Distributed Task Negotiation in Modular 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

1 Introduction

Modular robots are a class of robotic systems that consist of many autonomous modules. Each module includes a set of sensors, actuators, and computational resources. Examples of modular robots are (self-) reconfigurable robots [1] [2] [3], and etc. Driven by the local information received from their sensors, multiple modules may simultaneously initiate 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 and also detecting the termination of the selection process are then critical problems for controlling the modular robots.

Distributed Task Negotiation is a process by which modules in a modular robot can negotiate to 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 may not be static but change with configurations; the number of modules in the robot is not known; modules have no unique global identifiers or addresses; modules do not know the global configuration, and can only communicate with their immediate neighbors.

This paper presents a distributed algorithm called DISTINCT as a solution for the distributed task negotiation problem. The main idea is that all modules cooperate to build global spanning trees. Each tree is associated with a task. Modules that have initiated tasks start with building and being the root of their own spanning trees, but as they exchange messages for tree building, most modules will give up being the "root" and participate in merging current trees to build larger trees for other tasks. In this process, modules report their status to their parent module of the tree in which they participate. Eventually, only one tree will remain in the entire network of modules and the module that does not have parent and receives reports from all its children is the root.

When this root module receives all of the expected messages, it 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 robot configuration is acyclic (i.e., no loops in the network of modules). To ensure the correctness for arbitrary configuration, additional knowledge (such as 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 of the low polynomial order respect to 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 [4] 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, Self-reconfigurable robots [5], sensor networks [6], swarm robots [7], or multi-agent systems [8]. In distributed multi-robot systems, previous approaches, such as [9] and [10], often assume a designated central agent to dictate a task for all the conflicting agents. Other approaches, such as [11], prevent this problem to occur by allowing only one agent to be the only task initiator.

This work is different from all existing approaches. Unlike centralized approaches, DISTINCT algorithm scales well with configurations and is robust to individual module failures. By letting all modules to be the task initiators, DISTINCT allows cooperative distributed problem solving [12] and can deal with both task negotiation and termination detection.

3 Distributed Task Negotiation

We define the problem of distributed task negotiation in the context of a network of *nodes* (agents) that have communication *links* (channels). For a modular robot, nodes are modules and links are physical connections between modules. Nodes do not have unique global identifiers or addresses, and they can only communicate with their immediate neighbors through existing links (Message propagation time is ignored). 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. 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 $i \in \{1,\dots, N\}$; *L* is a list of communication links, l_{jk}, such that $j,k \in \{1,\dots, N\}$; *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 $i \in \{1,\ldots,N\}$ and $T' \subset T$. This means that the task selection function need not be global and each node can have its own task selection function to 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. For example in Figure 1(a), $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, $L = \{l_{12}, l_{14}, l_{13}, l_{45}, l_{46}\}$, $T = \{t_1, t_6\}$, and S is a selection function that prefers tasks with larger indexes and shared by all nodes. Initially, node p_1 and p_6 have initiated two tasks, t_1 and t_6, respectively. Figure 1(b) depicts a solution for the given problem where all nodes agreed on task t_6. Note that the nodes' indexes are not used in the negotiation process.

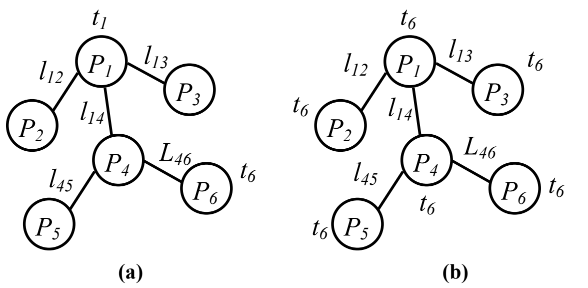

Figure 1: An example of a distributed task negotiation problem. **a**) Initially p_1 and p_6 initiated two tasks (t_1, t_6). **b**) A solution, when all agents have selected $t^* = t_6$.

4 Negotiation by Creating Spanning Trees

Assigning priorities to the competing tasks and forcing the nodes to select tasks that have higher priorities [13] will not solve this problem since the number of nodes and initiated tasks in the network are unknown.

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 gradually merge into one and only one TST, which represents the only selected task in the network.

