Proceedings of the 2007 IEEE/RSJ International
Conference on Intelligent Robots and Systems
San Diego, CA, USA, Oct 29 - Nov 2, 2007

Optimal Distributed Planning of Multi-Robot Placement on a 3D Truss

Seung-kook Yun* and Daniela Rus†

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology, Cambridge, Massachusetts, USA
*yunsk@mit.edu,†rus@csail.mit.edu

Abstract— **This paper considers the problem of allocating tasks among robots that operate on a 3D truss. Each robot is commanded to navigate to a different location for work. When the information about the robots' initial and desired locations are centrally known, this problem reduced to a classical disjoint-path problem. In this paper we consider the distributed problem where each robot knows its own goals only and we wish to plan an optimal set of steps for each robot that minimizes energy while fulfilling the task requirements. The challenge is to cope with possible path collisions. We present and analyze a distributed algorithm. We describe a simulation of this algorithm and show data from a physical experiment.**

I. INTRODUCTION

In this paper we continue the study of truss climbing robots began in [1] and consider coordination problems when multiple robots are tasked to do work on the truss. Each robot is allocated a different location on the truss. We wish to develop a distributed algorithm that uses local information only (e.g. sensing and communicating locally) to plan paths for each robot from their initial locations to the target locations. We consider a set of identical robots that are capable to navigate a 3D truss-like structure such as the Shady3d robots in [1]. The key technical challenge is to plan an optimal set of collision-free paths that minimize the number of steps (and therefore energy consumption) for each robot. Collisions occur when one robot unit blocks the way, as these robots can only travel on free truss segments. Since the robots have information about their own mission only, it is very likely that the robots may encounter other robots along the way and need to look for alternative paths.

This problem arises within construction and inspection applications. Trusses are encountered as part of bridges, scaffoldings, space structures, and underwater platforms such as oil rigs. Tasks related to trusses are often dangerous and difficult for human workers, as the bars are narrow. Space construction and maintenance outside a spacecraft require dangerous extravehicular activity (EVA) missions by astronauts. We wish to create truss-climbing robots can do significant work to inspect, augment, or repair engineered truss structures. In the more distant future, these robots might become capable of climbing natural structures, such as trees, to assist with agricultural applications.

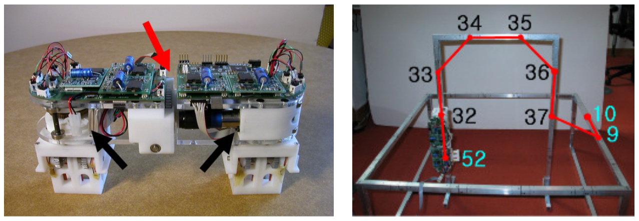
Coordinating a group of robots moving on a truss is easy when all the information about the environment, the robots, and their goals is available centrally. We can represent the truss as a graph, whose vertices are attachment places for the robots on the truss and edges connect adjacent links. Then the problem of moving k robots to their goal location along optimal collision-free paths reduces to a min-cost disjoint path problem with vertex capacities (since at one time each vertex can be occupied by only one robot.) This becomes an evacuation or assignment problem and has been studied extensively, for example optimal time and cost solutions are presented in [2]. The running time of one of the best algorithms is $O(k(k^2+n+m)log(n+k^2))$, where m is number of edges in a graph and n is number of vertices.

While simple, the centralized solution to this problem does not capture the reality of k robots moving autonomously and independently on a 3D truss to perform individual work at different locations. We wish to develop a distributed algorithm that relies on local information only, that can be realistically sensed and communicated by the robots. In this paper we describe a distributed planning algorithm for placing k identical robots on a 3D truss. We assume that the truss geometry is known to each robot and that the robot can detect and communicate with other robots located at neighboring nodes (e.g. one edge away). We analyze the running time of this algorithm and the competitive ratio. We show that our algorithm has quadratic competitive ratio and compare the result to a greedy algorithm whose competitive ratio is exponential. Finally, we present data from extensive simulations and from several physical experiments with Shady3D Robots.

The results in this paper draw from an extensive body of literature on truss-climbing robots [3]–[5], distributed motion planning [6], [7] and graph algorithms for matching [8] and flow [9].

