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

Lantao Liu and Dylan A. Shell
Department of Computer Science & Engineering
Texas A&M University
College Station, Texas, USA
Email: {lantao,dshell}@cse.tamu.edu

Abstract— **Many multi-robot scenarios involve navigation of a set of networked robots through a constrained environment to achieve coverage, maintain a predefined shape, sense at predefined locations, or to satisfy some other distance-defined property. When new robots and tasks are added to a network of already deployed interchangeable robots, a trade-off arises in seeking to minimize cost to execute the tasks and the level of disruption to the system. This paper examines a navigation-oriented variant of this problem in which robots are physically routed through an existing network. We propose a parametrizable method to tune emphasis between minimizing global travel cost (or energy, or distance), minimizing interruption (*i.e.,* obtaining the fewest number of robot reassignments), reducing travel distance per robot, and completing all operations as soon as possible. Since these are related optimization criteria, a single parameter provides sufficient flexibility to balance between them.**

Paths through the network are computed via a task-allocation formulation in which destination locations of newly deployed robots are added as tasks to an existing allocation. We adapt the graph matching variant of the Hungarian Algorithm—originally designed to solve the optimal assignment problem in complete bipartite graphs—to construct routing paths in sparse networks. We do this by constructing a three-dimensional graph that incorporates logical aspects of the Hungarian bipartite graph, and spatial elements of the Euclidean graph. The approach has several useful features including being particularly effective at generating multiple simultaneous, non-interfering paths. When new agent-task pairs are inserted, the assignment is globally reallocated in an incremental fashion so that it requires only linear time when the robots' traversal options have bounded degree. The algorithm is studied systematically in simulation and also validated with physical robots.

I. Introduction

Several problems, across a wide variety of application domains, for which coordinated multi-robot systems are a potential practical solution can be formulated as the task of moving a team of mobile robots from a set of current locations to a set of goal (or target) locations while minimizing some collective cost. This paper examines an algorithm for an incremental form of this problem: a set of tasks and robots are added to a network of already deployed robots and the objective is to use both newly added and existing robots to cover the previous deployment locations in addition to the new targets. This paper is a study of an efficient and flexible method for solving the problem of deciding which robots should be moved and to where. We call this the physical routing problem.

In order to demonstrate the generality of the underlying optimization problem, we consider a few examples where one may seek to compute a physical routing:

Deployment The work of Howard *et al.* [2002] is a good illustrating example of this problem being applied inductively to self-deploy a distributed sensor-actuator network. The first sensor-actuator node (really a robot) picks a location and moves to that position. Thereafter, nodes are added one by one to a set of locations that fill the space. The interchangeability of the robotic nodes means that the robots move out in a breadth first fashion as if emanating from a source.

Shape morphing Recently described by Topal *et al.* [2009], this problem involves having a group of robots transition smoothly from one shape to another in a cohesive way. This can be tackled as follows: robots assess whether their current position falls within the target shape or not; if it does, they consider themselves part of the already deployed system. The remaining robots are assigned new target positions in the goal shape.

Ad hoc network repair Suppose that the currently deployed robots form a wireless network, then newly added robots can be deployed either to strengthen existing connections or to occupy locations where failed robots had previously been located. Global positions are not necessary in order for the approach to apply in this case, since movements generated as routing traversals along the graph can be used for local gradient following on signal strength in order to seek out useful positions. (This is the problem which originally motivated this research as part of the DARPA LANdroids project.)

There may be multiple satisfactory solutions to the physical routing problem as described. Most simply, a newly added robot may directly move toward the nearest new target. If there are constraints on which robots may travel where, or if the time required to reach all the targets is important, then it may be better to have one of the already deployed robots travel to the target and another robot fill the resultant vacancy. With interchangeable robots, a whole chain of robots may simultaneously move toward the target, with each robot taking the place of the last. Thus, a robot is physically "routed" through the network.

In optimizing system performance important aspects to consider include the total distance travelled, the level of

disruption caused by redeploying robots and the time taken to complete the traversal by all the robots. The relative importance of these aspects will depend on the particular scenario and problem involved. This paper introduces a parametrizable method which allows one to tune the path properties among the set of performance criteria as desired. The 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. For the addition of a new agent-target pair, the solution can be computed in linear time when—as is frequently the case—the problem instance is represented by a graph with bounded degree. Importantly, the method allows simultaneous addition of multiple agents and targets; we illustrate that sequential treatment (*e.g.,* through repeated execution of existing single source shortest path algorithms) may fail to find the globally optimal solution. The approach we employ is to compute the tunable routing by solving an assignment problem over a bipartite graph (or bigraph) derived from a graph representing the spatial locations of the networked robots. The optimal assignment is then computed on the (potentially incomplete) bigraph in an incremental manner.

This paper contributes an efficient, tunable approach that can adjust the optimization output to trade between individual travel cost/time and system reallocation-disruption/shortest-path solutions. The approach is based on extending the classic Hungarian Method from dense (or complete) bigraphs to operate on sparse bigraphs. These bigraphs arise through a special construction of matching graphs from a Euclidean graph encoding the network of robots and their traversability constraints. This construction connects the conventional matching problem with the routing problem. The matching graphs that arise have a spatial interpretation, allowing one to see how geometric properties are inherited from the underlying Euclidean graph. They also maintain their logical (or topological) features. We analyze the conditions under which valid matching solutions are produced and how feasible routing paths result. Intensive simulation and physical robot experiments constitute the empirical validation of the proposed method, including comparison with algorithms for computing standard paths. In particular, the paper illustrates how the approach can address the problem of concurrent multiple paths arising from the insertion of multiple robots and targets.

II. Related Work

In this section we review work that are related to the proposed method. Since the proposed approach is a routing method but is achieved by running an optimal assignment algorithm —the Hungarian Algorithm— the review focuses on the aspects of optimal assignment algorithms (with special focus on the Hungarian Algorithm), popular uses of the assignment problem in multi-robot systems, and routing methods derived from assignment solutions.

A. Assignment Algorithms and the Hungarian Method

The multi-robot task allocation problem studies the problem of assigning robots to tasks in order to maximize the performance (or minimize the cost) of the whole team. Different assignment models have arisen in order to formulate and address differing task allocation scenarios (see Gerkey and Matarić [2004] for a review). An important dimension within the task allocation taxonomy is the cardinality of the mapping between robots and tasks, *viz.*, whether the assignment relationship between robots and tasks is one-to-one, one-to-many, many-to-one, or many-to-many. In this work, we are interested in the problem of assigning every robot to a unique task (one-to-one mapping), which is the most basic and probably most widely investigated assignment problem.

Numerous fast optimal assignment algorithms with time complexity bounded by $O(n^4)$ have been proposed during last century (see Burkard *et al.* [2009] for a review), among which the most well-known algorithm is probably the Hungarian Method (or Hungarian Algorithm) that was invented by Kuhn [1955] and refined by Munkres [1957]. While this original algorithm is tableau based, the bigraph variant of the algorithm with time complexity of only $O(n^3)$ has attracted much research.

One useful feature of the Hungarian Algorithm is that it can be executed incrementally. Toroslu and Üçoluk [2007] proposed the *incremental assignment algorithm* that finds a new solution with cost $O(n^2)$ after a new pair of vertices and their incident edges are added to a weighted bigraph with known matching. Motivated by examples in the transportation domain, Mills-Tettey *et al.* [2007] proposed the *dynamic Hungarian Algorithm* to handle cost changes with time complexity of $O(kn^2)$, where k is the number of cost changes. Both incremental and dynamic Hungarian Algorithms can be easily extended to address deletions with the same complexities. We proposed the *interval Hungarian Algorithm* to assign whilst evaluating the uncertainty in multi-robot task allocation problems [Liu and Shell, 2011]: with an existing solution, the *interval* algorithm needs an extra $O(n^3)$ to compute allowable cost intervals within which cost perturbations will not violate optimality of the current solution.

B. Optimal Assignment Problem in Multi-robot Systems

The problem addressed by this paper relates directly to redeployment of robots to tasks. Note that, here "redeployment" means more than merely "reassignment", because it includes the control policy which produces patterns of motion among the robots.

Work specifically addressing reassignment includes that of Karmani *et al.* [2007] and Shen and Salemi [2002], both of which consider assignment dynamics during the task commitment and trade the local tasks whenever profitable. In Karmani *et al.* [2007] a TSP (Traveling Salesman Problem)-like search strategy is given whereas in Shen and Salemi [2002] heuristic local searching algorithms are proposed. Decentralized reassignment research such as Ayanian *et al.* [2012] and Liu and

Shell [2012a] share similar basic ideas: an initial (possibly random) assignment is assigned for the whole system then during the task execution, small cliques can be formed under certain rules (Ayanian *et al.* [2012] forms a small group constrained by local communication whereas Liu and Shell [2012a] partitions the system into appropriate sized cliques by processing the cost matrix) so that robots inside a clique can refine the sub-assignment problems.

Some formation control problems can also be regarded as reassignment problems if one simply reassign those units with a significant discrepancy in their current locations from the final goal formation. Notable recent examples of formation control include that of Michael *et al.* [2008], Ravichandran *et al.* [2007], Ren and Sorensen [2008], and Alonso-Mora *et al.* [2011]). Smoothly shifting 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. For instance, Topal *et al.* [2009] morphs a formation by capturing the network configuration by analyzing adjacency matrix eigenvalues, and Elkaim *et al.* [2006] provides a lightweight artificial potential based method to control individual node movement and globally morph the formation. A clear, recent example of reallocation and formation work together is that of Agmon *et al.* [2010]. The authors designed a polynomial time graph-based method to extract a subset of the robots from a coordinated group so that this subset can perform a new task while minimizing the cost of the interacting with the remaining group.

Several pieces of work on multi-robot systems have used the Hungarian Algorithm to solve the assignment problem as part of some more complex architecture for addressing problems in a variety of domains. The vast majority of existing work does not modify either the Hungarian Algorithm or the bigraph, but simply provides the data to a solver to obtain matching solutions (for example, see works of Wurm *et al.* [2008], Elmaliach *et al.* [2009], and Liu and Shell [2012a]). Besides the incremental approaches such as Mills-Tettey *et al.* [2007] which dynamically deletes matched edges and inserts new bigraph edges, and Liu and Shell [2011] which hides matched (unmatched) edges in order to find the weight bounds within which current solution remains optimal, a good example that utilizes the bigraph structure is Giordani *et al.* [2010] in which a communication protocol is designed to route along the augmenting path in order to distribute the computation of an assignment.

The work in this paper is also based on the bigraph variant of Hungarian Algorithm and runs incrementally. An important contribution of this work lies in a novel transform and manipulation on the bigraph. More specifically, (1.) the bigraph is sparse (also one byproduct of sparsity is the linear time solution when the vertices have bounded degree), (2.) the bigraph incorporates the spatial embedding and can be interpreted in the three-dimensional space, (3.) a set of parametrized edges allows the algorithm to yield differing solutions that maps to different spatial routing trajectories. Note that, we do not modify the Hungarian Algorithm but simply employ it as a solver, although we do require that the implementation of the Hungarian Algorithm handles sparse bigraph and automatically halts if a solution does not exist.