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 other nodes in the tree with an "end of 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 and receives only one *TM*, then it will adopt the received task and create a "child-of" relationship toward the sender of the *TM* and propagate the received task by sending new *TM*s to the rest of its neighbors. Figure 2 shows an example in which nodes P_1 and P_6 have initiated tasks t_1 and t_6 respectively. Node P_2 and P_3 are the recipients of $TM(t_1)$, and therefore have selected task t_1. Similarly, P_4 and P_5 are the recipients of $TM(t_6)$, and therefore have selected task t_6. Parallel arrows show the "child-of" relationships.

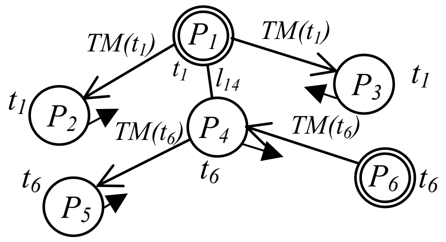

Figure 2: Task message propagation. Arrows on the links indicate messages in transit and arrows parallel to links indicate the "child-of" relationship. Double circles indicate the roots of partial TSTs.

Based on the above assumption, no message has been sent through the link l_{14}. As a result two TSTs have been formed; rooted at P_1 and 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* from another node, or **2**) a non-root node receives a *TM* from a node that is not its parent.

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*s to the rest of its neighbors. These nodes adopt the received task and propagate it to the rest of their neighbors.

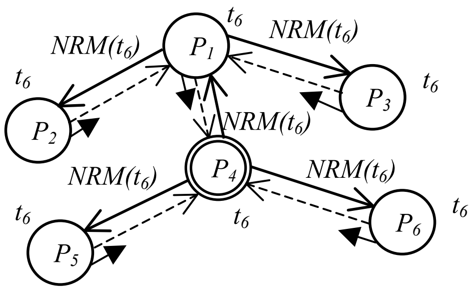

Figure 3: Merging partial TSTs from Figure 2. P_4 is the new root of the merged TST and dashed arrows indicate the AckMs.

Figure 3 shows an example of the second case when P_4, receives a *TM* from P_1. In this 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 "child-of" relationships, makes a choice between its previous task and the received task (using its task selection function), propagates a N*ewRoot 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. 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, and propagates a new *NRM* containing the received task to the rest of its children. The final result of the task selection process is a single TST with a specified root node and a selected task.

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 [14]. For each received *TM* and *NRM*, each node must reply with an Acknowledge Message (*AckM),* after receiving acknowledges from all its children. For a leaf node, this means that it will acknowledge immediately for every received message. A non-leaf node will send an *AckM* to its parent after it receives *AckM* from all of its children. A node that has no parent is the root of the final TST and it can conclude that the task negotiation process has succeeded.

Dashed arrows in Figure 3 indicate the *AckM*s. P_4 is the root node and expects to receive *AckM*s from all of its children. P_5 and P_6 are leaf nodes and respond immediately. After receiving all of the expected *AckM*s, P_4 detects the termination of the negotiation process and propagates a *Task Selected Message* (*TSM*), to all of its children. This message will be propagated to all the nodes in the tree and the task negotiation process successfully terminates.

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 *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 *Acknowledge Message* (*AckM*) is used for detecting the termination event. Finally, a T*askSelected Message(TSM)* 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, the *ParentLink* and *ChildLinks* variables specify the parent-child relationships among nodes in a TST, and the currently selected task is stored in the *SelectedTask* variable. In line (a) of the **initiated** procedure, a node designates itself as a root node by assigning a **null** value to its *ParentLink* variable.

```
 when initiated (task (t)) do
 SelectedTask = t; ParentLink = null;    ChildLinks = Links    (a)
 for each  L ∈ ChildLinks do send (L ,task (t)); end do;
end do;

when received (task (t), link (j)) 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 dosend (L ,newRoot (SelectedTask))
  end do; end if; end do;

when received (newRoot (t), link (j)) do
 SelectedTask = t; ParentLink = j; ChildLinks = Links - j;
  if (ChildLinks is not empty)
    for each  L ∈ ChildLinks do send (L ,newRoot (t)); end do;
  else send (ParentLink, ack (SelectedTask)); end if;  end do;

when received (ack (t), link (j)) do
 for each  L ∈ ChildLinks do
  if (ack msg was not received from L) return; end if; end do;
  if (ParentLink ≠ null) 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 (L, taskSelected (t));
 end do; terminate; end do;
```

