadlock, in which all the agents are in WAITING or I-WAITING state.
First, the agents in (I-)WAITING state detect the deadlocked situation (which will be discussed in detail
in Section 5), then they broadcast blocked messages to exchange their state information and break down the
blocked situation by comparing their priorities. The lower priority agents will help the highest priority agent.

4.Deadlock Handling Schemes

Deadlock proposes three schemes to handle deadlocks for the object sorting task. Here is a deadlock situation using the above cooperation protocol. There are 4 agents R= {r1,r2,r3, r4} and two objects 01,02 with n1 = n2=3. If r1 finds

object 01 and r2 finds object o2 at the same time, r3 and r4 will receive help
message from both r1 and r2. If r3 chooses to help rI and r4chooses to help r2 according to the select-help strategy, all the agents will enter the WAITING state. A deadlock occurs. We say { o1 ,02} causes the system coming into a deadlock. Here is a fact coming from the strategies used.

Fact 1.

If a set of objects will cause the system coming into a deadlock, they are large objects and found at the same time.

Explanation:

Assume there is a set of object K={oi/oiε 0, riεR, oi was found by ri at time t i } such that the system comes into a deadlock. Because a small object requires only a single agent to move, it will not cause a deadlock. So oi is a large object if oiE K. For any oi,OjEK, i#j,

case 1: ti < tj.

Because rj was in SEARCHING before tj according to the state diagram. r. must have been a will-help agent to ri and rejected by ri according to the when help strategy and the state diagram. So ri had enough partners to move oi. The result is that oj should not be in L. So this case is impossible.

case 2: ti > tJ.

Similar to case 1, it is impossible also.

So ti =tj. That is, the large objects causing system blocked are found at the same time. Because the small objects will not cause a deadlock, we focus on the large objects. Let L~{ oi/oiε 0, rjER s.t. oi was found by rj at time t}, I Lt I=k, be a set of large objects, and rt(oi) denote the agent r. such that oi E Lt and oi was found by rj at time t. Gt={rt(oi)/oi ε Lt}. Remember oi=(li,di,ni), and ni is the required number of agent to move oi. Let Ft={rIrE R,rt} be the set of agents which don't find large object at time t. Because the objects causing a deadlock must be found at the same time, the following discussion uses a fixed t. Lt is referred to as L) rt (oi) is referred to as r(oi), and Gt is referred to as G. We assume N>= nmax. If k=I, the other agents can help the agent in G and the system will not come into a deadlock. So a deadlock may occur if k > 1. It is obvious that a deadlock will occur if for all oi E L, ni > IFI+1.

There are three approaches for handling deadlocks: deadlock prevention, deadlock avoidance, and deadlock detection. Which deadlock-handling approach is suitable greatly depends on the application. Based on the strategies provided in previous section and a little modification, we propose three schemes to handle deadlocks for the object sorting task.

4.1 Deadlock detection

In what follows, we will present the analysis for any agent in (I-)WAITING state to detect the deadlock situation.

Lemma 1 shows that a deadlock can be detected by checking the number of objects causing agents to stay in the (I-)WAITING states if global knowledge about the agent state is available. In a distributed environment, Theorem 1 allows each agent to identify a deadlock based only on local information. Let MTT be the maximum travel time between any two locations in the bounded area, and Wt ,=Io1 o E Lt, some agent is in WAITING or I-WAITING st& at time t+u due to o}, so Wt,O=Lt. Wt,u is referred to as Wu in the following discussion.

Lemma 1.

Given IW,I>O, the system is deadlocked if IWu+2MTTI= IWuI.

(That is, the system is not deadlocked iff IWu+2MTTI<IWuI

Proof

The only if part is trivial and obvious. Next we consider the if part: given IWu+2&=IW,I. The agents are partitioned into the following sets according to their current states, except the FINISH state which means the task is finished:

MOVE={rER, r's state is MOVING or I-MOVING}
HELP={r€R, r's state is HELPING or I-HELPING}
WAIT={rER, r's state is WAITING or I-WAITING}
AVAIL={rER, r's state is SEARCHING or RETURNING or I-RETURNING or IDLE}
TEh@={rER, r's state is S-REPLY or R-REPLY or I-REPLY or SUB-FINISH}

Each state in TEMP is a transit state and can be ignored. Given IWuI>O, i.e. WAIT set is not empty, if the system was not deadlocked at time t+u, the agents in AVAIL can offer help immediately and the agents in MOVE or HELP may offer help in the future. If they can help to move an object at time t+u+v other than enter WAIT, then IWu+vI<IWuI. We consider the case in each set starting from the time t+u:

1)HELP set:

When an agent in HELP set reaches the object, either it enters MOVE set if there are enough agent to move the object or enters WAIT set. Because an agent stays in HELP state no more than a MIT. So the maximum time for these agents to reach and move an object in W, is a MTT.

