Proceedings of the 2003 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Las Vegas, Nevada · October 2003

Distributed Task Negotiation in Self-Reconfigurable Robots

Behnam Salemi, Peter Will, and Wei-Min Shen

USC Information Sciences Institute and Computer Science Department
Marina del Rey, USA, {salemi, will, shen}@isi.edu

Abstract

A self-reconfigurable robot can be viewed as a network of many autonomous modules. Driven by their local information, the modules can initiate tasks that may conflict with each other at the global level. How the modules negotiate and select a coherent task among many competing tasks is thus a critical problem for the control of self-reconfigurable robots. This paper presents a distributed algorithm called DISTINCT to solve this challenging problem and show that it can be successfully applied to the CONRO self-reconfigurable robots. A discussion how to apply DISTINCT to other types of distributed systems such as sensor network, swarm robots, or multi-agent systems is also given.

1 Introduction

A self-reconfigurable robot consists of many autonomous modules that can simultaneously initiate tasks on their own [1-3]. Driven by their local information, the modules may generate tasks that are competing even conflicting with one another. For example, in a snake configuration, the tail module may wish to move forward, while the head module may want to avoid an obstacle. How to select the correct task when there are many competing choices is then a critical problem for controlling the self-reconfigurable robots.

Distributed Task Negotiation is a process by which modules in a self-reconfigurable robot can negotiate and select a single coherent task among many different and even conflicting choices. This is a very challenging problem due to several reasons: the relationships among modules are not static but change with configurations, modules have no unique global identifiers or addresses, modules do not know the global configuration in advance, and can only communicate with immediate neighbors. Under these circumstances, the task negotiation problem demands a distributed solution. Modules must negotiate and select tasks through local communication, and they must synchronously terminate the negotiation process when every module knows locally that its current task has been accepted globally.

This paper presents the DISTINCT algorithm as a solution for the distributed task negotiation problem. The main idea is that all modules work together to build global spanning trees and each tree is associated with a task. Initially, all modules that have their own competing tasks start building their own trees, but as they exchange messages for tree building, most modules will give up their "root" status and participate in building trees for other tasks. In this process, modules report their status to their parent module in the tree that they participate, and the module that does not have parent but received reports from all its children is the root for the entire network of modules. When this happens, this root module can conclude that the negotiation process has succeeded and all modules in the tree have agreed on the same task. The correctness of this algorithm can be proved if the current robot configuration is acyclic (i.e., no loops in the current network of modules). To ensure the correctness for arbitrary configuration, additional knowledge (such as the network size, or module Ids) is needed so that the modules can detect the existence of loops in the network. The algorithm is efficient and its time complexity is in the low polynomials of the number of competing tasks.

The paper is organized as follows: Section 2 discusses the related work, Section 3 gives a formal definition of Distributed Task Negotiation; Section 4 presents the basic idea of creating and competing Task Spanning Trees; Section 5 describes the DISTINCT algorithm; Section 6 describes the experimental results in applying DISTINCT to the CONRO self-reconfigurable robots and simulated networks of modules; and finally Section 7 concludes the paper with future research directions.

2 Related Work

The distributed task negotiation problem occurs in many types of distributed systems including, for example, sensor networks [4], swarm robots [5], or multi-agent systems. In distributed multi-robot systems, previous approaches such as [6] often assume a designated central agent to dictate a task for all the conflicting agents. Another field that faces the same problem is distributed computing and algorithm design [7]. For example, our approach to detecting the termination of negotiation is inspired by the algorithm for termination detection in distributed computing systems [8].

This work is different from all existing approaches. Unlike centralized approaches, DISTINCT can scale well with configurations and is robust to individual module failures. Compared to most existing algorithms, our solution can deal with both task negotiation and termination detection among many modules.