Figure 4:The DISTINCT Algorithm

5.1 Example

Figure 5 shows a detailed example of how task selection and termination detection processes are performed. In Figure 5a, the root nodes P_1, and P6, are the only two task initiators initiating t_1 and t_6, respectively. Figures 5b, 5c, and 5d show the *TM*s and *AckM*s that are communicated by the nodes. On receiving *TM*s, nodes adopt the received tasks and create "child-of" relationships with their parents. In Figure 5c, P_4 can send an *AckM* to its parent only after receiving *AckM*s from both of its children.

Figure 5e shows the conflict detected by P_4 as a result of receiving the *TM* shown in Figure 5d. Figures 5f, and 5g show the communicated *NRM*s and the corresponding *AckM*s. Figure 5h shows the last *AckM* that P_4 receives before is detects the termination of the process. Figures 5i and 5j show the communicated *TSM*s.

5.2 Algorithm Correctness and Optimality

We now show that the DISTINCT algorithm will reach a stable state when all nodes have selected the same task and it is an optimal solution.

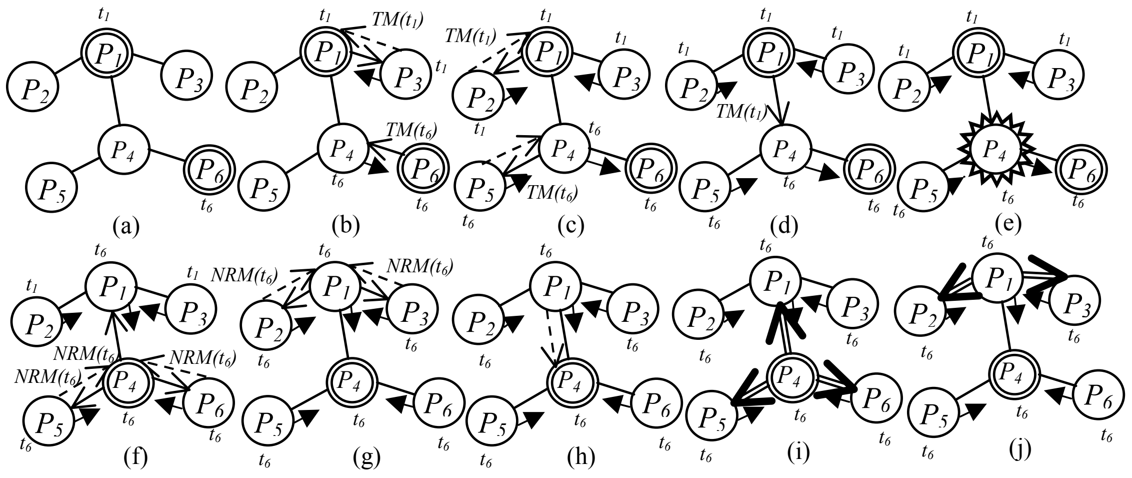
Figure 5: The steps of DISTINCT algorithm for task selection and termination detection.

After all initiated tasks are communicated, and just before any conflict is detected, the network is partitioned into a set of non-overlapping partial TSTs. Nodes in the same partial TST have selected the same task.

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 another partial TST, which contradicts 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 6. We call the connecting links of these nodes the *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.

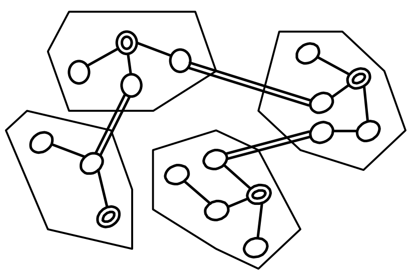
Figure 6: A network of nodes partitioned by partial TSTs. Polygons represent "super" nodes. The double lines are the conflicting links.

