Tunable Routing Solutions for Multi-Robot Navigation via the Assignment Problem: A 3D Representation of the Matching Graph

Lantao Liu and Dylan A. Shell

Abstract— **In scenarios in which new robots and tasks are added to a network of already deployed, interchangeable robots, a trade-off arises in minimizing the cost to execute the tasks and the level of disruption to the system. This paper considers a navigation-oriented variant of this problem and proposes a parametrizable method to adjust the optimization criterion: from minimizing global travel time (or energy, or distance), to minimizing interruption (*i.e.,* obtaining the fewest number of robot reassignments), and mixtures in-between. Paths are computed by a task-allocation formulation in which the destinations of newly deployed robots are added to an existing allocation. We adapt the graph matching variant of the Hungarian algorithm—originally designed to solve the optimal assignment problem in complete graphs—to construct routing paths by showing that there is an interpretation of the sparse Hungarian bipartite graph in three dimensions. When new agent-task pairs are inserted, the assignment is reallocated in an incremental fashion in linear time (assuming traversal choices are limited in number). The algorithm is studied systematically in simulation and also validated with physical robots.**

I. Introduction

Several multi-robot applications involve elements that can be formulated as the problem of moving a team of mobile robots from their current positions to a set target locations whilst minimizing some collective cost, *e.g.,* shape morphing [1], deployment [2], formation control [3], and task reallocation [4]. This paper formulates and proposes a solution to an incremental version of the problem: a set of robots at known locations are added to a network of already deployed (identical) robots and, additionally, a set of new desired locations is specified. The problem is to decide which nodes should be moved and to where.

One simple solution is to have each newly added robot move toward the new target nearest to it. If the time to reach all the targets is important, or if robots are limited in where they may travel, then it may be better to have a deployed robot move to the target and fill its vacancy with another robot. With interchangeable robots, a whole chain of robots may simultaneously move toward the target, each robot taking the place of the last. Potential optimization criteria include: the distance travelled (total cost), disruption caused by redeploying robots (number of adjustments), time taken (deadline for completion).

We describe how opposing criteria (the minimization of summed distances versus number of robots redeployed) can be balanced however desired by using a parametrizable approach. Our formulation considers a graph on which traversability costs and constraints are encoded (via weights or omitted edges). The method produces a *routing* in which a subset of the agents move along edges toward target vertices. Computing a solution for the addition of an agent-target pair costs linear time in (typical) problem instances where the graph has bounded degree. The method permits addition of multiple agents and targets simultaneously, because sequential treatment (*e.g.,* via repeatedly using single source shortest path algorithms) may fail to find the global optima.

We show that the path routing problem is modelled as a weighted matching problem on a two-layered, sparse graph which can be visualized in three-dimensions. Routings are produced by applying a variant of the classic Hungarian Algorithm. We present an extension of the original (complete bigraph) method to handle sparse bigraphs and include the conditions under which valid solutions are produced. Prior work has considered incremental assignment solutions (*cf.* [5], [6]) but our focus is on the particular instantiation arising from path routing and, thus, we address insertion of multiple robots and targets, sparsity, and disruption.

II. Related Work

A. Task reallocation, formation control and morphing

This paper connects research focused on enforcing some metric/shape constraints (*e.g.*, formation control [7], [8]) with techniques for reassignment of robots to tasks, *e.g.*, [4], [9]. Shifting smoothly from one shape to another —sometimes called morphing— can be performed using the method we describe, but is approached in quite distinct ways in the literature (*cf.*, [1] and [10]). A recent example of reallocation and formation work together is [11], which describes a polynomial time method to extract a set of robots from a larger group in order to perform some new task, but while also minimizing interference induced by the set.

B. Related approaches based on the Hungarian Algorithm

The Hungarian Algorithm is one of the most well-known assignment algorithms for solving $n \times n$ assignment problems, *i.e.*, those with n agents to be uniquely assigned to n tasks. It was proposed by H. W. Kuhn and refined by J. Munkres [12] to have time complexity $O(n^3)$. Recently, the bigraph variant of the algorithm has been extended in interesting ways. Toroslu *et al.* proposed the *incremental assignment algorithm* [5] that finds a new solution with cost $O(n^2)$ after a new pair of vertices are added to a weighted bigraph with known matching. Motivated by examples in the transportation domain, Mills-Tettey *et al.* [6] proposed the *Dynamic Hungarian Algorithm* to handle cost changes

Department of Computer Science and Engineering, Texas A&M University {lantao, dshell}@cse.tamu.edu A preliminary version of this work appeared as a poster in the Workshop on Network Science and Systems, ICRA 2010.