II. EXPERIMENTAL ENVIRONMENT

In our lab we constructed a scaled-down version of a 3D truss environment and robots that can travel on it. Figure 1(b) shows the environment and Figure 1(a) shows the robots we developed to work within such an environment. Shady3D is a small robot with three degrees of freedom of length 250mm and width 80mm. It is composed of two rotating *grippers* linked by a two-part *arm*. Each gripper can grasp and release a truss bar by closing and opening its paddles. Each gripper is connected to the arm by a rotating joint (the gripper joint), which enables the gripper to align with a truss in various orientations. The two sides of the arm are connected by another rotating joint (the middle joint),



(a) (b)

Fig. 1. (a) Shady3D robot and its structure: 3-joints and 2 grippers (b) truss structure and an example of deployment of a single robot: numbers denote nodes on the trusses [10]

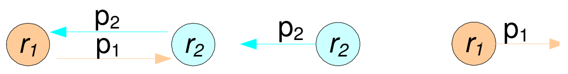
which creates a relative angle between the directions of the two grippers. The robot within a plane by using its ability to grasp and pivot. It makes transitions between planes by using the rotational degree of freedom on the bar.

We have built two fully working Shady3D robots and 5 Shady3d bodies that do not include any electronics, but can be used as obstacles during our experiments, to simulate the presence of up to 7 robots working together on the truss. The placement algorithms are implemented and tested using this environment.

III. Problem Formulation

Consider a truss with k robots on it. Each robot receives a set of goal locations. In this section we formulate the assumptions in the distributed placement problem we wish to solve. Local information only will be used to direct each robot to a goal location so that all targets are guaranteed to be occupied. We make the following assumption and notations:

- We are given a 3D truss with known geometry. The robots can grasp the truss at a discrete set of points. The truss is modeled as an undirected graph G, whose vertices are points where a robot can grasp (for example, the joint of the truss is not a vertex) and whose edges connect adjacent vertices. There is a positive cost on each edge.
- Each robot is modeled as one point that corresponds to an anchor gripper on the graph G.
- There are k identical robots on the truss.
- The robots can sense if an adjacent vertex is free or occupied by another robot.
- Each robot can communicate with the robots that occupy adjacent nodes.
- When two robots communicate, they can share all information (e.g. state, target location, etc.)
- The set of initial nodes in the graph is R; robot i is initially located at r_i. These locations are not known to the robots.
- The set of target nodes is T; $T = \{t_1, t_2, ..., t_k\}$
- The cost of a set of paths is the sum-total of the edge weights of the paths.
- The goal is for all target nodes to be occupied by the robots and for the overall path cost to be minimal.



(a) (b)

Fig. 2. Exchanging robots; r is a robot and p is a path (a) two crossing paths and (b) the system state after the exchange

IV. Distributed Algorithm by Locally Optimal Matching

In this section we describe a solution to the distributed placement of k robots on a truss using distributed locally optimal matching. The intuition behind this solution is that robots compute the location of the nearest target using as input the truss geometry and the list of targets. Then they start traveling toward their target in parallel. If the path of a robot is blocked by a different unit, or the robot finds that its target is already occupied, the robot is reconfigured by an operation of swapping state. This solution can be described as finding a matching between R and T.

Each robot runs a local algorithm for planning a path and moving along the path. The robot algorithm consists of several phases: initialization, path computation, path execution, and path reconfiguration in case of deadlock. Each robot's state includes the following data:

- *ID*: identification number
- *Status*: what it is doing now.
- *Settle Down*: true if it has settled down at the target
- *Pushing List*: a list of the robots pushing it now
- *Location*: currently occupying nodes
- *Initial and Target node*
- *matching list*: a list of the initial and target nodes a robot has learned by the collisions only with *settled-down* robots.
- *Path to the target*: a list of the nodes to the target

The following sections detail the phases of the algorithm.

A. Initialization

Using the initial state, the truss geometry, and the given set of targets, the robot computes the nearest target node.

B. Deployment

Algorithm 1 shows the *Distributed Deployment* procedure. This enables a robot to advance, to detect collisions, and to handle collisions. First, the algorithm checks if this robot has arrived at its desired target. If so, it sets the resource *Settle Down* as *true*, and stops. Otherwise, the robot checks for collisions by send a message If the next node is empty, it takes a step and updates the resources. In case of a collision the robot determines the collision type (one of five cases) and takes corresponding action as shown in SectionIV-C.

C. Handling Collisions

1) Crossing path: When the path of a robot and the blocking robot cross each other, the two robots exchange their destinations by exchanging all their state (see Figure 2.)