0-7803-7860-1/03/$17.00 © 2003 IEEE

3 Distributed Task Negotiation

Although the distributed task negotiation problem is prominent in self-reconfigurable robots, it is also critical for many other reconfigurable systems such as sensor networks or multi-agent organizations. Thus, we define the problem in the context of a network of nodes that have communication links. For a self-reconfigurable robot, nodes are modules and links are physical connections between modules. For a multi-agent system, nodes are agents and links are communication channels between agents. The difference is that nodes in this paper do not have unique global identifiers or addresses, and they can only communicate with their immediate neighbors through existing links. The links are half duplex, which means that two nodes connected by a link can transmit messages in both directions but not at the same time. We suppose that all nodes in the network can autonomously initiate tasks and many tasks can compete simultaneously in the network.

Formally, a distributed task negotiation problem consists of a tuple (P, L, T, S), where P is a list of nodes, p_i, such that $i \in \{1,..., N\}$; L is a list of communication links, l_{jk}, such that $j,k \in \{1,..., N\}$; T is a list of tasks, t_m, such that $1 \leq m \leq N$, and S is a set of task selection functions, S_i: $(T') \rightarrow t_i$, such that $i \in \{1,...,N\}$ and $T' \subset T$. Each node has a task selection function that can select a single task from a set of given tasks. A distributed task negotiation problem is solved when all nodes have selected the same task from T, called t^*, and have been notified that the negotiation process is terminated. Note that the index numbers assigned to P are only used for defining the problem and not used in the negotiation process. In addition, the size of the network is unknown to the individual nodes.

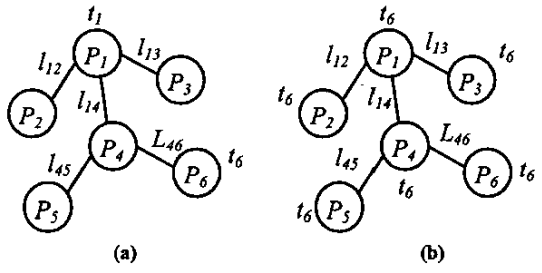

Figure 1: An example of a distributed task negotiation problem. **a)** Initially p_1 and p_6 initiated two tasks (t_1, t_6). **b)** A solution, when all agents have selected $t^* = t_6$.

To illustrate the above definition, consider the example in Figure 1(a), where $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, $L = \{l_{12}, l_{14}, l_{13}, l_{45}, l_{46}\}$, $T = \{t_1, t_6\}$, and S is a selection function that prefers tasks with greater indexes and shared by all nodes. Initially, node p_1 and p_6 have initiated two tasks, t_1 and t_6, respectively, and the rest of the nodes are waiting to receive tasks. Figure 1(b) depicts a solution for the given problem where all nodes agreed on task t_6.

4 Negotiation by Creating Spanning Trees

The most obvious solution for the problem is to assign priorities to the competing tasks and force nodes to select tasks that have higher priorities. However, since the importance of tasks cannot be determined statically, it is extremely hard to determine the correct priorities for an arbitrary set of competing tasks.

In our solution, nodes propagate their tasks to their neighbors and generate a Task Spanning Tree (TST) for each propagated task. As a result, when more than one task is initiated, a forest of partial TSTs is created. These partial TSTs negotiate with each other and gradually merge into one and only one TST. This final TST represents the task that has been selected by all nodes in the network. During the tree building process, all nodes report their status to their parent nodes. The negotiation process terminates when a node that has no parent has received reports from all of its children. This node is the root of the final TST, and it then notifies all nodes in the tree with an "end of task negotiation" message and all nodes will select the task associated with the final TST.

4.1 Distributed Task Selection

For nodes that have competing tasks to select a single task, the goal is to create a single TST. Each node must decide on two issues: 1) what task to select and propagate, and 2) how to be a part of a TST.

Initially, nodes that have competing tasks propagate their tasks by sending a *task message* (*TM*) to their neighbors and designating themselves as the root of a partial TST. Assuming that the recipient of a *TM* has no tasks for itself and receives only one *TM*, then it will adopt the received task and create a "child-of" relationship toward the sender of the *TM*. The recipient will in turn propagate the received task by sending a new *TM* to the rest of its neighbors.