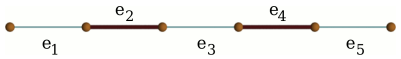


Fig. 1: An *augmenting path* is a sequence of alternating matched (bold) and unmatched edges, with the first edge and last edge unmatched. We say the edges are *augmented* when the matched and unmatched edges are flipped.

with time complexity of $O(kn^2)$, where k is the number of cost changes. Straightforward extensions to both algorithms allow deletions while preserving their complexities. Impelled by the particular bigraph arising from the routing problem, this paper examines the effect of sparsity. using the same incremental matching approach, *viz.* execution of stages of the Hungarian Algorithm, showing that a linear time solution is possible when the vertices have bounded degree.

Algorithm 1 The Hungarian Algorithm

Input:

An $n \times n$ assignment matrix represented as the complete weighted bigraph $G = (X, Y, E)$, where $|X| = |Y| = n$.

Output:

A perfect matching M.

1: Generate an initial labelling l and matching M in G_e.
2: If M perfect, terminate algorithm. Otherwise, randomly pick an exposed vertex $u \in X$. Set $S = \{u\}$, $T = \varnothing$.
3: If $N(S) = T$, update labels:
$\delta = min_{x \in S, y \in Y \setminus T}\{l(x) + l(y) - w(x, y)\}$

$$l'(v) = \begin{cases} l(v) - \delta & \text{if } v \in S, \\ l(v) + \delta & \text{if } v \in T, \\ l(v) & \text{otherwise.} \end{cases}$$

4: If $N(S) \neq T$, pick $y \in N(S) \setminus T$.
(a) If y exposed, then $u \to y$ is an augmenting path then augment matching M and go to step 2.
(b) If y matched, say to z, extend the tree: $S = S \bigcup \{z\}$, $T = T \bigcup \{y\}$, and go to step 3.

Notes:

- Equality graph $G_e = \{e(x, y) : l(x) + l(y) = w(x, y)\}$
- Neighbor of vertex $u \in X$: $N(u) = \{v : e(u, v) \in G_e\}$

III. Preliminaries

The graph matching formulation of the Hungarian Algorithm [12] appears in Algorithm 1. It treats the input $n \times n$ utility matrix as a complete bipartite graph (or bigraph) $G = (X, Y, E)$, in which X, Y respectively represents the set of agents and tasks, and E is the set of edges weighted by the utilities between agent-task pairs, $|E| \leq n^2$. The optimal assignment is sought as a maximally weighted perfect matching M where each agent in X is uniquely assigned to a task in Y. The algorithm augments the set of matched edges by searching for and flipping an augmenting path, as defined in Figure 1. In Algorithm 1, steps 2–4 describe the searching and flipping procedure. We call a single iteration of this procedure a *stage* (see Figure 2). Note that each stage finds exactly one augmenting path which increases the length with exactly one matching edge. Thus, the algorithm requires at most n stages to obtain all n matching edges, which form the optimal solution. The algorithm works well on the complete n^2 edged bigraph, and we show below that it also works for some non-complete bigraphs under certain conditions. We use the term *sparse bigraph* when a bigraph satisfies $|E| < n^2$.

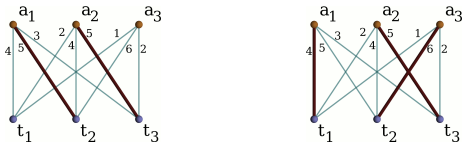


Fig. 2: (left) Two matched edges found after running two stages of the algorithm; (right) A perfect matching consisting of three matched edges is found after one more stage (by augmenting path $a_3 \to t_2 \to a_1 \to t_1$).

Theorem 3.1: If we define x_{ij} such that:

$$x_{ij} = \begin{cases} 1 & \text{if } e(i, j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

then for any sparse bigraph with each partition of size n ($|X| = |Y| = n$), the Hungarian Algorithm outputs an optimal solution if and only if there is a set $\{x_{ij}\}$ with $|\{x_{ij}\}| = n$ and satisfying:

$$\sum_{i=1}^{n} x_{ij} = 1, \;\; j = 1, \cdots, n \;\; \bigwedge \;\; \sum_{j=1}^{n} x_{ij} = 1, \; i = 1, \cdots, n. \quad (1)$$

Proof: Necessity follows the constraint definition of the assignment problem. Sufficiency: $|\{x_{ij}\}| = n$ implies at least n edges in the bigraph. Consider two cases:

1) if $|E| = n$ and all corresponding x_{ij} satisfy (1), indicating that each vertex has unit degree, then the edge set forms a perfect matching and optimal solution.