C. Routing with the Assignment Solution

Many classic graph algorithms could be applied to compute a redeployment routing. Well-known single source shortest path (SSSP) methods, such as Dijkstra's algorithm, can be used to efficiently seek the shortest path between any pair of start and goal vertices. Redeployment would involve all the robots along this path moving one hop toward the goal. Although not explicitly using Dijkstra's algorithm, Howard *et al.* [2002] employ an analogous technique via dynamic programming to find a shortest path in their incremental deployment scenario. However, these shortest path algorithms operate directly on a form of two-dimensional graph with single optimization objective or criterion.

The fact that a relationship exists between shortest path and assignment problems was previously investigated with techniques such as the general primal-dual methods [Papadimitriou and Steiglitz, 1998] and optimal auction algorithms [Bertsekas, 1991; 1990]. These methods transform the assignment problem to the single source single destination shortest path searching problem and aim at solving the problem as fast as possible. For example, a notable method of Bertsekas [1991] utilizes the well-known *auction* method (an optimal assignment algorithm) to search the shortest path on a directed graph. Distinct from those approaches, this paper offers the following perspective: spatial redeployment of a multi-robot team is achieved by routing agents on the two-dimensional topology (a standard undirected graph) but doing this by treating a construction which is a three-dimensional representation of a corresponding bigraph. Beyond the novelty of this representation, a primary advantage of the approach is that it incorporates both (metric) traversal information and reallocation (logical) costs simultaneously. The strength in the proposed approach is not merely that short paths can be generated, but rather that one can parameterize optimization criteria to trade among a set of performance objectives.

Moreover, the characteristics of matching solution (*i.e.*, assignment solution) reside in the matching graph guarantees that the produced routing trajectories are conflict-free. Since the Hungarian Algorithm is an optimal assignment algorithm, the paths computed from it also possess the property of global optimality, *e.g.*, the overall travel cost can be minimized, the number of robots redeployed can be the fewest, and the number of conflict-free paths that are allowed can reach the maximum.

III. Preliminaries: Definitions and Background

This section provides background information required for the remainder of the paper. First, the variant of the optimal assignment problem related to this work is described and formulated in Section III-A. In Section III-B we define and describe the two graphs to which we have already alluded. The detail of their relationship, which underlies the design

of the method, is presented in the section following these preliminaries. The final element presented in this section is the matching version of the Hungarian Algorithm, a famous *primal-dual* assignment method. This forms Section III-C and is intended both to aid subsequent discussion and to allow the reader to make use of the paper in a standalone fashion.

A. Assignment Problem Formulations

The incremental variant of the *assignment problem* that we focus on computes a matching between two distinct sets. This form of assignment arises in task-allocation problems denoted by SR-ST-IA in the Gerkey and Matarić [2004] taxonomy; the assignment can be seen as a mapping from each robot to exactly one task so that no two robots will be mapped to an identical task. In detail, an assignment $\mathcal{A}$ for a multi-robot system consists of a set of robots R and a set of tasks T. Given a cost matrix $C = (c_{ij})_{n\times n}$, where c_{ij} denotes the cost of having robot i perform task j, the objective is to find an assignment permutation $\varphi : \{i\} \to \{j\}$ such that the overall cost $c(\varphi) = \sum_{i=1}^{n} c(i, \varphi(i))$ is minimized. Since the cost of assigning robot i to task j can be different from that of assigning robot j to task i, *i.e.*, $c_{ij} \neq c_{ji}$, the cost matrix need not be symmetric. To ease the analysis, we assume the sets of robots and tasks have identical cardinality, $|R| = |T| = n$, although the algorithms described herein handle cases with $|R| < |T|$ too. In fact, we take special care to consider assignment scenarios in which an underlying matching graph is sparse.

This matching problem can be formulated equivalently by a pair of linear programs. The first is the *primal* program, which is a minimization formulation:

$$\begin{aligned} \text{minimize } f_p(\mathbf{x}) &= \sum_i \sum_j c_{ij} x_{ij}, \\ \text{subject to } & \sum_j x_{ij} = 1, \ \forall i, \\ & \sum_i x_{ij} = 1, \ \forall j, \\ & x_{ij} \geq 0 \quad \forall (i,j), \end{aligned} \tag{1}$$

where an optimal solution eventually is an extreme point of its feasible set (x_{ij} equals to either 0 or 1 in the solution).

The constraints $\sum_j x_{ij} = 1$ and $\sum_i x_{ij} = 1$ capture the *mutual exclusion* property which restricts each robot to be assigned to exactly one task and each task to be allocated to an unique robot, respectively.

There are corresponding *dual* vectors $\mathbf{p} = \{p_i\}$ and $\mathbf{q} = \{q_j\}$, with dual linear program:

$$\text{maximize } f_d(\mathbf{p}, \mathbf{q}) = \sum_i p_i + \sum_j q_j, \tag{2}$$

$$\text{subject to } \ p_i + q_j \leq c_{ij}, \quad \forall (i,j). \tag{3}$$

The *duality theorems* show that a pair of feasible primal and dual solutions are optimal if and only if the following is satisfied:

$$x_{ij}(c_{ij} - p_i - q_j) = 0, \qquad \forall (i,j). \tag{4}$$

This *complementary slackness* equation reveals the property of orthogonality between the primal and dual variables.

B. Graph Representations

This work is based on two graphs: the *Euclidean graph* and the *bipartite graph* (or *bigraph* for short).

What we term the Euclidean graph is a standard graph $G = (V, E)$ with a metric embedding so that the vertices in V describe locations and edges in E express distances between the vertex pairs. We let each vertex of G denote an agent, and let $w(i, j) = d(i, j)$ represent the weight of edge $e(i, j) \in E$, where $d(i, j)$ is the travel distance between agent pair (i, j). In addition, we observe that traversability constraints, limited sensing/communication ranges, and so on, mean that the graph G is likely to be sparse.

We call the graph describing logical aspects of the matching problem the bigraph. This is a graph $\tilde{G} = (X, Y, \tilde{E})$ whose vertices can be divided into two independent sets X and Y such that every edge $\tilde{e}(x, y) \in \tilde{E}$ connects a vertex $x \in X$ to one $y \in Y$. In our problem, bigraph $\tilde{G}$ is another representation for cost matrix $C = (c_{ij})_{n\times n}$, where X and Y respectively denote the set of agents and tasks, and the set $\tilde{E} = \{\tilde{e}(i, j)\}$ are edges weighted by the costs ($\tilde{w}(i, j) = c_{ij} = d(i, j)$) between associated agent-task pairs (i, j).

In a bigraph, a *matching* is a set of edges such that no two edges share a vertex in common. A *perfect matching* is a matching which matches all vertices, *i.e.*, each vertex of the bigraph is on a unique matching edge. Since the bigraph represents matchings naturally, sometimes it is also called the *matching graph*.

C. $O(n^3)$ Hungarian Algorithm with Matching Graph

The Hungarian Algorithm can efficiently solve an $n \times n$ assignment problem in $O(n^3)$ time. It is presented using the bigraph representation in Algorithm 1. The assignment problem is a matching problem where the goal is to find a maximally weighted perfect matching M. This implies that each agent in X is uniquely assigned to a task in Y.

Equation (4) gives conditions for optimality as the product of two factors. There are three cases:

1) $x_{ij} = 1$ and $c_{ij} - p_i - q_j = 0$. The associated edges $\tilde{e}(i, j) \in \tilde{E}$ are termed *matched edges*.
2) $x_{ij} = 0$ and $c_{ij} > p_i + q_j$. Edges satisfying these conditions are called *unmatched edges*.
3) $x_{ij} = 0$ and $c_{ij} - p_i - q_j = 0$. These edges are unmatched but should be seen as potential candidates which may become matched.

An edge $\tilde{e}(i, j)$ that satisfies $c_{ij} - p_i - q_j = 0$ is called *admissible* (either matched or potentially matched) and is recorded in the so-called *equality graph* G_e during execution of the Hungarian Algorithm.

The Hungarian Algorithm grows a matching by searching for a path, called an *augmenting path*, which has an alternating sequence of matched and unmatched edges with free end

Algorithm 1 The Hungarian Algorithm

Input:

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

Output:

A perfect matching M.

1: Initiate dual variables by $p_i(x) = \min_{\forall j}\{c_{ij}\}$ and $q_i(y) = \{0\}$,$\forall i \in [1, n]$; Initial matching $M = \varnothing$.

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 values of dual variables:
$\delta = min_{\forall x \in S, y \in Y \setminus T}\{\tilde{w}(x, y) - p(x) - q(y)\}$
$p(x) = p(x) + \delta \quad$ if $x \in S$,
$q(y) = q(y) - \delta \quad$ if $y \in T$,

4: If $N(S) \neq T$, pick $y \in N(S) \setminus T$.

(a) If y exposed, then $u \rightsquigarrow y$ is an augmenting path, then augment matching M and go to step 2.

(b) If y matched, say to z, extend the tree: $S \leftarrow S \bigcup \{z\}$, $T \leftarrow T \bigcup \{y\}$, and go to step 3.

Notes & Definitions:

- $\tilde{w}(x, y) = c_{xy}$, is the weight of edge $\tilde{e}(x, y)$.
- $N(u) = \{v \mid \tilde{e}(u, v) \in G_e\}$, where $G_e = \{\tilde{e}(x, y) \mid p(x) + q(y) = \tilde{w}(x, y)\}$ is the equality graph. If S is a set, $N(S) = \bigcup_{\forall u \in S} N(u)$.

(a) (b)

Fig. 1: An illustration of an *augmenting path*. (a) a sequence of alternating matched (bold) and unmatched (non-bold) edges, with the first edge and last edge unmatched. Such paths possess an odd number of edges and an even number of vertices; (b) the set of matched edges is *augmented* when the matched and unmatched edges are flipped.

nodes, as illustrated in Fig. 1. The algorithm augments the size of a matching by simultaneously flipping the matched and unmatched edges in the augmenting path, so that all matched edges become unmatched whereas all unmatched edges become matched. (Formal definitions of these operations on matchings are omitted, we suggest the reader refer to [Lovász and Plummer, 1986] for a detailed treatment.) In Algorithm 1, steps 2 to 4 describe the procedure of seeking and flipping an augmenting path. We call a single iteration of this procedure a *stage* (see Fig. 2). Note that each stage finds exactly one augmenting path which increases the size of the matching by exactly one edge. Thus, the algorithm requires at most n stages to obtain all n matched edges, thereby forming the optimal assignment solution.