To illustrate this idea, Figure 2 shows an example in which nodes P_1 and P_6 are the initiators of tasks t_1 and t_6 respectively and the rest of the nodes are non-initiator nodes. Node P_2 and P_3 are the recipients of $TM(t_1)$ sent by P_1, and therefore have selected task t_1. Similarly, P_4 and P_5 are the recipients of $TM(t_6)$ sent by P_6, and therefore have selected task t_6. In this situation, parallel arrows show the "child-of" relationships that the nodes have created.

Based on the above assumption, no message has been sent through the link l_{14}. As a result two TSTs have been formed; one rooted at P_1 and the other rooted at P_6. In each TST, all nodes have selected the same task.

At this point, if we relax the above assumption, two cases might occur: 1) either a root node receives a *TM*, or 2) a non-root node receives a *TM* from a node that is not its parent. An example of the first case happens in

Figure 2 when P_1, a root node, receives a *TM* from P_4. An example of the second case happens when P_4, a non-root node in the TST rooted at P_6, receives a *TM* from P_1, which belongs to another partial TST.

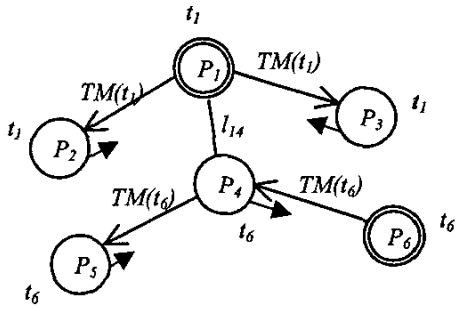

Figure 2: Task message propagation. Arrows on the links indicate messages in transit and arrows parallel to links indicate the "child-of" relationship. Double circles indicate the roots of partial TSTs.

In the first case, the recipient, which is a root node, drops being a root, adopts the received task, establishes a "child-of" relationship with the sender of the *TM* and propagates new *TM* to the rest of its neighbors, which are its children. In this situation, these nodes adopt the new received task and propagate it to the rest of their neighbors.

In the second case, the received *TM* is a conflicting message since it was received from a non-parent node. To resolve the conflict, the recipient node deletes all of its previous "child-of" relationships, makes a choice between its previous task and the received task (using its task selection function), propagates a *newRoot message* (*NRM*) containing the newly selected task to all of its neighbors, and then promotes itself as a new root for the selected task.

The role of *NRM* is to merge partial TSTs and create a new root for the resulting TST. Therefore, the recipient of a *NRM* adopts the received task, creates a new "child-of" relationship towards the sender of the *NRM*, becomes a non-root node (if previously a root), and propagates a new *NRM* containing the received task to the rest of its children.

Figure 3 shows the result of merging the two partial TSTs in Figure 2 for the situation, where P_4 has been the node that has received a conflicting *TM* from P_1. As a result, P_4 chooses a task between t_6 and t_1 (say t_6 is chosen), promotes itself to be the root of the new TST, and propagates $NRM(t_6)$ to P_1, P_5 and P_6, which turns P_1 and P_6 into non-root nodes. Consequently, P_1 will adopt t_6 as its new task and propagate a new *TM* to P_2 and P_3 for the task switch.

As shown in Figure 3, the final result of the task negotiation process is a single TST with a specified root node and a selected task. However, at this point the nodes do not know that the task negotiation process has been terminated. Unless a mechanism for detecting the termination of negation is in place, the nodes would wait indefinitely.

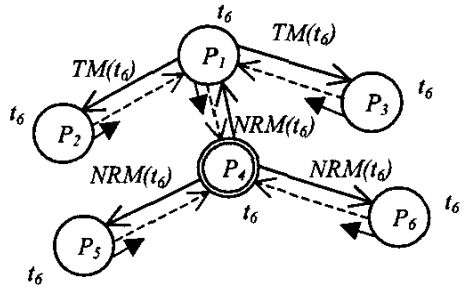

Figure 3: Merging partial TSTs from Figure 2. P_4 is the new root of the merged TST. The dashed arrows indicate the ack messages.