The correctness of the termination criterion can be seen as follows. Any leaf node of the TST is enabled to generate an *AckM* as soon as it receives *TM*s or *NRM*s. This in turn enables their parent nodes, and allows their parent nodes to generate *AckM*s. As a result, the root of the TST will receive all of its expected *AckM*s 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, conflict-detecting nodes will not send *AckM*s. Therefore, when there are still multiple TSTs in the network, the roots of partial TSTs will not terminate themselves prematurely.

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. Similarly, the total number of communicated messages is at most *NT*.

This shows that DISTINCT algorithm is an order of magnitude faster that centralized approach, which has the complexity of the order O(N^2). Furthermore, assuming that each node requires receiving at least one message to learn about a newly initiated task, we can conclude that in DISTINCT the number of communicated messages is optimal. That is because in the worse case, each task is communicated only once to each node.

6 Experimental Results

We have applied the DISTINCT algorithm to the CONRO self-reconfigurable robot and performed an extensive set of experiments in a large-scale 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 [4]. It consists of a network of autonomous modules (nodes) in which all the assumptions described in the introduction section hold. Figure 7a shows a CONRO module and Figure 7b shows a four-legged CONRO robot.

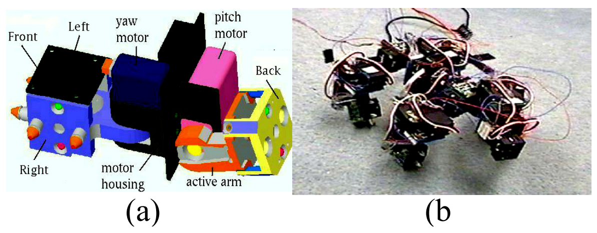

Figure 7: a) A CONRO module. b) A four-legged CONRO robot.

Each module has a set of sensors and actuators and decision-making is done locally. Therefore, it is possible that as a result of received information from the environment individual modules initiate multiple tasks. In our previous work for the distributed control of locomotion and self-reconfiguration of the CONRO robots, we assumed that only one of the modules in the robot could generate a single task at a time [5]. With the DISTINCT algorithm, we can now relax this assumption.

Using the DISTINCT algorithm, a CONRO robot can select a single task among multiple initiated tasks. For example, Figure 8 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 given earlier. Therefore, as we saw earlier, the robot is capable of selecting a single task and detecting the termination event. In this experiment, *Obstacle Avoidance* had a higher priority than the *forward walk* task.

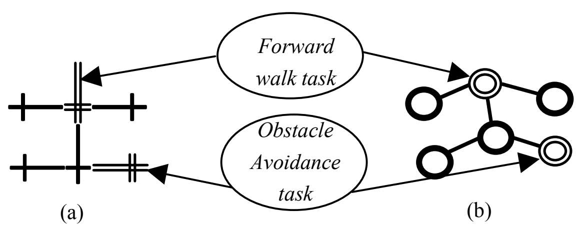

Figure 8: a) the schematic view of a four-legged CONRO robot. b) The node organization for the four-legged CONRO robot. The robot has initiated two tasks.

6.2 Performance Evaluation

We have also evaluated the performance of the DISTINCT algorithm in large-scale simulated self-reconfigurable 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.