The algorithm is known to work well on the complete bigraph (with $n \times n$ edges) and we show it also works for some non-complete bigraphs under appropriate conditions. We use the term *sparse bigraph* if a bigraph has $|E| < n^2$.

Theorem 3.1: For any sparse bigraph with sets of cardinality n ($|X| = |Y| = n$), the Hungarian Algorithm proceeds without halting until it successfully outputs an optimal solution if and only if there is an edge set $\tilde{E}_s \subseteq \tilde{E}$ with cardinality $|\tilde{E}_s| = n$ and satisfying:

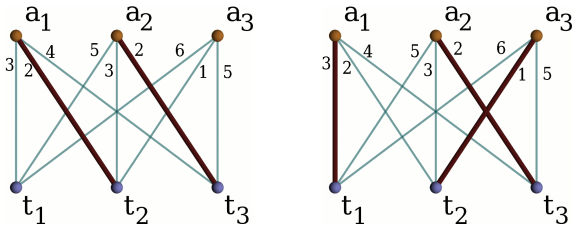
$$\begin{aligned} & x_{ij} = 1, \quad \forall (i, j) \mid_{\tilde{e}(i,j) \in \tilde{E}_s}, \\ & \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j, \\ & \sum_{j=1}^{n} x_{ij} = 1, \quad \forall i. \end{aligned} \tag{5}$$

Proof: Necessity follows the constraint definition of the assignment problem. Thus, consider sufficiency: $|\tilde{E}_s| = n$ implies that there must be at least n edges in the bigraph. We can prove sufficiency by considering two cases:

1) If the number of edges $|\tilde{E}| = n$ and all corresponding x_{ij} satisfy (5), this indicates each vertex in the bigraph has unit degree, thus, the n edges form a set which is a perfect matching. This must be an optimal solution on the basis of the property underlying the optimality condition of the Hungarian Algorithm itself.
2) If $n < |\tilde{E}| < n^2$, and there is a subset $\{\tilde{e}(i, j)\}$ that satisfies (5), then the Hungarian Algorithm can only fail to produce an optimal solution because it halted before finishing all n stages. Such an interruption may only occur because it failed to find an augmenting path. Failure to find an augmenting path must mean that only an incomplete path with odd number of vertices exists. Equivalently, the bigraph is not fully connected and there are isolated sub-graphs. This contingency is illustrated in the sub-graph formed with a_1, a_2 and t_2 in Fig. 3(b). 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 bigraph. Thereby, this contradicts (5).

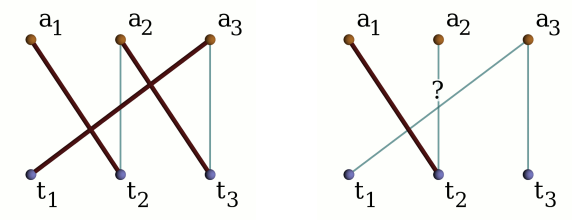
■

It has been noted elsewhere that adding dummy edges can convert a sparse bigraph to a complete (dense) bigraph.



(a) (b)

Fig. 2: (a) Two matched edges found after running two stages of the Hungarian Algorithm; (b) A perfect matching consisting of three matched edges is found after one additional stage (by augmenting path $a_3 \rightarrow t_2 \rightarrow a_1 \rightarrow t_1$).



(a) (b)

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

Dummy edges have the potential to become matched edges (and must do so, if the condition in Theorem 3.1 is violated) requiring additional special handling. With an implementation of the Hungarian Algorithm that takes as input a sparse bigraph form, the possibility of the solution being obtained on precisely those edges is determined automatically.

Two corollaries follow directly:

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: In a sparse bigraph, newly introduced agents and tasks can be incrementally assigned if and only if (*i*) one can connect the pairs to the already 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 the Assignment Problem to Routing

Suppose that the configuration of the robots after their initial deployment is given as a Euclidean graph $G = (V, E)$: robots at deployment locations are represented as vertices $v \in V$, and edges $e(u, v) \in E$ connect the location pairs that they can move to and are weighted by travel distances. Then when new agents and target positions are inserted at different locations, the objective is to compute a redeployment policy that ensures every target is efficiently serviced by a robot. One solution is to seek a sequence of robots so that each replaces the next to reach the target deployment configuration. This form of solution is what we term a "routing" because, much like a packet in a network, it can be visualized as the movement of a virtual baton which is propagated along a path. Of course, movement of this set of robots, in which each robot in the set moves to the location previously allocated to the next robot, may even occur concurrently to reach the target configuration efficiently.

The essence of this work is to synthesize a bigraph so that the matching for the optimal assignment problem on that bigraph is a routing solution. This method brings both spatial properties (from the geometry of Euclidean graph's embedding) and logical properties (represented in the bigraph) into consideration simultaneously. The attractive features that result include being able to parameterize optimization criteria and the ability to seek solutions for multiple new agents and tasks.

A. Matching Graph with Spatial Geometry

Conventionally, the bigraph $\tilde{G} = (X, Y, \tilde{E})$ and matchings computed on it have a topological interpretation. Even when they are weighted, nothing is assumed about the meaning of the edges themselves. Unlike the edges in the Euclidean graph $G = (V, E)$, which have geometric information, the matching edges in the bigraph represent only a logical description as they encode an assignment.

However, being undirected graphs, both have a common characterization: both the Euclidean graph and the bigraph can be represented with matrices: G can be represented with a symmetric adjacency matrix with $w(i, j)$ as entries, and $\tilde{G}$ can be represented with a non-symmetric cost matrix with $\tilde{w}(i, j)$ as entries. This suggests that, potentially, the bigraph may express spatial information *if the cost matrix is symmetric*. Thus, we let all off-diagonal entries of the two matrices have the following relationship:

$$\begin{aligned} \tilde{w}(i,j) = \tilde{w}(j,i) = w(i,j) = w(j,i), \\ \forall i \neq j, 1 \leq i, j \leq n, \end{aligned} \tag{6}$$

and thereby encoding a cost matrix which has become symmetric.

For an already deployed system with n robots, the cardinalities of vertices in G and $\tilde{G}$ satisfy

$$|V| = n \text{ and } |X| + |Y| = 2|V|, \tag{7}$$

and the cardinalities of edges in G and $\tilde{G}$ satisfy

$$|\tilde{E}| - n = 2|E|. \tag{8}$$

If we ignore the special n bigraph edges corresponding to the matrix diagonal entries for the moment, then Equations (7) and (8) show that both the vertices and edges are doubled when we relate those in $\tilde{G}$ to those in G. This relationship is the basic idea for the construction capable of unifying the two graph. Specifically, we obtain a mapping $\Omega : G \to \tilde{G}$ where, with known $G = (V, E)$, the mapping $\tilde{G} = \Omega(G) = (X, Y, \tilde{E})$ can be obtained in these steps:

1) Make a copy of vertices, $V' = V$, and let
$$X = V \text{ and } Y = V'. \tag{9}$$

2) Add the bigraph edges:
$\forall (i, j) \mid_{e(i,j) \in E, i,j \in V}$,
$$\begin{aligned} \tilde{E} \leftarrow \{\tilde{e}(i,j), \tilde{e}(j,i)\}, i \in X, j \in Y, \\ \tilde{w}(i,j) = \tilde{w}(j,i) = w(i,j). \end{aligned} \tag{10}$$

3) Add the special bigraph edges that correspond to the matrix diagonal:
$$\tilde{E} \leftarrow \tilde{E} \cup \{\tilde{e}(i,i)\}, \quad \forall i \in V. \tag{11}$$
The weights of these special edges are defined in a parametrized manner, as described below.

This graph may be imagined as if an identical copy of the Euclidean graph G had been created and placed over G. We can visualize it as if the copy is lifted so as to float above the first graph. Via this "extrusion" a three dimensional mesh is formed with two identical layers plus edges that connect the layers. Here the two layers correspond to the two partitions of a bigraph, *i.e.*, the bigraph vertex sets satisfy $X = Y = V$. Each edge $e(i,j) \in G$ is replaced with a pair of edges $\tilde{e}(i,j) \in \tilde{G}$ and $\tilde{e}(j,i) \in \tilde{G}$. (Note: unlike edges in G, in $\tilde{G}$ edges $\tilde{e}(i,j) \neq \tilde{e}(j,i)$ since i, j are nodes from different vertex sets — either X or Y.)

An example is illustrated in Fig. 4(b). In the remainder of the paper we adopt the convention that the vertices X in the top layer represent the agent set and vertices Y in the bottom layer denote the task set. Since nodes of either layer are copies from the Euclidean graph, this synthesized graph thus also conveys information about the spatial locations (top layer describes the agent locations, and bottom layer describes the task locations). If an agent node is matched to a task node, the agent needs to move from its current location to the newly assigned task location and, when a pair of agent and task nodes are vertically aligned, one can simply imagine that the agent has reached its deployed location and completed the position shift. We term such vertically aligned edges *vertical edges*. When these vertical edges are part of the matching, the agents have no need for relocation.

Because this synthesized graph is a matching graph, we call it the *3D bigraph* and continue to use symbol $\tilde{G}$ to denote it. The matching of a 3D bigraph is initiated with only those well aligned vertical edges whose weights are predefined to satisfy the constraints of matched edges. We return to these details of matched edges later; it is important that we first examine properties of the 3D bigraph itself.

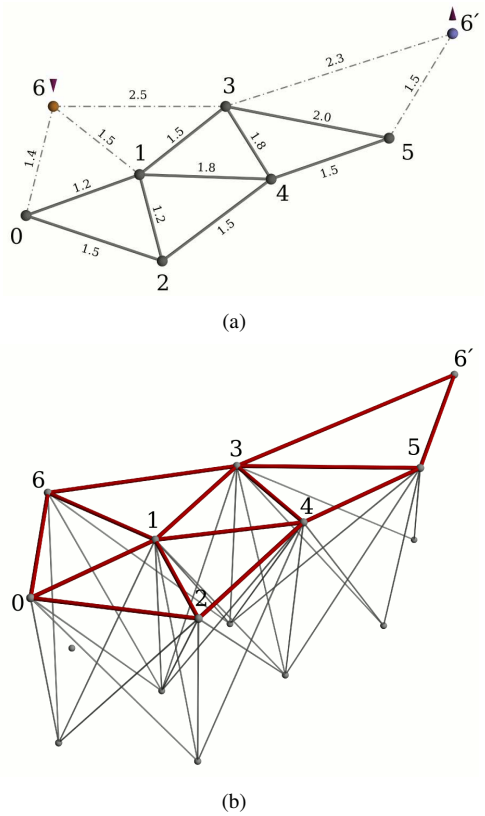


Fig. 4: The mapping from a Euclidean graph to a 3D bigraph. (a) A Euclidean graph of networked robots with only nearest neighbors connected. Vertices 6 and $6'$ are newly inserted agent and task, respectively; (b) The corresponding sparse bigraph visualized in 3D. Bold red edges on top layer do not exist but just show the projection relationship with graph in (a).