2) if $n < |E| < n^2$, and there is a subset $\{x_{ij}\}$ that satisfies (1), then the Hungarian algorithm can only fail to produce an optimal solution because it was halted before finishing n stages. The interruption must be that it failed to find an augmenting path. Given the procedure employed, this failure in a sparse bigraph means only an incomplete path with odd number of vertices exists. In other words, the bigraph is not fully connected and there are isolated sub-graphs, a contingency illustrated in the sub-graph formed with a_1, a_2 and t_2 in Figure 3(right). It is impossible to find a perfect matching in sub-bigraphs with odd numbers of vertices and, therefore, it is impossible to find a global matching solution for the whole graph, contradicting (1). ■

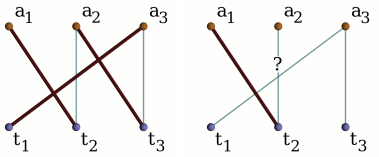


Fig. 3: (left) A matching in a sparse bigraph, weights omitted for clarity. In this example, the matching solution is unique as shown in the bold edges, and forms the optimum; (right) The algorithm halts on this arrangement since no augmenting path is rooted at a_2. No perfect matching is found.

Two corollaries follow for sparse bigraphs:

Corollary 3.2: In a sparse bigraph, a Hungarian stage continues without halting if and only if an augmenting path can be found connecting a free pairwise agent and task.

Corollary 3.3: Newly introduced agents and tasks can be incrementally assigned if and only if (*i*) one can connect the pairs to the built bigraph (*i.e.,* the bigraph property still holds after connection); and (*ii*) one can find augmenting paths between the sets of newly introduced agents and tasks.

IV. From Assignment Problem to Routing

We are given a set of deployed robots described as a weighted graph: robots at deployment locations are represented as vertices, while edges connect each robot to the locations they can move to, weighted by traversal cost. When new agents and target positions are inserted, the objective is to compute an efficient traversal that ensures every target is serviced by a robot. Target locations are treated as tasks and a task reassignment computed. This forms a sequence of robot/task replacements which is a routing through the graph.

The process is illustrated in Figure 4: in 4(a), a new agent-task pair is introduced to an existing perfect matching (higlighted in bold): vertex a_4 —leftmost vertex in top partition— and t_4 —rightmost vertex in bottom partition— are the newly inserted agent and task, respectively. A single incremental stage obtains the new assignment shown in 4(b). The vertices in each partition are shown on a 1D line in 4(b), but when the two partitions are vertices on a 2D plane, the bigraph can be visualized in 3D as in 4(b). The vertices in the bottom layer represent the task locations and the vertices in top layer are the agents currently assigned or to be assigned. Before the introduction of new agent-task pairs, the positions of vertices in top layer and bottom layer are vertically aligned and are pairwise matched with bold edges, indicating a "stabilized" state with all robots at assigned destinations. When a new agent-task pair is inserted, an additional stage of the Hungarian algorithm will find an augmenting path that will connect this new agent-task pair. The matched edges in this augmenting path provide the global routing solution. In Figure 4, for example, the augmenting path of $a_4 \to t_1 \to a_1 \to t_2 \to a_2 \to t_4$ means that agent a_4 should move to task location t_1, and a_1 move to t_2, and a_2 to t_4. This is reflected in the top layer in Figure 4(c); the projection of the matching (as arrows) to this plane is a routing solution.

Theorem 4.1: So long as the 2D network for the routing problem is connected, there are always augmenting paths.