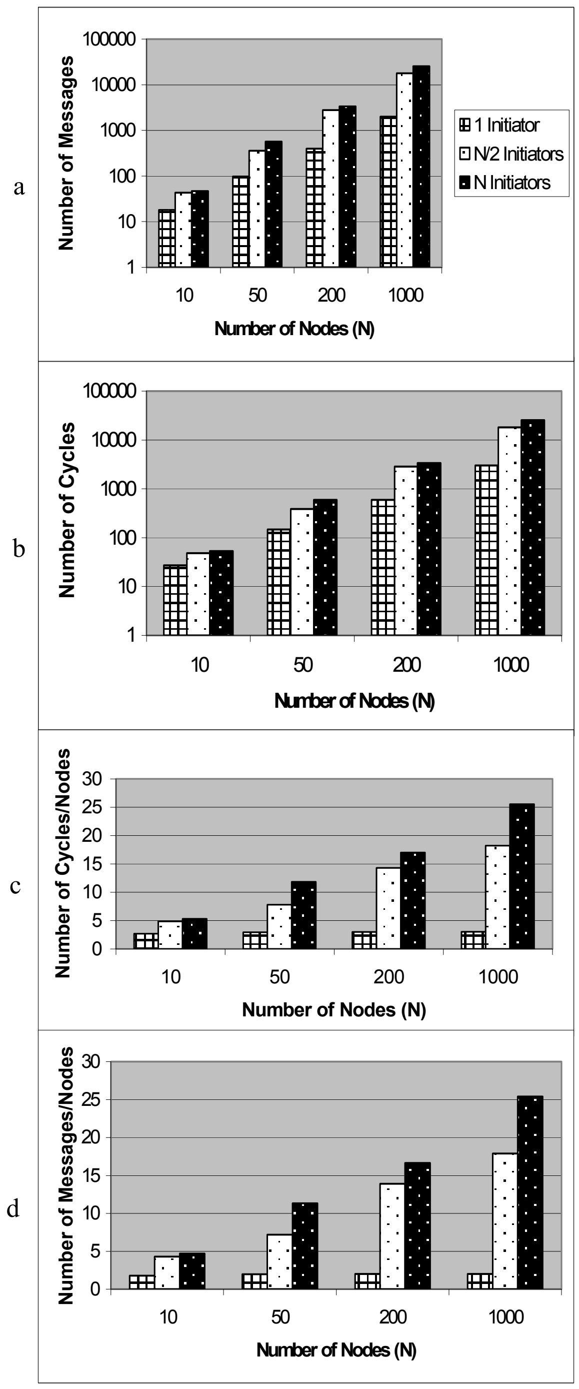

Figure 9: a) the total number of messages; b) the total number of cycles; c) the number of cycles per node, and d) the number of messages per node.

Each experiment is performed five times and averaged. 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.

Figure 9a shows the number of total messages sent by the 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.

Figure 9b 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. 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.

Figure 9c and 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.

Figure 9d 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 and the cost is of the low polynomial order with respect to the number of competing tasks.

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 detect the termination of the negotiation process. 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 adding IDs, 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.* IEEE Spectrum, 2002.
2. Rus, D., Z. Butler, K. Kotay, M. Vona,, *Self-Reconfiguring Robots.* ACM Communication, 2002.
3. Salemi, B., WM. Shen and P. Will. *Hormone Controlled Metamorphic Robots*. in *ICRA*. 2001.
4. Castano, A., W.-M. Shen, P. Will, *CONRO: Towards Miniature Self-Sufficient Metamorphic Robots.* Autonomous Robots, 2000: p. 309-324.
5. Shen, W.-M., B. Salemi, and P. Will., *Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots.* IEEE Transaction on Robotics and Automation, 2002. **18**(5): p. 700-712.
6. Estrin, D., R. Govindan, J. S. Heidemann, S. Kumar, *Next Century Challenges: Scalable Coordination in Sensor Networks.* Mobile Computing and Networking, 1999: p. 263-270.
7. Bonabeau, E., M. Dorigo, G. Theraulaz,, *Swarm Intelligence: From Natural to Artificial Systems*. 1999: Oxford University Press.
8. Tambe, M., *Towards Flexible Teamwork.* Journal of Artificial Intelligence Research, 1997. **7**: p. 83-124.
9. Mataric, M., *Integration of Representation Into Goal-Driven Behavior-Based Robots.* IEEE Transactions on Robotics and Automation, 1992: p. 304-312.
10. Yim, M., *Locomotion with a unit-modular reconfigurable robot (Ph.D. Thesis)*, in *Department of Mechanical Engineering*. 1994, Stanford University.
11. Stoy, K., Shen,WM., Will, P.,, *Using Role-Based Control to Produce Locomotion in Chain-Type Self-Reconfigurable Robots.* IEEE/ASME Transactions on Mechatronics, 2002. **7**(4): p. 410.
12. Durfee, E.H., Lesser, V.R., and Corkill, D.D, *Distributed Problem Solving.* Encyclopedia of Artificial Intelligence, Second Edition, 1991.
13. Lynch, N.A., *Distributed algorithms*. Data Management Systems, ed. J. Gray. Vol. 1. 2000: Morgan Kaufmann.
14. Djikstra, E.W., and Scholten, C.S. *Termination Detection for Diffusing Computations*. in *Information Processing Letters 11*. 1980.