Algorithm 1 Distributed Deployment

```
1:  if has reached my target then
2:      Settle Down = true
3:  else
4:      Communicate with adjacent robots
5:      if Next node empty then
6:          Move to the next node
7:      else
8:          switch Cases of the Collision
9:              case The paths are crossing
10:                 Exchange the robots
11:             case The blocking robot has not settled down
12:                 Add Pushlist(my PushList + my ID)
13:                 if ID∈PushList & ID=min(PushList) then
14:                     Exchange the robots
15:                 end if
16:                 Wait until it moves while pushing it
17:             case settled down and the distinct target
18:                 Exchange the robots
19:             case settled down and non-anchor is occupying
20:                 request Step Aside
21:             case settled down and the same target
22:                 Merge matching information of the robots
23:                 Find a new locally optimal matching
24:         end switch
25:     end if
26: end if
```

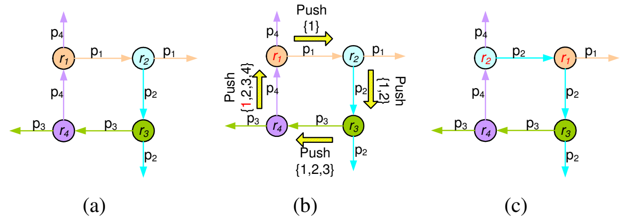


Fig. 3. Breaking a deadlock: (a) a deadlock (a cycle) (b) communication protocol for preventing the deadlock: *push* (c) the cycle is broken by exchanging identities.

2) Breaking a Deadlock: A deadlock is a status in which some robots can not move even though their paths do not cross as shown in Figure 3. There are four robots $\{r_1, r_2, r_3, r_4\}$. Their paths $\{p_1, p_2, p_3, p_4\}$, form a rectangular cycle (see Figure 3(a)). The *Pushing List* (which consists of robots waiting to advance) is used to eliminate deadlock. Each blocked robot sends its list to the blocking robots. The blocker merges the list onto its own. When a robot finds itself on its Pushing List (as shown in Figure 3(b)), it is in a cycle. In this case, the robot with lowest ID forces the blocking robot to execute *Exchanging the robots* until a robot in the list does not block it anymore. After this forced exchange, the cycle will be broken as shown in Figure 3(c). The robot with the lowest ID can proceed.

3) Stepping aside: Stepping Aside is necessary because Shady3D has two grippers and occupies two nodes. Consider

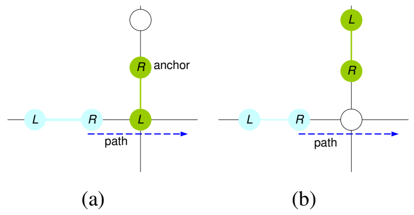


Fig. 4. Stepping Aside: (a) the blue robot requests stepping aside of the green robot (b) after the stepping aside of the green robot

the scenario in Figure 4(a). The green robot has settled down at the node occupied by the right gripper (the anchor), and the blue robot is trying to go through the node occupied by the non-anchor gripper of the green robot. Even though the paths do not cross, there is no cycle, and the two robots have the distinct targets, the blue robot is stuck. In this case the blocked robot requests the blocking robot to step aside so it can move as shown in Figure 4(b).

4) Finding a new target: The last selection criterion explains how to select a new target when two robots that block each other have the same target node. To solve this conflict, the blocked robot finds another target node, computes a new path to it, and follows the path as it has done. The new target is a node in the optimal set of the target nodes matched with the initial nodes of the blocked robot and the Matching lists of both robots.

Algorithm 2 describes the locally optimal matching. When a robot p has found that the other q occupied its target, they merge their Matching list $\{R_q, T_q\}$ and $\{R_p, T_p\}$. Using the merged Matching list and its starting node, it calculates the next target which is incident on the locally optimal matching M_{p+q}. This is computed efficiently using the *Hungarian algorithm* [11] which has $O(k^3)$ runtime and can be used with any size matrix.

Algorithm 2 Getting a Locally Optimal Matching

```
1: Merge the matching lists $\{R_q, T_q\}$ and $\{R_p, T_p\}$
2: Find the next target which belongs to $M_{p+q}$ by Hungarian algorithm
3: Change matching lists of both robots to the merged one
```

D. Termination

If at least one robot is moving, the total distance between the robots and the target nodes is strictly decreasing. If each robot has a distinct target node the perfect matching has been reached. Since there is no deadlock in the system the actions of the robots will converge and terminate.