Proof: The connectedness of the graph means a routing path exists between any two vertices. Since any routing path represents a specific solution, we can always find at least one solution for any agent-task pair. ■

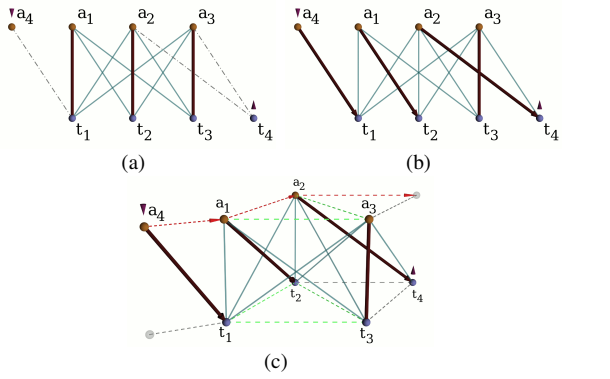


Fig. 4: The bigraph representing a single-pair routing problem. (a) An existing bigraph with perfect matching (bold edges) is supplemented with a new agent and task; (b) A new perfect matching is found after a single stage of the algorithm; (c) When the tasks and agents are actually planar locations then the bigraph picture is more precisely viewed in 3D, the top layer shows the resulting routing solution.

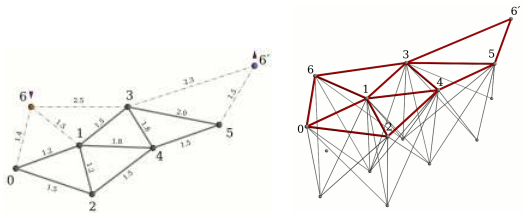


Fig. 5: An example topology with only nearest neighbors connected. (left) Vertices 6 and 6′ are newly inserted robot and task, respectively. (right) The corresponding **sparse** bigraph visualized in three dimensions.

This indicates that one may simply connect the newly introduced agent-task pair with a given graph and, provided one ensures each element is connected with at least one edge, the search for a specific parametrized path will yield a result.

V. Controlling Path Properties

Two properties of the resultant routing paths are important: (1) The summed cost/time for traversing the edges, which one wishes to minimize; (2) The number of reassigned agents, since usually each agent reassignment bears a cost.

The Hungarian algorithm maximizes utility, while a typical routing problem aims at minimizing the path length. We transform the minimization problem to a maximization one by negating all input values. Also, once all agents reach their targets they should maintain their "vertical" matching, thus, we assign the utility of this edge with a value at least as large as other outgoing edges. If we define utility matrix U:

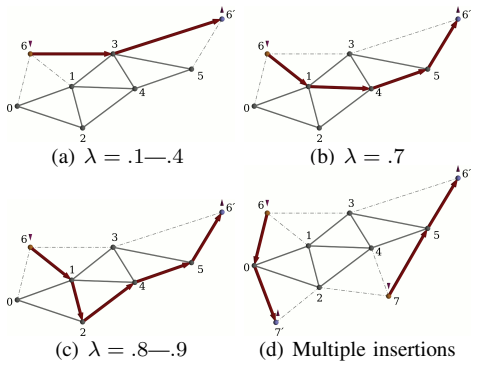
$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \dots & u_{nn} \end{pmatrix}$$

then $u_{ij} \leq 0$, and $u_{ii} \geq max\{u_{ij}\}, \forall j$. If $e(i,j) \notin E$, then we let $u_{ij} = -\infty$. The larger the value of u_{ii}, the more likely that edge $e(i,i)$ will remain matched in subsequent Hungarian stages. Let $diag(U) = [u_{11}, u_{22}, ..., u_{nn}]$. Then, with scaling parameter $\lambda \in [0,1]$, we obtain a new diagonally scaled utility matrix:

$$U' = U + (\lambda - 1)\, diag(U) = \begin{pmatrix} \lambda u_{11} & u_{12} & \dots & u_{1n} \\ u_{21} & \lambda u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \dots & \lambda u_{nn} \end{pmatrix}. \quad (2)$$

Only the diagonal is scaled, *i.e.*, $diag(U') = \lambda diag(U)$. We initialize each u_{ii} with $u_{ii} = \max_j\{u_{ij}\}$, then by tuning $\lambda \in [0,1]$, the diagonal utilities vary as $u'_{ii} \in [u_{ii}, 0]$. Consider the example in Figure 5. Vertices 0–5 are previously deployed robots and Vertices 6 and 6′ are the newly introduced agent and task. Figure 6 shows different paths as a function of λ, and Table I (top) provides statistics for total path length, number of reallocated robots, average path edge length, and the finishing time (proportional to the longest edge length). To compare the difference between the sparse and dense graphs, we also ran the experiments on the dense graph constructed from Figure 5, where each vertex is connected to all other vertices; the statistics appear in Table I (bottom). Both graph types are similar except when $\lambda \in [0, 0.4]$, since the dense graph adds new longer edges directly connect distant agent-task pairs and, thus, shorter alternative paths

can be found. The similarities between the sparse and dense graph solutions reflect the fact that edges connecting nearest neighbors, despite producing a sparse graph, permit most paths to be captured, a useful result for applications since edges may also represent sensing or communication links.