4.2 Distributed Termination Detection

In order to detect the termination of the task negotiation process, we use an approach similar to the "termination detection algorithm for diffusing computation" by Dijkstra and Scholten [8]. For each received *TM* and *NRM*, each node must reply with an acknowledge message (*AM*) after it receives acknowledges from all its children. For a leaf node, this means that it will acknowledge immediately for every received message. For a non-leaf node, it will send an acknowledge message to its parent after it receives *AM* from all of its children. If a non-leaf node receives all *AM* from all its children and it has no parent, then this node is the root for the final TST and it can conclude that the task negotiation process has succeeded.

In Figure 3, dashed arrows indicate the *AM* messages. The root node, P_4, expects to receive *AM*s from each of the P_1, P_5, and P_6 nodes. Since P_5 and P_6 do not have any child nodes, they send their *AM* as soon as they receive $NRM(t_6)$ messages from P_4. However, P_1 sends its *AM* to P_4 only after it receives *AM*s from P_2 and P_3. When P_4 receives all of its expected *AM*s, it detects the termination of the negotiation process and propagates a *taskSelected* message to all of its children. This message will be propagated to all the nodes in the tree and the task negotiation process is successfully determined.

5 The DISTINCT Algorithm

The distributed task negotiation process described above has been implemented as an algorithm called DISTINCT. Given a distributed task negotiation problem, this algorithm ensures that all nodes will select the same task coherently; regardless of the number of competing tasks initiated in the network.

Figure 4 illustrates the procedures of the DISTINCT Algorithm. Four types of messages are used. First, a

```
when initiated (task (t)) do
   SelectedTask = t;
   ParentLink = null;                    (a)
   ChildLinks = Links
   for each L ∈ ChildLinks do
    L.ackProcessed = false;
   end do;
   for each L ∈ ChildLinks do
    send (L ,task (t))
   end do;
end do;
when received (task (t), link (j)) do
 for each L ∈ Links do
 L.ackProcessed = false;
 end do;
 if (SelectedTask = null or ParentLink = j)
   SelectedTask = t;
   ParentLink = j;
   ChildLinks = Links - j;
   if (ChildLinks is not empty)
      for each L ∈ ChildLinks do
        send (L ,task (t));
      end do;
   else send (ParentLink, ack (t)); end if;
 else SelectedTask =SelectionFunction (t, SelectedTask);
   ParentLink = null;
   ChildLinks = Links
   for each L ∈ ChildLinks do
    send (L ,newRoot (SelectedTask))
   end do; end if;
end do;
when received (newRoot (t), link (j)) do
 SelectedTask = t;
 ParentLink = j;
 for each L ∈ Links do
  L.ackProcessed = false;
 end do;
 ChildLinks = Links - j;
 if (ChildLinks is not empty)
  for each L ∈ ChildLinks do
   send (L ,newRoot (t)) end if; end do;
 else send (j, ack (t)); end if;
end do;
when received (ack (t), link (j)) do
 j. ackProcessed = true;
 acknowledgeComplete? = true;
 for each L ∈ ChildLinks do
  if (L. ackProcessed =false)
   acknowledgeComplete? = false;
   break; end if; end do;
 if (acknowledgeComplete? = true)
  if (ParentLink ≠ null)
   ParentLink. ackProcessed = true;
   send (ParentLink, ack (t));
 else send(Links, taskSelected (t)); end if; end if;
end do;
when received (taskSelected (t), link (j)) do
   for each L ∈ ChildLinks do
    send(Links, taskSelected (t)); end do;
   terminate;
end do;
```

Figure 4: The DISTINCT Algorithm

task message (*TM*) is used for propagating the initiated tasks. Second, a *newRoot message* (*NRM*) is propagated when a conflict is detected and partial TSTs are to be merged. Third, an *ack messages* (*AM*) is used for detecting the termination event. Finally, a *taskSelected message* is propagated from the root of the final TST to all nodes in the network.