V. Analysis

In this section, we analyze the optimality and the computational runtime of the distributed matching algorithm for placing robots of a truss. We show that the distributed algorithm has $O(k^2)$ asymptotic competitive ratio to the global optimum and the total runtime of all robots is bounded by $O(k^5 + k^2(n+m)logn)$.

A. Online matching: previous work

Our analysis uses several results on minimal weight partial matching from [8]. The minimal weight partial matching M_i which is the set of edges that form the minimal weight partial perfect matching between the subset $\{r_1, r_2, ..., r_i\}$ and subset of T with a minimal number of edges in $M_i - M_{i-1}$. Let T_i be the subset of T consisting of vertices of T which are incident on M.

Lemma 1: The cost of the M_is form a monotonically non-decreasing sequence.

Lemma 2: For each i, the set difference $T_i - T_{i-1}$ contains exactly one vertex.

Lemma 3: For a union set composed of vertices of M_{i-1} and $\{r_i, t_i\}$ where r_i is chosen to be a incident vertex on M_i, the cost from r_i to t_i is bounded by the cost of $M_{i-1} + M_i$

Proofs of Lemma 1- 3 are given in [8]. From now, We also use M as the cost of the set M.

In this section we extend the partial optimal matching results to our algorithm. Let $R_p \in R$ contain the robots that settled down on their target nodes according to robot p. Let M_{R_p} be the min-cost matching of the sub-group R_p. Note that M_{R_p} is sequentially constructed as a new robot collides with it. Therefore, Lemma 1- 3 hold for R_p and M_{R_p}.

B. Running Time

Suppose each robot has the cost matrix C where C_{ij} is the cost of the shortest path from r_i to t_j. We use the partial matrix of $C(R_{p+q}, T)$ corresponding to a set of $\{R_p \cup R_q \cup initialnode\}$ and entire T. After getting the new target, the matching lists are exchanged with the merged list, and the robot follows a new path to the target.

Lemma 4: The initial node and target node matched by a settled down robot stays fixed.

Proof: Suppose a robot starting at r_a has arrived at its target t_a and settled down. This robot will leave the target node, if and only if another robot exchanges the resources with it, by Algorithm 1. This can only happen for type 3 collisions. In this case, the exchange of states between the two robots causes the matched pair (r_a, t_a) to be maintained.

Lemma 5: A merged matching list from the lists of two robots has the exactly same number of initial nodes and target nodes.

Proof: Suppose that robots A and B have their matching lists $\{R_a, T_a\}$ and $\{R_b, T_b\}$ which have the same number of nodes in both R and T. If A has the same target as B, they collide. Suppose B is settled. The lemma is true if and only if:

$$n(R_a \cap R_b) = n(T_a \cap T_b), \tag{1}$$

where a funcion $n()$ notifies the number of a set. By Lemma 4, Equation 1 holds, because otherwise at least one of the initial nodes in $\{R_a \cap R_b\}$ must match more than one target nodes.

Lemmas4 and 5 imply that it is necessary to add only one target node to $\{T_a \cap T_b\}$ in order to match $\{R_a \cap R_b, r_a\}$. Figure 5 shows how a robot behaves using the algorithm. In Figure 5(a), robot p with matching list $\{R_p, T_p\}$ collides with robot q with $\{R_q, T_q\}$ at the node t_q. Then p gets a new target and the merged list $\{R_p \cup R_q, T_p \cup T_q\}$. Then p follows a new path from t_q to t_{p+q}. Note that the path to t_q from somewhere among T_p belongs to the previous step. We have $O((p+q)^3)$ running time by Hungarian algorithm [11].