(a) $\lambda = .1$—$.4$ (b) $\lambda = .7$ (c) $\lambda = .8$—$.9$ (d) Multiple insertions

Fig. 6: (a)—(c) Examples of routing results under different λ values. (d) The routing paths of two inserted agent-task pairs.

We are also interested simultaneously inserting multiple agent-task pairs, as illustrated in Figure 6(d). Following Corollary 3.3, this can solved by running multiple Hungarian stages on the new tree with exposed roots at the inserted agents. (The condition in Theorem 4.1 must be made slightly stronger to guarantee the existence of a matching.)

TABLE I: GRAPH PATHS PROPERTIES

λ	Total Length	Reallocation Number	Average Length	Finishing Time	
0–.4	4.8	2	2.4	2.5	Sparse Graphs
.5–.6	5.3	3	1.77	2.5	
.7	6.3	4	1.58	1.8	
.8–.9	7.2	5	1.44	1.5	
1	8.3	6	1.38	1.5	
0–.3	4.3	1	4.3	4.3	Dense Graphs
.4	4.8	2	2.4	2.5	
.5–.6	5.3	3	1.77	2.5	
.7	6.3	4	1.58	1.8	
.8–.9	7.2	5	1.44	1.5	
1	8.3	6	1.38	1.5	

Optimality of the assignment is guaranteed even when multiple robots/tasks are added: although scaling the diagonal adjusts multiple utility values, doing so does not affect the existing feasibility which means that no additional stages are needed to "repair" the graph. This constrasts with the dynamic Hungarian method [6], which must adjust values by first removing and then reinserting them.

VI. EXPERIMENTS AND RESULTS

Two complementary forms of evaluation were conducted: (1) a simulation study systematically evaluating the effect of λ and degree on large graphs ($n = 300$); (2) demonstration on physical robots ($n = 6$) ensuring no unreasonable assumptions or simplifications have been made.

Influence of degree: Figures 7(a)–(d) are examples illustrating paths in graphs of different degrees of 50, 30, 10, and 5, respectively, with fixed $\lambda = 0.7$. The newly inserted agent-task pair is located in the bottom left corner (robot) and upper right corner (task). The paths have few changes for graphs with degrees above 10, but a large change happens

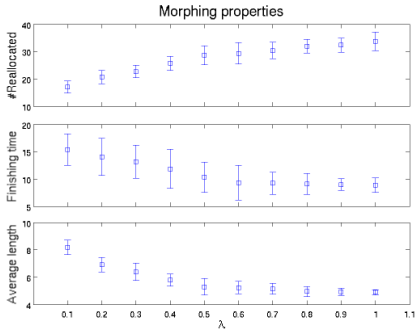


Fig. 8: Properties of resultant routes as a function of tuning parameter λ.

when the degree is 5. Experiments across a range of graphs yielded following observation: regardless of size, graphs with degree of above ~ 10 produce very similar paths, but this is not the case with degree less than ~ 7. This characteristic is important in practical applications where the graph represents communication with few neighboring robots (see further discussion in Section VII).

Influence of λ: Figures 7(e)–(h) show paths "straighten" and grow shorter as λ decreases from 1 to 0.1. Keeping degree $k = 10$, statistics from ten experiments for each λ were collected. Figure 8 plots: (1) the number of robots reallocated; (2) the route finishing time; and (3) the average distance each robot moves. As λ increases, the number of reallocated robots increase, whereas both the finishing time and average moving distance decrease, and the trend flattens out. Rates of change are biggest when $\lambda < 0.5$.

Generality: We also tested cases when multiple agent-task pairs are introduced. Figures 7(i)–(k) show two agent-task pairs inserted at the same time: a robot at each lower corner and a task at each upper one. Figures 7(i) and 7(j) show two separate paths when a one single pair is inserted, and 7(k) is the case when the two pairs are added simultaneously. The new paths in Figure 7(k) have higher quality because the routing solutions generated from the Hungarian algorithm represent a (global) optimal matching. Figure 7(l) shows that the algorithm also works in higher dimensions.