Task initiator nodes begin by calling the **initiated** procedure then wait for incoming messages. The 'Links' variable is the list of the communication links of a node. In addition, the *ParentLink* and *ChildLinks* variables specify the parent-child relationships among nodes in a TST. In line (a) of the **initiated** procedure, a node designates itself as a root node by assigning a **null** value to its *ParentLink* variable. As a result, all of the communication links (Links) of the root nodes are marked as *ChildLinks*. The *ackProcessed* variable is used for keeping track of the received *ack messages* to detect the task negotiation termination event. The currently selected task is stored in the *SelectedTask* variable. The *acknowledgeComplete*? in the **ack** procedure is a local variable that checks if all the expected number of **ack** messages are received. When the value of this local variable is true for the root of a TST, it detects that a single TST has been formed and the task negotiation process is terminated. Consequently, it propagates a **taskSelected** message to all of its children. The recipients of these messages will call the **taskSelected** procedure and eventually all nodes in the network will select the same task and the negotiation process is successfully terminated.

5.1 Algorithm Correctness

We now show that the DISTINCT algorithm will reach a stable state when all nodes have selected the same task. Assume there are N nodes in the network. As a result of communication of the initiated tasks, and just before any conflict is detected, the network is partitioned into a set of non-overlapping sub-trees, which are the partial TSTs. Nodes in the same partial TST have selected the same task.

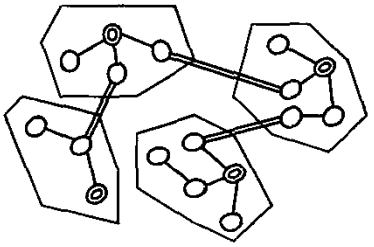

Figure 5: A network of nodes partitioned by partial TSTs. Polygons represent "super" nodes. The double lines are the conflicting links.

Based on the property that any two nodes in a tree are connected by a unique path, we may conclude that there is at most one connecting link between any two partial TSTs. Otherwise there will be more than one path from a node in one partial TST to a node in the other partial TST, which will contradict the above-mentioned property. Consequently, if each partial TST is

considered to be a single "super" node, the resulting network is also a tree; see Figure 5. The connecting links of these nodes are called *conflicting links* since the messages that they transfer cause conflicts in the recipient nodes.

Based on the above description, and by considering that this algorithm merges partial TSTs that have conflicting links between them, the DISTINCT algorithm will eventually produce one single TST. Furthermore, since the selected task for all merged TSTs is the same, only one task will be selected. In addition, due to the facts that there are only N-1 links in a tree with N nodes, and that merging will monotonically reduce the number of nodes, the number of *conflicting links* will monotonically reduce to zero. This means a single TST can be created after at most N times merging.

The correctness of the termination criterion can be seen as follows. Any leaf node of the TST is enabled to generate an *ack message* as soon as it receives *task* or *new root messages*. This in turn enables their parent nodes, and allows their parent nodes to generate *ack messages*. As a result, the root of the TST will receive all of its expected *ack messages* and the termination of the task negotiation process will be detected. It is important to notice that as long as nodes are receiving conflicting messages, which represent the existence of conflicting links and therefore multiple roots, nodes will not send *ack messages*. Therefore, the roots of partial TSTs will not terminate themselves prematurely when there are still multiple TSTs in the network.

In this algorithm we have used half duplex communication links between nodes. This is required to avoid cases where two neighboring nodes communicate simultaneously, which in some situations would result in both nodes designating themselves as roots, producing deadlocks or other unexpected results.

The complexity of DISTINCT can be estimated as follows. In the worst case, every initiated task may override all of the other nodes selected tasks, therefore the worst-case time complexity of the DISTINCT algorithm is $O(NT)$ where N is the number of nodes and T is the number of initiated tasks.