B. Redeployment Routing for Inserted Agents and Tasks

When new agents and targets are inserted, the objective is to seek a redeployment policy that ensures every target is quickly serviced by an agent. A redeployment policy for a newly inserted agent-task pair can be represented as a path with two ends connecting the agent and task respectively; this forms a chain in which agents may move, replacing one other, and ultimately reaching the target location.

Suppose that a set of agents A and a set of task locations B are inserted into the current 3D bigraph $\tilde{G} = (X, Y, \tilde{E})$. Assume that they are not collocated with an existing entry in $\tilde{G}$ and, thus, are also unmatched. Then the new bigraph $\tilde{G}' = (X', Y', \tilde{E}')$ becomes:

$$X' = X \cup A \text{ and } Y' = Y \cup B, \tag{12}$$

and

$$\tilde{E}' = \tilde{E} \cup \tilde{E}_A \cup \tilde{E}_B, \tag{13}$$

where $\tilde{E}_A$, $\tilde{E}_B$ are edges that connect vertices sets $A \leftrightarrow Y'$, $B \leftrightarrow X'$ within the traversal (or sensing, or communication) ranges, respectively. Note that the symmetry of the 3D bigraph edges does not apply to the newly inserted edges. This is clearest in an example, see the dashed edges shown in Fig. 5(a): besides a newly inserted agent-task pair connected with dashed lines, all other agent-task pairs are previously deployed and hence matched. The existing matching can be seen as thick vertical lines. This also indicates that the Hungarian Algorithm need only incrementally assign the newly inserted agents, with $|A|$ (for simplicity, suppose $|A| = |B|$) stages of the algorithm. Once all the agents are assigned and new augmenting paths have been found, the routing paths are obtained by *projecting* the augmenting paths in 3D bigraph to either planar layer.

Next, we discuss the example in Fig. 5 in greater detail, showing how the Hungarian bigraph encodes the routing problem with one agent-task insertion. In Fig. 5(a), a new agent-task pair, (a_4, t_4), is added to an existing matching graph with bold edges showing the previous matching. A single incremental Hungarian stage produces the new assignment solution shown in Fig. 5(b). The 3D bigraph in Fig. 5(c) is the same matching graph as in Fig. 5(b), but reveals the spatial relationship between the agents and task locations: the vertices in the top layer are the agents currently assigned or to be assigned, while the vertices in the bottom layer represent the task locations allocated or to be allocated. An augmenting path

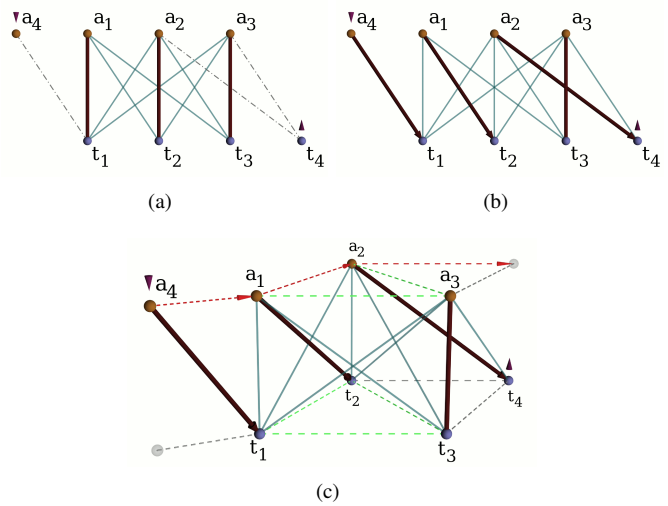


Fig. 5: The bigraph representing a single-pair routing problem (weights omitted). (a) An existing bigraph with perfect matching (bold edges) is supplemented with a new agent and task; (b) A new perfect matching is found after a single stage of the algorithm; (c) When the tasks and agents are actually planar locations then the bigraph picture is more precisely viewed in 3D, the top layer shows the resulting routing solution.

starting from the inserted agent and ending with the inserted task provides the routing path in the projected Euclidean graph. In this example, the augmenting path $a_4 \to t_1 \to a_1 \to t_2 \to a_2 \to t_4$ means that agent a_4 should move to task location t_1, and a_1 move go to t_2, and a_2 to t_4. This is the routing path in the Euclidean graph and is shown with dashed arrows in the top layer in Fig. 5(c).

Theorem 4.1: So long as the graph $G = (V, E)$ for the routing problem is connected, there are always augmenting paths in $\tilde{G} = \Omega(G)$ between pairwise vertices.

Proof: Since a routing path $P : s \rightsquigarrow g$ (connecting a start node s and a goal node g) in G is also a simple graph without any cycles, P can be denoted as $P = (V^p, E^p) \subseteq G = (V, E)$. The 3D bigraph mapped from the path P can be obtained by $\tilde{G}^p = \Omega(P)$, where $\tilde{G}^p$ can be written as $\tilde{G}^p = (X^p, Y^p, \tilde{E}^p) \subseteq \tilde{G} = (X, Y, \tilde{E})$. Since all vertical edges $\tilde{e}^p(i, i) \in \tilde{E}^p$ are matched and all non-vertical edges $\tilde{e}^p(i, j) \in \tilde{E}^p$ $(i \in X^p, j \in Y^p, i \neq j)$ are unmatched, an augmenting path with alternate unmatched and matched edges must be found in $\tilde{G}^p$. The connectedness of graph G means a routing path exists between any two vertices $s, g \in V$, indicating that we can always find an augmenting path for the corresponding vertices pair in $\tilde{G}$. ∎

This indicates that one may simply connect the newly introduced agent-task pair with a given connected graph and, with Corollaries 3.2 and 3.3, the Hungarian Algorithm must yield a path that corresponds to the optimal assignment.

V. Controlling Properties of the Routing Path

The preceding section showed that inserting any agent-task pair to an already deployed system must result in a path that connects them. However, in most cases there will be more than one path through the graph that connects the agent and task pair. Since the different paths are likely to have different properties, selection of one over another should consider the desired behavior and expected performance considerations carefully. We are interested primarily in the following four properties of the resultant routing paths:

1) The total travel cost which is usually proportional to the length of the path or the total energy consumed.
2) The total number of agents to be redeployed, since usually each agent reassignment bears a cost. At the very least, this cost involves interruption and preempted of the previously executed tasks.
3) The average travel distance for individual agents. Redeployment may seek to balance the energy consumption across the robot group.
4) The global finishing time or the time at which all redeployments have been successfully completed, (*i.e.,* the elapsed time until every redeployed agent has reached its newly allocated task location).

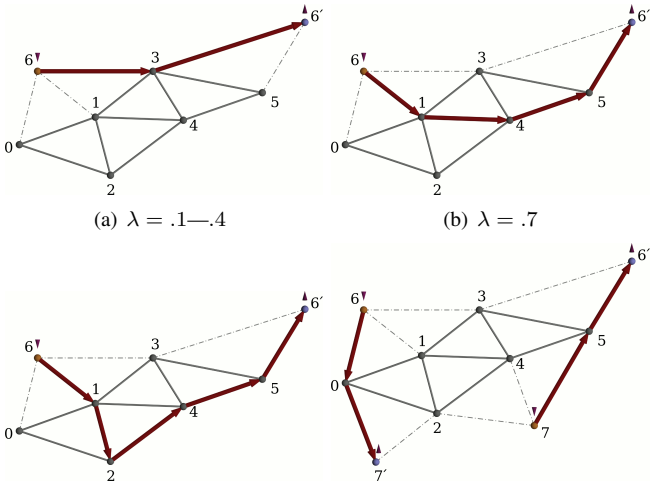
These properties can be understood in terms of distinct metrics: the Euclidean distance captures aspects of travel costs, while the *hopping* (or *geodesic*) distance quantifies the number of edges in the path and, therefore, measures the number of nodes involved (and interrupted) in the deployment.

In this section, we first introduce a method for controlling the generation of paths for the case of insertion of a single agent-task pair. The method involves manipulating the cost matrix itself. We then extend the single agent-task pair case to the general case where multiple agents and multiple tasks are inserted simultaneously, and show how paths with attractive properties result.

A. Parametric Matrices

Although construction of the bigraph has been described in some detail, the edge weights corresponding to the matrix diagonal have not yet been specified. One can interpret their significance as the "state energy" of the agents once they reach their target locations, and the lower the energy, the more stable (harder to disrupt) the agents in the system.

To maintain a vertical pairwise agent-task matching for all agents, we assign each vertical edge of the bigraph a weight of at most the minimal cost of all outgoing edges. This also ensures that the feasibility of a perfect matching in the bigraph is always maintained (see constraints of Program (2)). Formally, if any c_{ij} in a cost matrix $C = (c_{ij})_{n \times n}$ satisfies $0 \leq c_{ii} \leq \min_{\forall j \neq i}\{c_{ij}\}$, all vertical edges $(\tilde{e}(i, i), \forall i)$ form a perfect matching. This is because each cost on the matrix diagonal is the minimum value of row in which it resides, each agent's task on the diagonal is the task which produces the global least cost and also satisfies the mutual exclusion requirement.



(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.

So it is clear that suitable choice of the diagonal values can ensure that the existing deployment has a straightforward optimal matching. But the weights of the vertical edges have a more general interpretation too: they may play a role in determining the degree to which the current assignment may be disrupted, *i.e.*, the smaller the value of c_{ii}, the more likely that edge $\tilde{e}(i,i)$ will remain matched in subsequent Hungarian stages and vice versa. (Note that here the argument is in terms of a dense matrix, but if $\tilde{e}(i,j) \notin \tilde{E}$, it is marked unavailable and cannot be used in Hungarian Algorithm.) Let $diag(C) = diag(c_{11}, c_{22}, \cdots, c_{nn})$ denote the diagonal of matrix C. Then, with scaling parameter λ, we obtain a new diagonally scaled cost matrix:

$$C' = C + (\lambda - 1)diag(C) = \begin{pmatrix} \lambda c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & \lambda c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & \lambda c_{nn} \end{pmatrix}. \quad (14)$$

Only the diagonal is scaled in C', *viz.*, $diag(C') = \lambda diag(C)$. Since we initialize each c_{ii} with $c_{ii} = \min_{\forall j}\{c_{ij}\}$, by tuning $\lambda \in [0,1]$, the diagonal costs vary as $c'_{ii} \in [0, c_{ii}]$. The rationale for using interval $[0,1]$ is provided below.