A. Comparison with a Standard Shortest Path Algorithm

As λ decreases the total path length of the routing path is reduced; how short will the path be when $\lambda = 0$? We compared the shortest paths computed from our morphing algorithm with Dijkstra's well-known single source shortest path (SSSP) method. In most cases, paths computed by the two methods are identical; when not exactly identical, the paths' lengths have a relative difference in length of a few percent ($\frac{len(morph)-len(dijkstra)}{len(dijkstra)} < 2\%$). Figure 9 compares two such paths. Although the solutions are not always identical, paths with minimum λ do approximate Dijkstra's shortest path for practical purposes. The advantage of the proposed method over other SSSP algorithms lies in its low computational complexity for sparse graphs, the incremental paradigm, and its tunable path outputs.

B. Physical Robot Experiment

We also verified our algorithm with a physical robot implementation. To construct the sparse bigraph, the algorithm

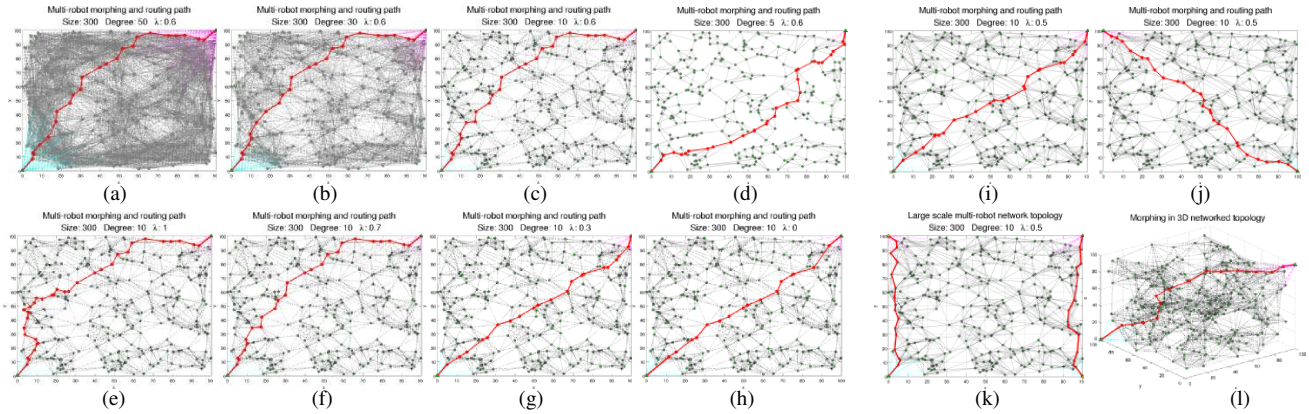


Fig. 7: (a)–(d) Paths produced in topologies of varying degree; (e)–(h) Paths of varying λ; (i)–(k) Multiple optimal paths; (l) Path in a 3D scenario.

does not need each robot's position, but requires instead the nearest neighbors' distances for each robot. If, however, the robots' positions are known, bigraph construction is straightforward. In our scenario, robots localize themselves and then we assign them target locations. New robot and task pairs are added thereafter.

We used six iRobot create robots (see Figure 10) equipped with ASUS EEE netbooks and Hokuyo URG-04LX-UG01 laser sensors. Each robot is provided with a scale map of our building and uses a particle filter [13] available as part of the *player* package [14] to localize itself. The netbooks process all the sensor input, calculate the utility estimates, provide wireless communication, log the data, *etc.* Utilities are computed by negating the path length between a robot and the desired designation, which is computed as the sum of all the path segments between the waypoints generated by the planner. Each robot follows this series of waypoints to reach the assigned task location (see Figure 10(a)).

The robots communicate with each other using UDP: each robot listens for messages sent by teammates. When a new robot-task pair is inserted, the new robot first queries the bigraph from the deployed robots, connecting itself and adding the associated task to the 3D bigraph. Since the deployed system already has all robot-task pairs matched, the new robot runs one stage of the algorithm to compute the new matching. When multiple robots are introduced, one robot is randomly selected from the set of new robots to be responsible for constructing the new bigraph. Finally, this robot runs the requisite stages ($\#stages = min\{\#robots, \#tasks\}$), to obtain a new allocation solution, which it then broadcasts to every member as a routing commitment.