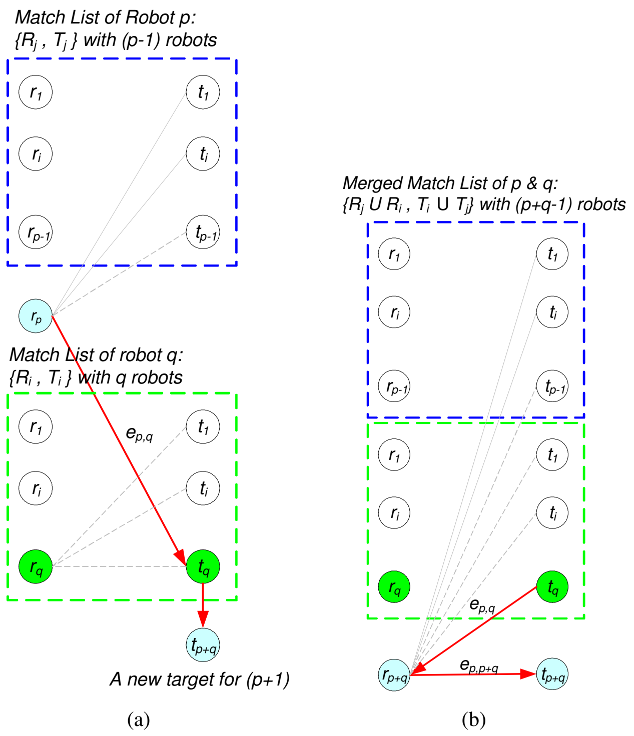


Fig. 5. The procedure of the deployment

C. Optimality by Competitive Ratio

In this section we investigate optimality by finding the competitive ratio, which is defined as the cost of the worst case to the globally optimal one.

Lemma 6: When robot p with matching list $\{R_p, T_p\}$ collides with robot q with matching list $\{R_q, T_q\}$ at node t_q(See Figure 5), the path to the new target t_{p+q} costs up to $4M_{p+q}$, where M_{p+q} is the optimal matching between the set $\{R_p \cup R_q, r_p\}$ and $\{T_p \cup T_q, t_{p+q}\}$.

Proof: The path is the red line in Figure 5(a), from t_q to t_{p+q}. By triangular inequality (Figure 5(b)), the cost is bounded by the edges, (t_q, r_p) and (r_p, t_{p+q}).

$$cost(t_q \to t_{p+q}) \leq e_{p,q} + e_{p,p+q} \tag{2}$$

The first edge is due to the previous collision and bounded by $M_{p-1} + M_p$, which is the locally optimal matching between $\{R_p, T_p\}$ and $\{\{R_p, r_p\}, \{T_p, t_q\}\}$ respectively, by Lemma 3. The second is also bounded by $M_{p-1+q} + M_{p+q}$, the locally optimal matching of $\{R_p \cup R_q, T_p \cup T_q\}$ and $\{\{R_p \cup R_q, r_p\}, \{T_p \cup T_q, t_{p+q}\}\}$. Therefore, Equation 2 becomes:

$$\begin{aligned} cost(t_q \to t_{p+q}) &\leq M_{p-1} + M_p + M_{p-1+q} + Mp + q \\ &\leq 4M_{p+q} \end{aligned} \tag{3}$$

because the cost of the local optimum never decreases by Lemma 1.

Lemma 7: The total cost of the path for the i^{th} robot to the settled down position is bounded by $(4i-3)M_{R_i}$.

Proof: We use induction. For convenience, assume that the robots deploy sequentially, one by one after the previous one's settlement. When the first robot settles down, the lemma holds. Suppose the bound is true for $(i-1)$, and then the cost P_{i-1} is:

$$P_{i-1} \leq (4(i-1)-3)M_{R_{i-1}} \tag{4}$$

By lemma 6, P_i is bounded by following:

$$\begin{aligned} P_i &\leq P_{i-1} + 4M_{R_i} \\ &\leq (4(i-1)-3)M_{R_{i-1}} + 4M_{R_i} \\ &\leq (4i-3)M_{R_i} \end{aligned} \tag{5}$$

Lemma 8: The distributed deployment algorithm has $O(k^2)$ asymptotical competitive ratio.

Proof: The total cost of deployment is the sum of all k robots' cost, and it can be written as follows:

$$\begin{aligned} \sum_{i=1}^{k}(4i-3)M_{R_i} &\leq \sum_{i=1}^{k}(4i-3)M_R \\ &= (2k^2-k)M_R \end{aligned} \tag{6}$$

where M_R is the global optimum.

Lemma 9: The total running time of the distributed placement algorithm is $O(k^5 + k^2(n+m)logn)$.

Proof: Whenever a robot is getting a new target by the proposed algorithm, $O(k^3)$ runtime is required. Therefore, total runtime for Hungarian algorithm is $O(k^4)$. $O(k(n+m)logn)$ is to calculate the cost matrix. Total runtime of all robots is obtained to multiply k to the runtime of each robot $O(k^4 + k(n+m)logn)$.

Thus, our proposed distributed matching algorithm has $(2k^2-k)$competitive ratio, and $O(k^2/logk)$ times the running time of the central controller.