Next we use the example in Fig. 4(a) to demonstrate how this affects the resultant routing. Vertices 0–5 are previously deployed agents and Vertices 6 and 6′ are the newly introduced agent and task, respectively. Fig. 6 shows different paths as a function of λ, and Table I provides statistics for total path length, number of redeployed/reallocated robots, average path edge length, and the finishing time (which is proportional to the longest edge length). It is interesting to compare the difference between the sparse and dense graphs. The same scenario was considered on the dense graph constructed from Fig. 4(a), where each vertex is connected to all other vertices. The statistics for this dense graph are shown in Table II. Table I and Table II are similar except when $\lambda \in [0, 0.4]$. This is explained by the fact that in the dense graph new longer edges directly connect distant agent-task pairs and, correspondingly, shorter alternative paths can be found. The similarities between the sparse graph and dense graph solutions reflect the fact that edges connecting nearest neighbors, despite producing a sparse graph, permit most paths to be captured. This result is very useful for practical applications because a sparse graph with only the nearby neighbors connected is easy to obtain when the edges represent the sensing or communicating links.

TABLE I: Sparse Graph Paths

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

TABLE II: Dense Graph Paths

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

From Table I and II we can see that, as parameter λ varies from 0 to 1, two trends manifest themselves. Both the total travel distance and the number of redeployed robots monotonically increase, whereas both the average travel distance and finishing time decrease monotonically. These properties are derived from one source, the hopping distance, *i.e.,* the number of edges in the routing path:

- The disruption caused by the introduction of robots and tasks can be measured by the number of redeployed robots, which is the hopping distance minus 1.
- The total travel distance, *viz.*, the sum of edge lengths in the path, is well approximated by the hopping distance in Euclidean graphs when neighbors within the some disk — *e.g.*, communication or sensing range — are connected (*cf.* analysis in Liu *et al.* [2011]);
- When the multi-robot system is uniformly distributed in space, the average travel distance (average edge length of a path) is inversely proportional to the hopping distance.
- Similarly, the finishing time (the longest edge length) is inversely proportional to the hopping distance too.

Extensive experiments are provided with further evidence of these facts, and are presented in the next section on experimental results.

B. Multiple Robot-Task Insertions Producing Multiple Paths

When multiple pairs of agents and tasks are introduced simultaneously (*i.e.,* $|A| > 1, |B| > 1|$), multiple paths

must be produced to ensure each element of B is adequately serviced. One straightforward way to do this is by adding pairs sequentially. This produces a sequence of paths, each generated with the addition of a single pair, one after the other. Because the paths are generated individually, there may be interference between paths and robot motions may have to be constrained to ensure they do not occur simultaneously. Correctness technically requires that navigation for a routing should only commence once the actions needed for execution of the preceding path have been completed. Naturally, this process of serialization of the navigation limits the system's concurrency and hence performance. (Despite the potentially long wait, this is essentially the method used for deployment in Howard *et al.* [2002].) Another drawback of the sequential scheme is that the performance may depend on the ordering of agent-task pair insertions and it is easy to produce scenarios which naïve orderings result in poor routing solutions.

While sequential application of the algorithm is possible, we show how concurrent treatment of batches of agents and tasks is possible and that one can produce paths that are globally efficient. This works by virtue of the matching on the 3D bigraph being optimal. In doing this, we identify the forms of interference that may occur between paths and illustrate that these forms are absent when paths are generated using the proposed approach. This allows deployment to be carried out concurrently by each of the agents and avoids the problem of serializing traversals.

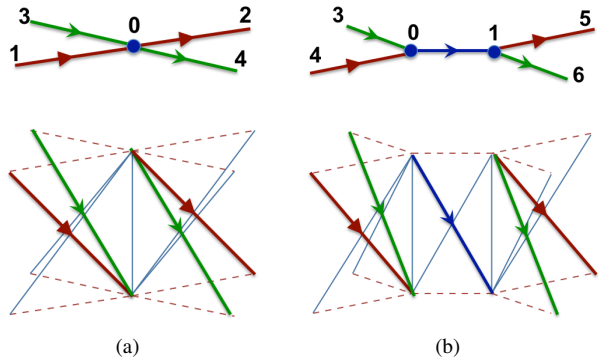


Fig. 7: Neither node nor edge will be shared among multiple paths produced from a matching graph. (a) Assumption of a shared node 0 at the crossing of path $1 \to 0 \to 2$ and path $3 \to 0 \to 4$. The bottom graph is the 3D bigraph showing the violation of mutual exclusion constraint; (b) Assumption of a shared edge $e(0,1)$ belonging to both path $3 \to 0 \to 1 \to 6$ and path $4 \to 0 \to 1 \to 5$.

1) Node Sharing: When two paths both have a vertex in common, we call that node a *shared node*. Shared nodes represent a crossing between two paths. They arise when multiple paths are generated independently, *e.g.*, using a standard shortest path algorithm. A shared node means that the agent at the intersection of different paths is required to simultaneously replace multiple other agents on corresponding paths, which is impossible. An example is illustrated in Fig. 7(a).

Lemma 1: For multiple pairs of agent-task insertion, assume m paths $P_i = (V_i^p, E_i^p) \subseteq G$ ($m > 1$ and $i = 1, 2, \cdots, m$) are generated from the Hungarian Algorithm, then the set $\{P_i\}$ do not (pairwise) share any node, *i.e.*, $V_k^p \cap V_l^p = \varnothing$, where $k, l = 1, 2, \cdots, m$ and $k \neq l$.

Proof: A shared (crossing) node $v_s \in \bigcup_{\forall i} P_i$ has more than one incoming routing edge and more than one outgoing routing edge. More than two routing nodes must be connected to v_s:

$$|\{u \mid e(v_s, u) \in P_i, \forall i = 1, 2, \cdots, m\}| > 2. \qquad (15)$$

This means that in bigraph $\tilde{G} = \Omega(G)$ either the corresponding agent node (in top planar graph) or the task node (in bottom planar graph) or both have more than one matched edges, which contradicts the mutual exclusion constraint violating feasibility of the assignment solution. ■

2) Edge Sharing: When two paths both have an edge in common, we call that edge a *shared edge*. Practically it represents a request for the agents involved to perform multiple traversals simultaneously, which is also impossible. Naturally, a shared edge implies some shared nodes; multiple shared edges can form a *shared path*. An illustration with a simple shared path (one shared edge) is in Fig. 7(b).

Lemma 2: For multiple pairs of agent-task insertion, assume m paths $P_i = (V_i^p, E_i^p) \subseteq G$ ($m > 1$ and $i = 1, 2, \cdots, m$) are generated from the Hungarian Algorithm, then $\{P_i\}$ do not (pairwise) share any edges or path segments, *i.e.*, $E_k^p \cap E_l^p = \varnothing$, where $k, l = 1, 2, \cdots, m$ and $k \neq l$.

Proof: The proof follows reasoning involved in the shared node case. ■

Routing paths P and Q are *disjoint* if neither nodes nor edges are shared between them. Thus, the two preceding Lemmas (1) and (2) lead to the following theorem:

Theorem 5.1: Hungarian Algorithm running on bigraph $\tilde{G} = \Omega(G)$ produces only disjoint paths on graph G.

These disjoint paths connect a set of new agents as sources and a set of new task locations as destinations, finding paths in a *multiple source multiple goal (MSMG)* routing problem.

C. Analysis of MSMG Paths

It is useful to compare the solution quality of independently formed paths with concurrent optimization of MSMG paths. Here we provide both an intuitive understanding by manipulating the paths produced by the Hungarian Algorithm on the 3D bigraph and optimality proofs for conditions of particular relevance to robotic applications.

1) Path Revision: One gains an understanding of how paths are modified by examining the operation of the Hungarian Algorithm at intermediate points in its execution. The algorithm computes a path connecting all agents and tasks by producing a matching through an incremental process which continues until the matching is perfect. By examining the projected routing path before a perfect matching is found, we observe a process of modification and revision of paths as

new, better choices become available, *i.e.*, as subsequent pairs of agents and tasks are processed. Earlier paths are improved by cancelling out expensive segments and shifting to new short-cuts which connect to nearer destinations. This revision mechanism arises from the search for an augmenting path, which modifies matched edges as the algorithm proceeds.

Visualizing the differences in the paths in both Euclidean graph and bigraph forms helps to explain how the process eventually reaches the global optimum. Fig. 8 shows a simple example of path revision: initially the system contains only two agents 0 and 1 with an edge $e(0,1)$ connecting them, agent 3 and task 6 are inserted and a path $3 \to 0 \to 1 \to 6$ is generated. Next, agent 5 and task 4 are added to the search; two concurrent paths must be found. The bottom graph of Fig. 8(a) shows the 3D matching graph in which the matched edge between agents 0 and 1 is flipped during augmentation so it is unmatched in the subsequent path $5 \to 1 \to 0 \to 4$. Thereafter, edge $e(0,1)$ is no longer a segment of any routing path. Two new shorter disjoint paths $3 \to 0 \to 4$ and $5 \to 1 \to 6$ are produced as output, as shown in Fig. 8(b).

2) Optimality Analysis: Here we examine conditions for globally optimizing the overall path length (summed length of all MSMG path segments) and overall hopping distance. Both are important because the former minimizes the total travel distance for a operation, while the latter requires the fewest task interruptions to the system.

Let $\mathbb{P} : A \rightsquigarrow B$ denote a set of MSMG paths which connect elements in A to B and are projected from a matching computed with the Hungarian Algorithm. Further, let $\mathcal{S}(\mathbb{P})$ denote the perfect matching weight sum of the 3D bigraph in which MSMG paths $\mathbb{P}$ reside.

Theorem 5.2: When $w(i,i) = 0$ $(\forall i \in V \setminus A)$, the MSMG routing paths $\mathbb{P}$ have the globally shortest path length.

Proof: When $\tilde{w}(i,i) = 0, \forall i \in V \setminus A$, the sum of weight in the perfect matching is

$$\begin{aligned}\mathcal{S}(\mathbb{P}) &= \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{P}}} \tilde{w}(i,j) + \sum_{\forall v \notin \mathbb{P}} \tilde{w}(v,v) \\ &= \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{P}}} \tilde{w}(i,j) + 0 \\ &= \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{P}}} w(i,j),\end{aligned} \tag{16}$$

which is exactly the total length of all MSMG routing paths. The Hungarian Algorithm always produces a perfecting matching with an assignment solution that is globally optimal, the total path length must therefore be the shortest as well. ■

Theorem 5.3: The MSMG routing paths $\mathbb{P}$ reach the global shortest hopping distance $\mathcal{D}(\mathbb{P})$ when the weights $\tilde{w}(i,i)$ $(\forall i \in V \setminus A)$ are sufficiently small.