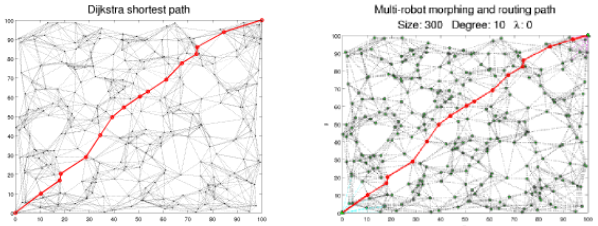


Fig. 9: (a) Shortest path from Dijkstra's algorithm; (b) Path produced by the proposed incremental matching method when $\lambda = 0$.

We placed task locations uniformly in the left and right corridors as shown in Figure 10. The robots are started at random initial locations and, after localizing themselves, they move to the designated (initially assigned) task locations. Once all robots have reached their respective destinations, a new robot-task pair is inserted as shown in Figure 10(a). The new robot "morphs" to its task location via a routing path: either left or right traversal in this case. We manipulated the degree of vertices in the graph and the parameter λ. Note that the graph is purposely unbalanced so that the left path always has more vertices. Figure 11 shows the results of a run on the dense graph that connects all robot-robot pairs. The horizontal axis is the degree of unbalancedness in robots on left and right paths, *e.g.*, $l2$-$r1$ means there are 2 robots in the left path and 1 robot in the right path. The result indicates that, generally, the paths switch from one to the other for a value of λ between $0.3 \sim 0.4$. This is consistent with the conclusion from simulation that path switching happens more often as $\lambda \to 0$. Additionally, we tested the situation where each robot is connected to only its nearest neighbor. The result differs only at $\lambda = 0$: in a dense graph the morphing solution directly allocates the new robot to the new task, whereas in sparse graph it chooses the path on the right (the shortest routing), as expected. We also tested the cases of inserting multiple robot-task pairs, and the results show that for any λ, the robots always morph to the nearest task locations with shortest routes. Figure 10(b) is an example showing two robot-task pairs added to a system already containing 4 robots, and the algorithm produces the two shortest paths, which is globally optimal.

C. Analysis

In this paper we consider the task of computing an optimal routing which formulate as an assignment problem. We believe the following are advantages over other algorithms:

1) Low computational complexity: Each stage requires $O(n^2)$ for dense bigraph [5], but for the sparse bigraph of edge degree k, the complexity is bounded by $O(kn)$, *i.e.,* linear in the system size. When the problem arises from a network of robots communicating with nearby neighbors, this latter bound applies.

2) The incremental paradigm: Newly inserted robot-task pairs become deployed parts of the system after the routing is completed, and only neighboring information need be updated to be ready for further robot-task insertion. Moreover, because of the special aligned matching relation, removal of a vertical matched pair in the 3D bigraph is as simple as disconnecting incoming and outgoing edges.

3) Tunable paths: The algorithm does not modify any of the 2D network information but achieves different assignment solutions by scaling a special "virtual" edge for each vertex via λ, which can be easily computed and visualized.

VII. General applicability of the formulation

The approach is applicable more generally than the specific scenario used above. It is not necessary for the routing graph edge weights to be Euclidean distances, nor necessarily for the robots to be localized. The optimization criteria do not depend on particular locations, rather known or estimated distances or traversability costs between nearby robots can suffice. For example, a swarm of wirelessly networked robots without localization capabilities can use local radio signal strength for both cost estimation , and actual (gradient-based) traversal from one node to another. In this case, minimal distance versus time can be important as one wishes to minimize time while also limiting the number of slow, unreliable searching movements (*i.e.,* bounding total reallocations).

The model can also be extended to topological change of large multi-agent systems by disconnecting individual agent-task pairs and reconnecting them with new task assignments, one may "morph" the topology using the approach. This can be used in the simulation of flocking formations, or global shape morphing of particle systems, or the topological variations for mobile robots moving together but adjusting for navigation in irregular and confined environments.

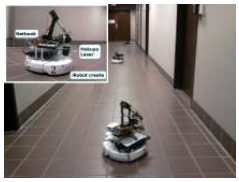
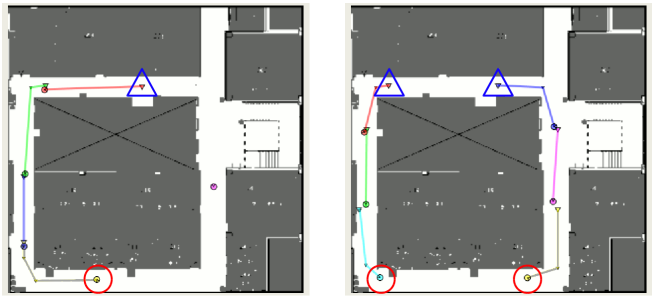


Fig. 10: Robots on the left actively localize and execute tasks in the corridor of our research building; the map of the building is shown below. The inset picture shows the physical robot used in the experiments, in detail.