The quadratic competitive ratio is due to the fact that the robots do not share their matching lists. If a robot can share the information with the collided robots by any method, we achieve $O(k)$ competitive ratio, which is the competitive ratio of the online matching algorithm.

D. Comparison to The Greedy algorithm

In this section we show that the performance of a distributed greedy algorithm has an exponential competitive ratio and thus much worse than our algorithm.

Consider an intuitive greedy algorithm where each robot successively finds the nearest target node. Consider Figure 6 where $(k-1)$ robots have occupied $(k-1)$ target nodes and one robot must find its target by the greedy algorithm. If the left edge of the robot has $(1+\varepsilon)$ cost, where ε is a small positive number, the robot will continuously go to the right node and finally return to t_k after visiting all the right nodes. The cost is (2^k-1), while the optimal cost is only $(1+\varepsilon)$. Therefore, in this case, the competitive ratio is (2^k-1) when ε is small.

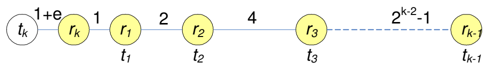


Fig. 6. A bad situation for the greedy algorithm.

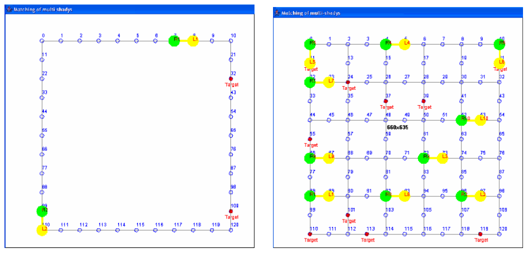


Fig. 7. (a) Sparse graph with two robots (b) Dense graph with ten robots

VI. Implementation

We implemented the distributed deployment algorithm in simulation and on the physical platforms described in Section II.

A. Simulation

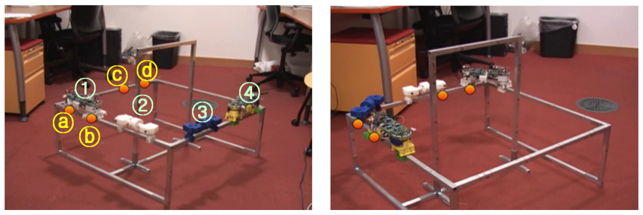
We have implemented the distributed placement algorithm in Java. We simulated each robot as an independent process (thread) to ensure parallelism. The target nodes and the initial placement of the left and right grippers of each robot were randomly selected. The left gripper was initially used as the robot's anchor.

We have tested two kinds of geometries and generated sparse and dense graphs, as shown in Figure 7. For each graph, 2∼10 robots are simulated 100 times, respectively. The parameter of the fixed communication time was set 0.1 second. The number of the communications and the faster total execution time are inversely proportional to this time. The total time is defined as the duration from the start of the simulation till the termination of each robot process.

The statistical results collected from these simulations are shown in Table I. For both graphs, the average ratio of the cost by the distributed algorithm to the cost of the centralized globally optimum algorithm are very slowly proportional to

TABLE I
RESULT OF THE SIMULATIONS

# of robots	Avg. Ratio Real/Opt		Avg. Comm. per robot		Worst Cast Real/Opt	
	Sparse	Dense	Sparse	Dense	Sparse	Dense
2	1.1	1.1	0.7	0.8	1.8	1.7
3	1.2	1.3	1.3	1.8	2.1	2.8
4	1.3	1.4	1.5	2.7	2.4	2.6
5	1.5	1.6	3.0	3.9	2.5	3.3
6	1.6	1.8	4.4	5.1	2.6	3.4
7	1.8	1.7	4.2	3.0	2.9	3.2
8	1.9	1.8	4.9	3.9	3.5	3.1
9	1.9	1.9	5.8	3.9	4.4	4.1
10	2.0	2.1	6.2	5.0	3.3	3.4



(a) (b)

Fig. 8. reconfiguration of four robots

TABLE II
RESULT OF THE EXPERIMENTS

Exp #	Optimal Cost	Traveled Cost	Exchange Count	Avg. Comm.	Finding new Opts
1	15	15	9	22	3
2	14	25	4	5	5
3	19	20	3	3	3
4	4	4	0	0	0
5	17	17	2	13	3
6	28	28	14	14	3

the number of the robots, whereas the analytical worst bound increases in a quadratic fashion. Even with 10 robots, the ratio is only around two. It appears that the graph type does not affect it. The average number of communication per robot is larger in the sparse graph than in the dense graph. This makes sense because a robot tends to have more chances to collide on a sparse graph.