2)AVAIL set:

Every agent in the set is a will-help agent. If the agents in AVAIL are accepted by the requiring-help agents, they enter HELP set. So the maximum time for these agents to reach and move an object in Wu is a MTT.

3)MOVE set:
Because an agent stays in MOVE and enters AVAIL set no more than a MTT. After that, they enter HELP set immediately if there is any requiring-help agent. So the maximum time for these agents to reach and move an object in W, is 2 MU.So the maximum time for the agents not in WAIT to reach and move an object in Wu is 2 MT. On the other hand, they all enter WAIT set if they cannot move any object in Wu in 2MP. That is, the system is deadlocked if

IWU+2MTTI = IWuI.
Theorem 1.
When any agent stays in WAITING or I-WAITING state for 2(N-1)MTT, the system is deadlocked.
Proof:
The system is deadlocked if all the agents find large objects at the same time, i.e. IWoI= N. If it is not the case,
i.e. IWoI < N. From Lemma 1, the system is not deadlocked at t+2M7T if IW2& < IWoI. In the worst case: IWoI=N-1, IW2N-1 M d = 0 if the system is not deadlocked at t+Z(N-I)dT. That is, all the objects in L have been moved, and there is no agent in WAITING or IWAITING due to the objects.

So if an agent stays in (I-)WAITING state for 2(N-l)MTT, all the agents must be in (I-)WAITING state and the system is deadlocked. This scheme is very simple and effective. However, it may be inefficient because the deadlock detection time is proportional to the number of agents. The system performance will degrade when the number of agents increases. To avoid unnecessary waiting, let an agent r broadcasts an is-blocked message every 2 MTT when staying in (I-)WAITING, and the other agents not in (I-)WAITING reply with not-blocked message. If r receives any not-blocked, it keeps waiting, otherwise, it has detected a deadlock situation. This improvement can detect a deadlock quickly once a deadlock occurs. But, it introduces redundant message transmission.

4.2 Object priority

This scheme prevents deadlocks by the following parts: It assigns a unique priority to each object.
Select-help strategy is modified. An agent selects the agent having found the highest priority object.
Select-partner strategy is modified. An agent selects its partners by fist considering their states are not in (I-)WAITING, then shorter distance. When-help strategy is modified. An agent in (I-)WAITING state also can offer help if it is not the agent having found the highest priority object. An object has a higher priority if it is nearer to its destination. In our two-dimensional experimental environment, for example, the priority of an object is determined by comparing the following order:
1. the distance between the object and its destination,
2. x-coordinate of the object,
3. y-coordinate of the object.

Because there is no more than one object having the same (x,y) coordinates, different object has different priority. The scheme guarantees deadloclk prevention when an agent can offer an help. The last part is for the situation when ni >IFI+1 for all oi E L.
This scheme is very simple and easy for implementation. Nevertheless, many redundant messages are transferred for replying to the help coming from higher priority agents, or rejecting the will-help agents which are more than required.

4.3 Feasible sequence

This scheme utilizes the concept of feasible sequence and associated algorithms in order to guarantee that selecting an agent to offer help doesn't cause a deadlock. Meanwhile, it has additional advantages :
It eliminates the redundant messages transferred.
It is load-balancing.
It can improve the system performance.
Definition: feasible sequence.
A feasible sequence is a permutation sequence SI ,..., si ,..., sk of L, si €: L, 1s i 5 k, such that SI is moved to its destination by the agents in F and r(sl),then s2 is moved to its destination by the agents in F,r(sl), and r(s2), ... , imd finally sk is moved to its destination by all agents.
For example, assuming R:={r1,r2,r3,r4}L, = {ol,02},G ={r1,r2}, n1=3, n2=2, a feasible sequence may be 01.02 or 02.01. If n1=3, n24, the feasible sequence is 01.02 only.
Theorem 2.
Any deadlock may be caused by L can be avoided if there is a feasible sequence in L.
Proof:
Fact 1 states that L may cause a deadlock. However, there is a feasible sequence in L. Assume sI, ..., Sk is the feasible sequence, we can let S1 be moved to its destination first, then s2 , ... , and let sk be moved to its destination at last. Because all the elements in L are moved to their destinations, the deadlock caused by the set L is avoided.
Proof:
If there is a feasible sequence in L, the agents can moved these objects according to the sequence. Furthermore, they can distribute themselves to different objects such that more than one objects can be moved simultaneous and increase the performance. In fact, the proposed protocol includes the effect. An agent can reply to the agents in G according to the sequence r(sl), ..., r(sk). If it is rejected by r(sl), tryr(s1),etc. However, there are redundant messages transferred.

Algorithm load-balancing can eliminate the redundant messages.

If there is no feasible sequence in L, the select-help strategy must be modified to avoid coming into a deadlock.