6 Experimental Results

We have applied the DISTINCT algorithm to the CONRO self-reconfigurable robot and performed an extensive set of experiments in a Java simulated self-reconfigurable system.

6.1 Task Negotiation in CONRO Robot

Metamorphic robots are modular robots consisting of a network of autonomous modules (nodes), which can autonomously attach and detach each other to form different configurations. CONRO is an example of such metamorphic robots [9]. In our previous work for the distributed control of locomotion and reconfiguration, we assumed that only one task was generated by one module in the robot at a time [3]. With the DISTINCT algorithm, we can now relax this assumption.

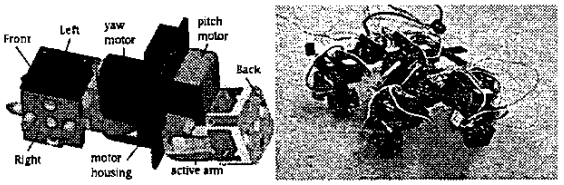

Figure 6: A CONRO module and a four-legged CONRO robot.

CONRO robots consist of a network of autonomous modules that can be modeled as a network of nodes in which all the assumptions described in the introduction section hold. Using the DISTINCT algorithm, a CONRO robot can select a single task among multiple initiated tasks. For example, Figure 7 shows the schematic view of a four-legged CONRO robot and its equivalent node network. Two modules of the CONRO robot have initiated *forward walk* and *Obstacle Avoidance* tasks. The network for this robot is the same as the network in the examples in Figure 2-3. Therefore as we saw earlier, the robot is capable of selecting a single task and detecting the task negotiation termination event. In this experiment, *Obstacle Avoidance* had a higher priority than the *forward walk* task.

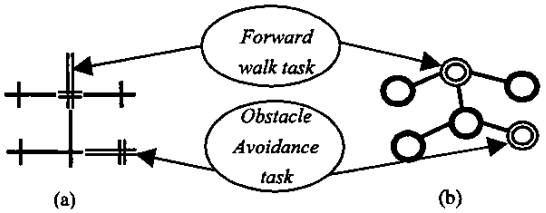

Figure 7: a) the schematic view of a four-legged CONRO robot. b) The node organization for the four-legged CONRO robot. The robot has initiated two tasks.

6.2 Performance Evaluation

We have also evaluated the performance of the DISTINCT algorithm in simulation with networks that have N = 10, 50, 200, and 1000 nodes. Each node has four connectors for connecting to other nodes. Configuration of the networks is randomly generated, and for each configuration we randomly selected a subset of nodes (with 1, N/2, N nodes in it) to initiate tasks. Each experiment is performed five times and averaged. Figure 8(a) shows the number of total messages sent by the nodes. Figure 8b shows the total number of cycles required for solving each distributed task negotiation problem on a logarithmic scale. Cycles are the number of times that a node executes a loop to check the received messages and send new messages.

Figure 8c and 8d show the average number of cycles per node and the average number of messages per node, respectively.

As we can see, when there is only one task initiator in the network, each node needs only two messages for each child and one message for its parent to build a tree that links all nodes. When half or all of the nodes are competing, the number of messages increases, because more modules must build and merge partial spanning trees and switch their tasks. In all cases, the experiments show that the DISTINCT algorithm ensures that all nodes select one and only one task in a distributed manner and the cost is of the low polynomial order with respect to the number of competing tasks.