(a) (b)

(a) One robot-task pair is inserted, the new robot is circled in lower corridor and the new task location is labelled with triangle in upper corridor. (b) Two robot-task pairs being inserted.

VIII. Conclusion

In this paper solves a problem arising as new robots and tasks are added to already deployed units: it moves the new robots into maximally useful positions while adjusting the deployed robots only as necessary and where doing so can save costs. The result is a physical routing of robots through the graph encoding navigation constraints. Our approach applies a variant of the Hungarian algorithm to compute optimal assignments incrementally in sparse bipartite graphs. The graph involved is a two layered bigraph with weighted links connected in 3D. Multiple solutions are produced by adjusting the spacing between the two layers, allowing one to balance optimization criteria to minimize total distance, the level of disruption caused by redeploying robots and the time to complete the adjustment.

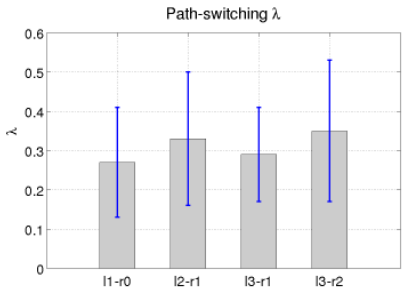


Fig. 11: A plot of the watershed λ that switches between the left and right paths. The horizontal axis value $1\langle\alpha\rangle$-$r\langle\beta\rangle$ denotes the left path has $\langle\alpha\rangle$ robots (vertices) and the right path contains $\langle\beta\rangle$ robots.

References

[1] S. Topal, I. Erkmen, and A. M. Erkmen, "Morphing a Mobile Robot Network to Dynamic Task Changes over Time and Space," in *Intl. Conf. on Automation, Robotics and Control Sys.*, 2009, pp. 192–199.

[2] A. Howard, M. J. Matarić, and G. S. Sukhatme, "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks," *Autonomous Robots 13(2)*, pp. 113–126, 2002.

[3] T. Balch and R. C. Arkin, "Behavior-based formation control for multi-robot teams," *Trans. on Robotics and Auto. 14(6)*, pp. 926–939, 1997.

[4] R. Karmani, T. Latvala, and G. Agha, "On scaling multi-agent task reallocation using market-based approach," in *Intl Conf. on Self-Adaptive and Self-Organizing Systems*, 2007, pp. 173–182.

[5] I. H. Toroslu and G. Üçoluk, "Incremental Assignment Problem," *Information Sciences*, vol. 177, no. 6, pp. 1523–1529, Mar. 2007.

[6] G. A. Mills-Tettey, A. Stentz, and M. B. Dias, "The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs," Carnegie Mellon University, Tech. Rep. CMU-RI-TR-07-27, 2007.

[7] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, "Distributed multi-robot task assignment and formation control," in *IEEE Intl. Conf on Robotics and Automation*, 2008, pp. 128–133.

[8] W. Ren and N. Sorensen, "Distributed coordination architecture for multi-robot formation control," *Robotics and Autonomous Systems 56(4)*, pp. 324–333, 2008.

[9] W.-M. Shen and B. Salemi, "Distributed and dynamic task reallocation in robot organizations," in *IEEE Intl. Conf. on Robotics and Automation*, 2002, pp. 1019–1024.

[10] G. Elkaim, R. Kelbley, and A. M. Erkmen, "A Lightweight Formation Control Methodology for a Swarm of Non-Holonomic Vehicles," in *IEEE Aerospace Conference*, Big Sky, MT, 2006.

[11] N. Agmon, G. A. Kaminka, S. Kraus, and M. Traub, "Task Reallocation in Multi-Robot Formations," *J. of Physical Agents 4(2)*, 2010.

[12] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the SIAM 5(1)*, pp. 32–38, 1957.

[13] D. Fox, "KLD-sampling: Adaptive particle filters," in *Advances in Neural Information Processing Systems (NIPS-14)*, 2001, pp. 713–720.

[14] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proc. Intl. Conf. on Advanced Robotics*, 2003.