Proof: Assume there exists another set of MSMG paths $\mathbb{Q} : A \rightsquigarrow B$ with an overall shorter hopping distance $\mathcal{D}(\mathbb{Q}) < \mathcal{D}(\mathbb{P})$. Let all weights of 3D bigraph vertical edges be equal to a value ξ, *i.e.*, $\tilde{w}(i,i) = \xi, \forall i \in V \setminus A$. Since all nodes not on the paths themselves maintain their matching, the weight sums for the two matching solutions are

$$\mathcal{S}(\mathbb{P}) = \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{P}}} \tilde{w}(i,j) + (|V| - \mathcal{D}(\mathbb{P}))\xi, \tag{17}$$

and

$$\mathcal{S}(\mathbb{Q}) = \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{Q}}} \tilde{w}(i,j) + (|V| - \mathcal{D}(\mathbb{Q}))\xi, \tag{18}$$

respectively. Then we have

$$\begin{aligned}\mathcal{S}(\mathbb{P}) - \mathcal{S}(\mathbb{Q}) = &\sum_{\forall (i,j)|_{e(i,j)\in\mathbb{P}}} \tilde{w}(i,j) - \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{Q}}} \tilde{w}(i,j) \\ &+ (\mathcal{D}(\mathbb{Q}) - \mathcal{D}(\mathbb{P}))\xi.\end{aligned} \tag{19}$$

Now if we let

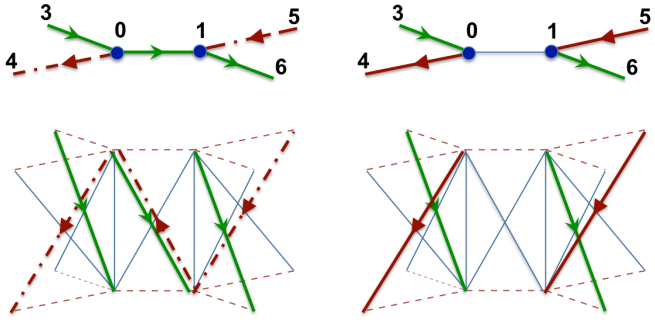
$$\xi = -(|V|\zeta + \epsilon), \tag{20}$$

where ϵ is a small positive value and $\zeta = \max(C)$ is the largest value of the cost matrix. Since any cost $c_{ij} \geq 0$, with (19), we have

$$\begin{aligned}\mathcal{S}(\mathbb{P}) - \mathcal{S}(\mathbb{Q}) &\geq -\sum_{\forall (i,j)|_{e(i,j)\in\mathbb{Q}}} \tilde{w}(i,j) - \mathcal{D}(\mathbb{P})\xi \\ &\geq |V|\zeta + \epsilon - \sum_{\forall (i,j)|_{e(i,j)\in\mathbb{Q}}} \tilde{w}(i,j) \\ &> 0,\end{aligned} \tag{21}$$

which contradicts the optimality of the matching from the Hungarian Algorithm, indicating that when the weights $\tilde{w}(i,i)$ $(\forall i \in V \setminus A)$ are sufficiently small the MSMG paths $\mathbb{P}$ must have the shortest global hopping distance. ■

Until now, the reason for the bounds on the range of λ have not been explained. The proofs of Theorem 5.2 and 5.3



(a) (b)

Fig. 8: An illustration of path revision. The top two figures are 2D graphs, and bottom figures are the corresponding 3D bigraphs.

show that when $\lambda = 0$, the weights $\tilde{w}(i,i) = \lambda c_{ii} = 0$ are small enough to capture the Euclidean shortest path. However, the value of zero might not be sufficiently small to obtain the shortest hopping distance (proof of Theorem 5.3 shows that in general a negative value can be necessary in some cases). However, the Euclidean graph generated by connecting locations within some spatial neighborhood (*e.g.*, capturing sensing or communication limitations, or local traversability) is special. Our previous work [Liu *et al.*, 2011] illustrates that in such graphs, the path minimizing the hopping distance and the path minimizing total length have a bound that describes how much they may differ. The difference shrinks as the number of neighbors for each robot increases. We therefore regard $\tilde{w}(i,i) = \lambda c_{ii} = 0$ as sufficiently small to obtain short hopping distances (the experimental results in Section VI-A also show that when $\lambda \leq 0$, the trends of all different measurements tend to flatten out). Moreover, we do not wish $\lambda < 0$ because in that case the global path length begins to increase. Also, since λ cannot be greater than 1 otherwise values on the matrix diagonal may no longer be the smallest in the row (which would violate the feasibility of the vertical matched edges in the 3D bigraph), this explains why select $\lambda \in [0, 1]$.

D. Concurrent Paths in Narrow Bridges

In investigating paths formed in complex environments, it is important to determine the maximal number of concurrent paths that can be formed. Narrow spaces can pose a challenge because they can impose a limit on the concurrency that is possible; understanding these limits allows one to decide when sequential treatment (*e.g.,* for subsets of A and B) might be called for. This sections quantifies this aspect.

Definition 1: Let $G = (V, E)$ be a connected graph. A subset $C \subseteq V$ is called a *vertex cut* if $G - C$ (the remaining graph after removing all vertices in C and their incident edges) is disconnected. A *minimal vertex cut* is a vertex cut with the least cardinality.

Definition 2: The *local connectivity* $\kappa(u, v)$ (or $\kappa(A, B)$) is the size of the smallest vertex cut separating non-adjacent vertices u and v (or vertices sets $A \subseteq V$ and $B \subseteq V$).

Theorem 5.4: (*Menger's Theorem*) Let $G = (V, E)$ be a graph and $A, B \subseteq V$, then $\kappa(A, B)$ is equal to the maximum number of disjoint A-B-paths (*i.e.*, the paths that connect vertices of A and B) in G.

Proof: Three proofs appear in Diestel [2005]. ■

Theorem 5.4 provides a tight upper bound for the possible disjoint routing paths. However, we wish to know how close to this bound the disjoint paths produced by the Hungarian Algorithm on the corresponding matching graphs are.

Theorem 5.5: For connected graph $G = (V, E)$ with $A, B \subseteq V$ and $\min\{|A|, |B|\} \geq \kappa(A, B)$, the number of disjoint paths generated from the Hungarian Algorithm is equal to $\kappa(A, B)$, *i.e.*, Hungarian Algorithm running on $\tilde{G} = \Omega(G)$ outputs a set of disjoint A-B-paths, so that each path consists of exactly one cut vertex belonging to a minimal set of cut vertices.

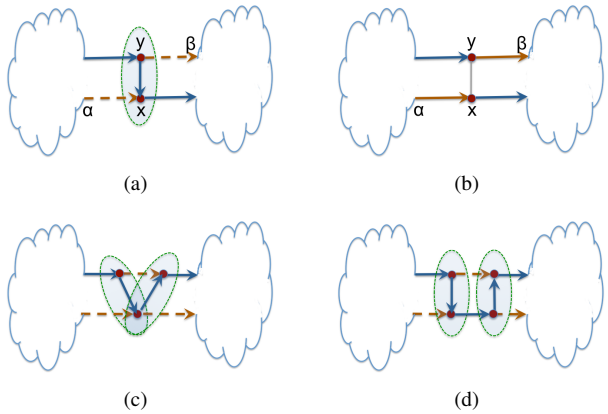
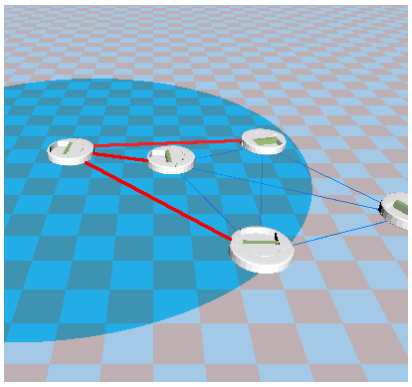


Fig. 9: Conditions for concurrent paths passing through a narrow bridge in a simplified Euclidean graph. (a) Nodes x and y on a path (solid arrowed edges) are from the same minimal vertex cut (circled in an ellipse); (b) Two paths are generated after having found an augmenting path $\alpha \to x \to y \to \beta$; (c)—(d) Vertices of a path that are from intersecting and independent minimal vertex cuts, respectively.

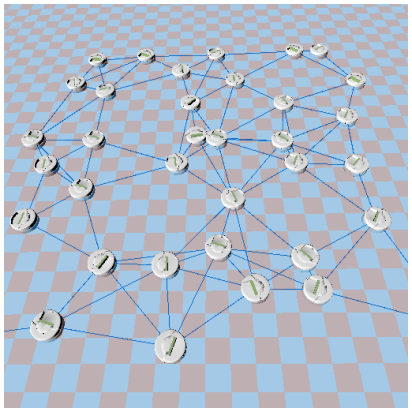
Proof: Assume a maximal set of disjoint paths $\mathbb{P}_{max} = \{P_i\}$, $i = 1, \cdots, m$ is output, and assume a minimal vertex cut of G is C. If the number of paths $|\mathbb{P}_{max}| < |C|$, there must be some path P_l ($l \in [1, m]$) that contains more than one cut vertex from C. If these vertices form a set $V_l' \subseteq V$ with $|V_l'| = K$, then there must be $K - 1$ edges $E_l' \subseteq E$ (which can also be path segments) connecting these vertices. For an arbitrary edge $e(x, y) \in E_l'$ where $x, y \in V_l'$, x, y must be incident with other edges that are not on routing paths (a property following from the mutual exclusion constraint and the definition of a minimal cut), suppose they are $e(\alpha, x)$, $e(y, \beta)$ respectively. (This is illustrated in Fig. 9(a).) Then path $\alpha \to x \to y \to \beta$ forms an augmenting path. Flipping matched and unmatched edges cancels edge $e(x, y)$ and effectively bridges two new paths (this is shown in Fig. 9(b)). Similarly, other edges in E_l' can also be revised and cancelled, and each such revision will add exactly one new path. Other complex conditions involving multiple intersecting or independent sets of minimal vertex cuts (see. Fig. 9(c) and 9(d)) are treated analogously. There must be $\kappa(A, B)$ disjoint paths generated, each of which consists of exactly one vertex from a minimal vertex cut. ■

VI. Experiments and Results

We conducted two complementary forms of evaluation. The first, a simulation study involving large graphs, systematically evaluates the impact of edge degree, the effect of values of λ, and multiple concurrent paths in narrow environments. The second involves validation of the algorithm on physical robots ($n = 6$) and ensures no unreasonable simplifications or assumptions have been introduced whilst developing the algorithm.



(a)



(b)

Fig. 10: (a) Connection with neighbors within a sensing/communication disk; (b) A sparsely connected Euclidean graph for a mobile multi-robot system.

A. Simulation of large-scale networked swarms

Evaluation of the algorithm on inputs representative of large-scale systems was tested by constructing sparse Euclidean graphs. In the graph, vertices represent the already deployed, currently stationary, robots. Edges represent traversable links, each being weighted by the Euclidean distance between the associated agent-task pair. Positions of the vertices were randomly generated, with each connected to its nearest neighbors within a disk to model a sensing or communication range. This allows the sparsity of the graph to be controlled and the edge degree k of vertices to be manipulated. Fig. 10 is an example, showing the communication disk and Euclidean graph.