B. *Physical Experiment*

The simulation algorithm has been transfered to the hardware system in Section II. For these experiments we use two Shady3D robots, two inert Shady3D robot bodies, and a simple truss structure as shown in Figure 8. Shady3D communicates via Bluetooth receivers. Its range is adjusted to the one-edge distance. The inert robots are introduced to increase the number of robots and collisions in this setup (at the moment we have only two Shady3D robots.) They are manually controlled and simulated by the main computer as if they move and even communicate by themselves. We set 5 seconds as communication rate and 10 seconds as the moving time of the inert robots (the human operator moves them during this time).

We performed six experiments using different robot placements and target locations. Snapshots of the experiment are shown in Figure 8, where four robots are moving and orange circles represent the target nodes.

The performance summary for the six physical experiments is given in Table II. While the competitive ratio is almost the same as for the simulation case, the average communication is much higher. Note that the number of communication is highly dependent on the communication rate.

We have encountered errors in the experiments. The errors are caused by over current of the motors in case of misalignment of the robot gripper and the truss- while moving. We are working on improving the robustness of the hardware.

VII. CONCLUSION

This paper describes a distributed localized algorithm for placing multiple identical robots at desired locations on a truss in a path-optimal way. We developed this work in the context of a modular mobile manipulator Shady3D. The algorithm is based on successive locally optimal matching The robots discover the target nodes incrementally through collisions and self-organize as the solution. Our solution is feasible, and has a quadratic competitive ratio $O(k^2)$ which is much more efficient than the greedy solution which exhibits exponential competitive ratio. We have also analyzed that running time as $O(k^5+k^2(n+m)logn)$ as compared to the centralized offline algorithm with $O(k(k^2+n+m)log(n+k^2))$ runtime. In the simulations, we have found out the algorithm works very efficiently in a view of the cost and the communication. Finally, we applied it to the real system with Shady3Ds and additional inert robot modules.

We view this work as first steps of our long-term vision of assembling robots and structures using robot modules and components form the environment. In our current work we are investigating how to combine the capability of the robots to self-organize with a construction task.

VIII. ACKNOWLEDGEMENTS

We are grateful to prof. David Karger for his help and guidance with analysis of the algorithm, and Sooho Park for his comment and discussion about the matching. This work was supported by NSF and Samsung Scholarship to Mr.Yun. We are grateful for this support.

REFERENCES

[1] Y. Yoon and D. Rus, "Shady3d: A robot that climbs 3d trusses," in *Proceedings of the International Conference of Robotics and Automation*, 2007.

[2] H. W. Hamacher and S. A. Tjandra, "Mathematical modeling of evacuation problems: State of the art. pedestrian and evacuation dynamics," Institut Techno- und Wirtschaftsmathematik, Tech. Rep.

[3] H. Amano, K. Osuka, and T.-J. Tarn, "Development of vertically moving robot with gripping handrails for fire fighting," in *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Maui, Hawaii, USA, 2001, pp. 661–667.

[4] A. Greenfield, A. A. Rizzi, and H. Choset, "Dynamic ambiguities in frictional rigid-body systems with application to climbing via bracing," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, Apr. 2005, pp. 1959–1964.

[5] M. Vona, C. Detweiler, and D. Rus, "Shady: Towards robust truss climbing with mechanical compliances," in *Proceedings of the International Symposium on Experimental Robotics*, 2006.

[6] Z. Butler and D. Rus, "Distributed planning and control for modular robots with unit-compressible modules," *International Journal of Robotics Research*, vol. 22, pp. 699–715, 2003.

[7] W. Lee and A. Sanderson, "Dynamic analysis and distributed control of the tetrabot modular reconfigurable robot system," *Autonomous Robots*, vol. 10(1), pp. 67–82, 2001.

[8] B. Kalayanasundaram and K. Pruhs, "Online weighted matching," *Journal of Algorithms*, vol. 14(3), 1993.

[9] J. Edmonds and R. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of ACM*, vol. 19, pp. 248–264, 1972.

[10] Y. Yoon, "Modular robots for making and climbing 3-d trusses," Master's thesis, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, 2006.

[11] H. Kuhn, "The hungarian method for the assignment problem," *Naval Res. Logist. Quart.*, vol. 2, pp. 83–97, 1955.