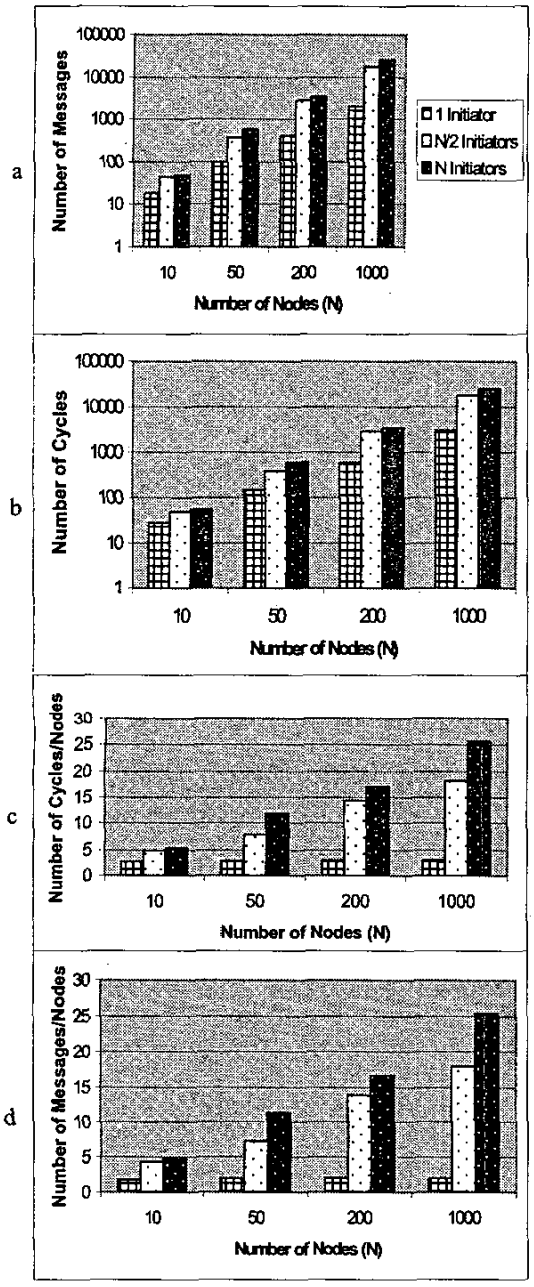

Figure 8: a) the total number of messages; b) the total number of cycles; c) the number of cycles per node, and d) the number of messages per node.

7 Conclusion

This paper presented a distributed algorithm called DISTINCT as a solution for distributed task negotiation in a network of autonomous and self-reconfigurable nodes. Such a network can be interpreted as a self-reconfigurable robot, a sensor network, or a multi-agent organization. The algorithm allows a large number of distributed nodes to agree and select a task from many competing choices and terminate the negotiation synchronously. The algorithm is proved correct in acyclic graphs and its time complexity is of the low polynomial order respect to the number of competing tasks. The future direction of this work is to handle networks that have loops. We believe using some additional knowledge such as the size of the network, nodes can detect the loops and achieve the same results shown by the DISTINCT algorithm.

8 References

1. Yim, M., Y. Zhang, D. Duff, *Modular Robots*, in *IEEE Spectrum*. 2002.
2. Rus, D., Z. Butler, K. Kotay, M. Vona, *Self-Reconfiguring Robots*, in *ACM Communication*. 2002.
3. Shen, W.-M., B. Salemi, P. Will, *Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots*. IEEE Transactions on Robotics and Automation, 2002. **18**(5).
4. Estrin, D., R. Govindan, J. S. Heidemann, S. Kumar, *Next Century Challenges: Scalable Coordination in Sensor Networks*, in *Mobile Computing and Networking*. 1999. p. 263-270.
5. Bonabeau, E., M. Dorigo, G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. 1999: Oxford University Press.
6. Mataric, M., *Integration of Representation Into Goal-Driven Behavior-Based Robots*. IEEE Transactions on Robotics and Automation, 1992. **8**(3): p. 304-312.
7. Lynch, N., *Distributed and Parallel Algorithms*. 1990: MIT Press.
8. Dijkstra, E.W., C.S. Scholten, *Termination Detection for Diffusing Computations*. Information Processing Letters, 1980. **11**.
9. Castano, A., W.-M. Shen, P. Will, *CONRO: Towards Miniature Self-Sufficient Metamorphic Robots*. Autonomous Robots, 2000. **8**: p. 309-324.