We first investigated the influence of degree, *i.e.*, the number of nearest neighbors, on the quality of the routing paths. Figures 11(a)—11(d) are examples illustrating the paths in graphs of different bounded degrees of 50, 30, 10, and 5, respectively, with fixed λ (an arbitrary value of 0.7 for all of instances). The newly inserted robot is located in the bottom left corner and the newly inserted task in the upper right corner. The routing paths have few changes for graphs with degrees above 10, but a significant change happens when the degree is under 5. We ran many different experiments with graph sizes ranging from tens of vertices to hundreds of vertices, and found the following trend: whatever the size, graphs with degree of above ~ 10 generate very similar paths, but this is not the case with degree less than ~ 7. This characteristic suggests suitability for practical applications: due to the limitations of sensing and communication, each robot may only be able connect to a small number of neighbors within a certain range. As long as the number of accessible neighbors is greater than ~ 7, the established sparse graph is sufficient to capture most routing paths that can be found in a corresponding graph of much higher density.

As mentioned before, the resulting paths have several particular properties of interest: the overall path length describing the total travelling costs; the total number of agents to be redeployed (how much disruption there is to the previous assignment); the average travel distance for each robot (average edge length in the routing path) and the time used to complete the redeployment (how fast the reassignment can be done). We examined the relationship between these properties and tunable parameter λ. Figures 11(e)— 11(h) show different paths when λ decreases from 1 to 0, causing a gradual straightening of the paths as total lengths become shorter. For each λ, we collected statistics from 30 sets of experiments, with bounded degree $k \approx 10$ (a moderate value, see the discussion above suggesting that this is reflective of many different small values). Fig. 12 includes four plots, which show the total travel distance, the number of robots reallocated, the average distance each robot moves, and the finishing time to complete the routing. We observe that as λ increases, both the total travel distance and the number of reallocated robots increase monotonically, whereas both the average moving distance and the finishing time decrease monotonically. These trends gradually flatten out as λ approaches 0, indicating that 0 is generally enough to capture critical path properties. Moreover, except the total travel distance, the rates of change for other three properties are largest when λ is in the interval between 0 and 0.5. We conclude that if λ is tuned to obtain some desired behavior for these three properties, the tuning stepsize should be reduced within that region.

We also tested the cases when multiple agent-task pairs are introduced simultaneously. Figures 11(i)—11(k) illustrate the insertion of multiple agent-task pairs at the same time. The robots are located in the lower corners and tasks are assigned in upper corners. Fig. 11(i) and 11(j) show two separate paths when only one agent-task pair is inserted, and Fig. 11(k) shows the case when the two pairs are added simultaneously. The two new disjoint paths in Fig. 11(k) have higher quality than the single pair cases because the routing solutions generated from the Hungarian Algorithm represent a globally optimal matching. Fig. 11(l) shows that the algorithm also works in higher dimensions.

To verify instances in which the Euclidean graph contains narrow bridges between the inserted robots and tasks, we examined scenarios involving more complex environments. As shown in Fig. 13, an attenuating wall with a narrow door was added to the simulation environment. The results validate

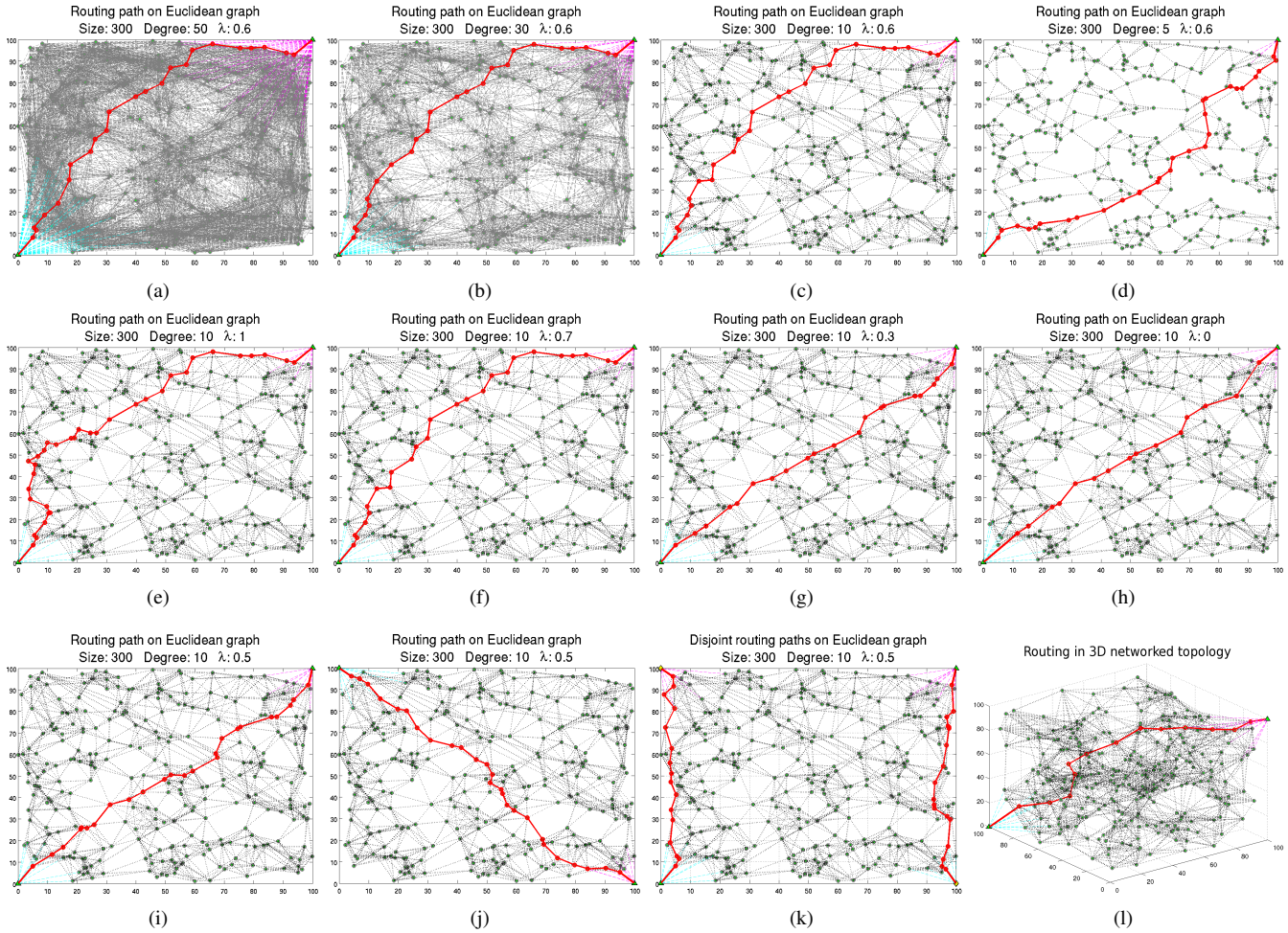


Fig. 11: (a)–(d) Paths produced in large scale multi-robot topologies of varying degree; (e)–(h) Paths in large scale multi-robot topologies of varying λ; (i)–(k) Multiple optimal paths; (l) Path in a 3D scenario.

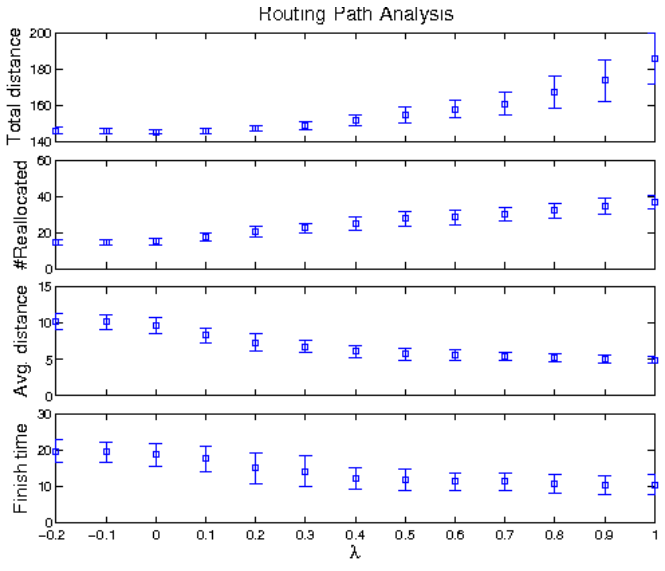
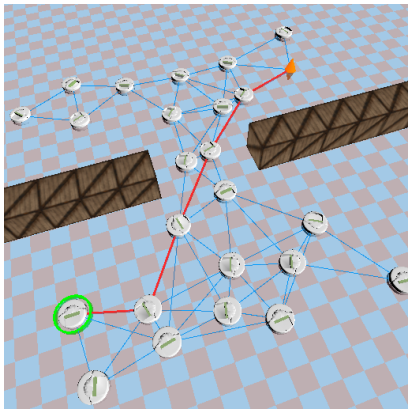


Fig. 12: The computed route's properties as a function of tuning parameter λ.

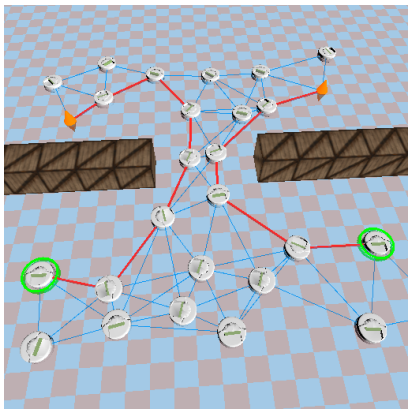
Theorem 5.5: the algorithm can generate paths precisely up to the bound determined by the cardinality of minimal vertex cut (see Fig. 13, as at most two paths across the narrow waist of the graph can be found).

We also compared the shortest paths computed from our method with one of the most well-known SSSP methods, the Dijkstra's algorithm, for the condition when $\lambda = 0$. In all cases, our shortest routing paths are identical to those determined via Dijkstra's classic shortest path algorithm. Fig. 14 shows an example of two identical paths found via Dijkstra's algorithm and the matching method. The advantage of the proposed method over the popular SSSP algorithms lies in its computational complexity in sparse graphs, the scalable and tunable path outputs, as well as the manner in which multiple concurrent paths are intelligently produced.[1]

[1]An earlier comparison of the proposed method with shortest paths from Dijkstra's algorithm showed minor differences in the addition of one node [Liu and Shell, 2012b]; this was traced back to a rounding error and has subsequently been corrected. The fact that the algorithms will produce identical paths even when the inputs are numerically sensitive constitutes quite strong evidence.