In addition, the agents in G may need to exit WAITING state to help each other. For example, assume R=(r1 ,r2,r3,r4), L={ol,02}, G={r1 ,r2}, r(oI)=rI, r(o2)=r2, n1=4, n2=4, and the sorted sequence of L is O1 ,O2. Though there is no feasible sequence, both r3 and r4 can select r1 as the helped agent. Besides, r2 must exit WAITING state and go to help r1 to avoid a deadlock. After o1 having been moved to its destination, the agents can continue to move 02. In order to implement the scheme, the select-help strategy is modified to the following:

Step 1. If k = 1, the agent in G is the selected agent.
Otherwise, continue the next step.

Step 2. Use algorithm find-feasible-sequence to check if there exist a feasible sequence in L.

Step 3. If there is a feasible sequence in L, use algorithm load-balancing to select the helped agent .Otherwise, select the agent according to the order of the sequence SI, ..., sk sorted in algorithm find feasible-sequence. That is, if will-help r(sl) is rejected, try r(s2), etc.

Let I(oi)=i be an index function for oi E L.

Algorithm Find-feasible-sequence.

Step 1. Sort L by keys ni and agent priority to an sequence SI, ..., sk. First sort by key ni with non-decreasing order. If ni = nj , compare the agent priorities of r(oi) and r(oj) to determine their order.

Step2. C=N- k
For 1<=i<=k do
J=I(si)
C=C+1
else
mark si+], ..., sk as the unsatisfied sequence,
the result is no feasible sequence,
exit.

step 3. The sequence SI,. . . ,sk is a feasible sequence.

Algorithm Load-balancing.

Step 1. Initialize the current number of agent, Ci , for all oi E L to 1.

Step2.for l<=i<=N do
i f r i E F then
for 1<= j < k do
l=I(sj);
if Cl< nl then
if i is equal to my id. then r1 is the
selected agent,
exit;
else cl=c1+1.

The agents in F also need some modification for their behaviour when they stay in (I-)WAITING state. If there is a feasible sequence, they can stay (I-)WAITING state and wait for help. While there is no feasible sequence, the agents in G must exit (I-)WAITlNG state to help each other in order to avoid a deadlock.

Step 1. Use algorithm find-feasible-sequence to check if there is a feasible sequence in L.

Step 2. If there is no feasible sequence in L, and the object found by itself is in the marked unsatisfied objects si+], ..., Sk, select the helped agent according to the order of the sequence si+l, ..., Sk. That is, if will-help r(si+l) is rejected, try next one in the sequence, etc. The sorted sequence is kept by all agents and is referred when they can help the others till all objects in L are moved.

5. Conclusion

This paper focuses on the problem of cooperation and deadlock-handling for a multi agent system. The agent architecture, cooperation strategies and deadlock-handling schemes are proposed for the task.

Increasing the number of agents will decrease the waiting time of agents and speedup the execution time. Three deadlock-handling schemes were proposed and analyzed to guarantee a deadlock-free cooperation. Preliminary results with two of the schemes were presented and more experiments are under way.

6. References

[I] R. C. Arkin, "Motor Schema-Based Mobile Robot Navigation",
International Journal of Robotics Research, Vol. 8,

[2] R. C. Arkin, "Cooperation without Communication: Multiagent
Schema Based Robot Navigation", J. of Robotic Systems,
Vol. 9(3), April 1992, pp. 351-364.

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

[4] 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.

[5] R. A. Brooks, P. Maes, :M. Mataric and G. More, " Lunar Base Construction Robots", IEEE International Workshop
on Intelligent Robots and Systems (IROS PO), pp. 389-392,
Tsuchiura, Japan, 1990.]

[6] R. A. Brooks, "A Robust Layered Control System For A Mobile Robot", IEEE J. of Robotics and Automation, Vol. RA-2, No. 1, March 1986, pp. 14-23.
[7] T. Fukuda, S. Nakagavva, Y. Kawauchi, and M. Buss, "Structure Decision for Self Organizing Robots Based on Cell Structure - CEBOT", Proc. of IEEE International Conference
on Robotics and Automation, Scottsdale Arizona, pp.
[8] S. Hackwood and S. Beni, "Self-organization of Sensors for
Swarm Intelligence", Proc. of 1992 IEEE International Conf. on Robotics and automation, Nice, pp. 819-829, 1992.
[9] F. C. Lin and J. Y.-j. Hsu, "A Decentralized Cooperation Protocol for Autonomous Robotic Agents", Proc. of The Second International Symposium on Autonomous Decentralized
Systems (ISADS 95), Phoenix, Arizona, April 1995.
[I0] M. Mataric, "Minimizing Complexity in Controlling a Mobile Robot Population", Proc. of I992 IEEE International Conf. on Robotics and Automation, Nice, pp. 830-835, 1992.
NO. 4, August 1989, pp. 9'2-112.