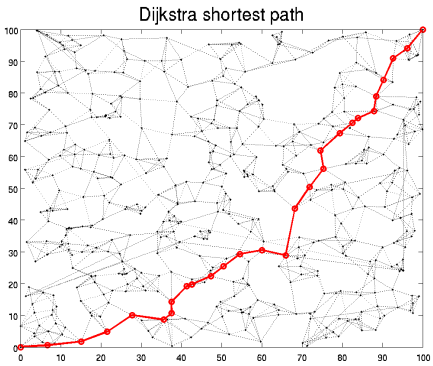


(a)

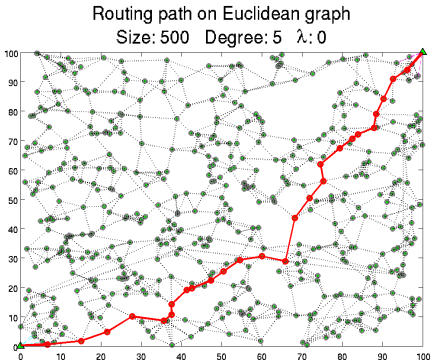


(b)

Fig. 13: A Euclidean graph containing a narrow bridge. (a) A single robot and task pair is inserted (the circled robot and the cone) on either size of the wall; (b) Two pairs of robots and tasks are inserted, with two disjoint paths successfully found. No extra robot task pairs can be added in this example.



(a)



(b)

Fig. 14: (a) Shortest path from Dijkstra's algorithm; (b) Path produced by the proposed method when $\lambda = 0$.

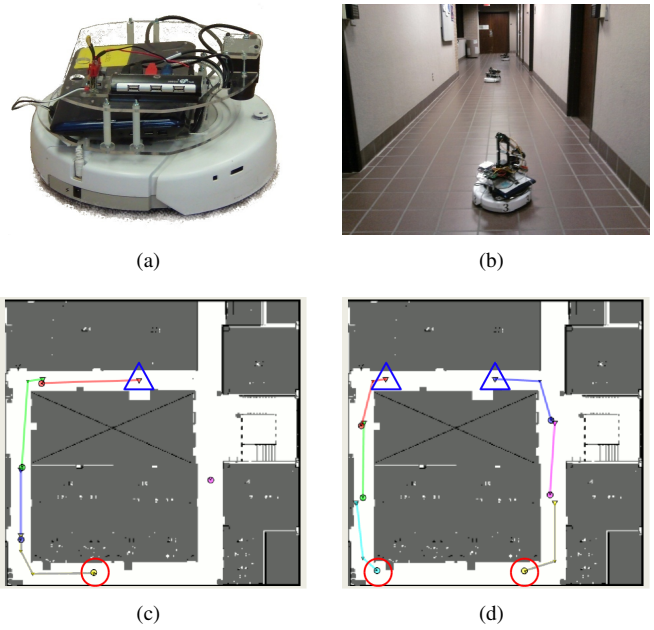
B. Physical robot experiments

We also verified our algorithm with an implementation on physical robots. To construct the sparse bigraph, the algorithm does not need the robots' absolute position information, but instead requires the nearest neighbors' distances for each robot. However, if the robots' positions are known, it makes the bigraph construction easy since the distances between any pair of robots can be directly computed. In the experiment, position information was first obtained via a popular localization algorithm; next, we assigned the localized robots to specific task locations. After the assigned robots have reached their respective destinations, new robot and task pairs were injected into the system.

More specifically, the robots we used are the iRobot create platform, as shown in Fig. 15(a). Each robot is able to localize itself in our research building via a particle filter localization method [Fox, 2001] available as part of the *player/stage* project [Gerkey *et al.*, 2003]. Localization is achieved by using the Hokuyo URG-04LX-UG01 laser sensor, as well as a map drawn to scale with the dimensions of our building. An ASUS EEE netbook on each robot provides the computing platform; it processes all the sensor input, provides onboard communication, calculates cost estimates, logs data, *etc.*

Costs are simply the planned path lengths between robot and task locations. The path length is obtained through a *wavefront* global path planner and a *VFH+* (Vector Field Histogram) local path planner [Ulrich and Borenstein, 1998]. Each robot follows a series of waypoints generated by the path planner to reach the assigned task location (see Fig. 15(c)). The path length is the sum of all the path-segments among its corresponding waypoints. Note that the robots localize themselves quickly with appropriate parameters (*e.g.*, reasonable initial pose estimate, a small laser range variance, *etc.*) that can be tuned in advance.

The robots communicate with each other using UDP. Each robot runs a UDP server and listens to the messages sent by teammates. When a new robot-task pair is inserted into the system, the new robot first queries the latest bigraph from the deployed system, then it connects both itself and the newly allocated task to the 3D bigraph with edges weighted by distances. Since the deployed system already has all robot-task pairs matched, the new robot need only run one stage of the Hungarian Algorithm incrementally to get a new perfect matching; the result is the routing solution. When



(a) (b) (c) (d)

Fig. 15: (a) The physical robot used in experiments; (b) Robots localizing and executing tasks in the corridor of our research building; (c)–(d) Localization results and task allocation information observed by the playernav tool. (c) 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. (d) Two robot-task pairs being inserted.

multiple robots with multiple tasks are introduced, one leading robot is randomly selected from the set of new robots to be responsible for constructing a new bigraph based on the queried system (as well as the connecting newly introduced robots and tasks into the graph). This leading robot runs the Hungarian Algorithm with the necessary stages ($\#stages = \min\{\#robots, \#tasks\}$), to obtain a new allocation solution, which it broadcasts to every team member for an instantaneous routing. The team members then have a commitment to execute those tasks.

We placed task locations uniformly in the left and right corridors as shown in Fig. 15(c), and started the robots at random initial locations. The robots localize themselves and 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 Fig. 15(c). The new robot moves to its task location via a routing path, either left or right traversal in our case. We manipulated both the degree of vertices in the graph and the tuning parameter λ. Note that the number of vertices between the left and right paths is purposely unbalanced so that the left path always has more vertices. Fig. 16 shows the results of a run on the dense graph that connects all robot-robot pairs. The x-axis denotes the degree of unbalancedness in the number of robots in the 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

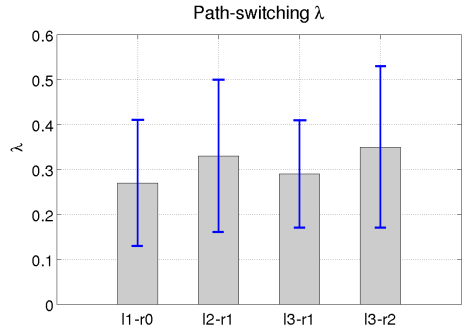


Fig. 16: A plot of the watershed λ that switches between the left and right paths. The horizontal axis value l$\langle\alpha\rangle$-r$\langle\beta\rangle$ denotes the left path has $\langle\alpha\rangle$ robots (vertices) and the right path contains $\langle\beta\rangle$ robots.

the conclusion from the simulation data that path switching occurs more frequently when λ is less than 0.5. Additionally, we tested the situation where each robot is connected to only its nearest neighbors (only the nearest robot in each direction is connected, so that the graph remains connected but with minimal vertex degree). Here, the result differs when λ approaches 0: in a dense graph the routing solution directly allocates the new robot to the new task, whereas in the sparse graph it chooses the path on the right (the shortest routing). This is to be expected since in the dense graph the shortest path is the single edge directly connecting the start and goal.

We also tested cases involving insertion of multiple robot-task pairs. The results show that, despite the sparsity of the graphs or the difference of λ values, the Hungarian Algorithm always correctly generates disjoint paths that connect the newly inserted robots to their nearest available task locations. To model the narrow bridge case, each robot only connected with its nearest neighbors. The minimal vertex cut is then only 2 with the cut vertices located at opposite corridors (two bridges). Fig. 15(d) shows the 2 paths that pass through the minimal vertex cuts. No further robot and task can be inserted in this example.

VII. Discussion

This paper considers the task of computing tunable routing solutions by formulating the problem as an assignment problem. We believe it has the following advantages compared to other algorithms which may also be utilized for the problem:

1) Tunable paths via edge scaling: The algorithm does not modify any of the network connection information (and the underlying Euclidean graph) but achieves different routing solutions by scaling a special "virtual" edge for each vertex via λ. This is easily computed and easily visualized.
2) Concurrent paths for multi-pair insertions: The paths produced when more than one pair of agents and tasks are inserted are guaranteed to be disjoint. Such paths ensure that concurrent movement of all the robots to their redeployment positions is possible. Since environmental constraints may limit the number of concurrent paths, an important property is that the algorithm can make maximum use of capacity of

bridges in the graph, and will actually realize the theoretical upper-bound if needed.

3) Low computational complexity: Each Hungarian stage requires $O(n^2)$ for dense bigraphs [Toroslu and Üçoluk, 2007], and for a sparse bigraph with edge degree of k, the computational complexity is bounded by $O(kn)$. When the problem arises from a network of robots communicating with nearby neighbors, we expect that this latter bound, *i.e.,* linear in the system size, will be applicable.

It is worth emphasizing that the approach described in this paper is applicable more generally than the specific scenario we have used as an example throughout. 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. As a motivating example, consider a swarm of wirelessly networked robots whose positions are not known. Local signal strength readings can be used both to estimate costs, and actually to perform gradient-based traversal from a particular node to another. Different focuses on minimal distance versus time can be important in wireless networking when the traversal is performed by gradient seeking to a particular location: we want to minimize time while also limiting the number of slow, unreliable seeking movements (*i.e.,* limiting the number of reallocations).

The model can also be extended to topological change of large multi-agent systems. For instance, by disconnecting individual agent-task pairs from the topology and reconnecting them with new task assignments, we can "morph" the topology using the approach described. 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.

VIII. Conclusion

This paper proposes a solution to a problem which arises in scenarios where new robots and tasks are added to a network of already deployed units. We seek to move the new robots into maximally useful positions and to adjust the deployed robots only as necessary where doing so reduces costs (*e.g.*, energy, time). The result is a physical routing of robots through the graph encoding navigation constraints.

Our approach applies an variant of the Hungarian Algorithm to solve the optimal assignment problem. The resultant matching forms a routing path, but efficient computation requires we treat the assignment problem incrementally and address sparse graph cases. The paper's most significant contribution is in identifying and interpreting the relationship between the routing graph in Euclidean space and the optimality of the operations on the bigraph represented in three dimensions. The bipartite matching graph that arises from planar path routing is of a specialized form: it is a two layered bigraph with weighted links connected in 3D space. By adjusting the weights of special edges between the two layers we are able to produce a range of solutions. This allows one to balance optimization criteria to minimize total distance travelled, the level of disruption caused by redeploying robots, and the time taken to complete the adjustment by all robots. The algorithm is studied systematically in simulation and also validated with physical robots in our research building.

Funding Acknowledgment

This research was supported by the Department of Computer Science and Engineering at Texas A&M University and the Texas Engineering Experiment Station (TEES).

References

[Agmon *et al.*, 2010] Noa Agmon, Gal A Kaminka, Sarit Kraus, and Meytal Traub. Task Reallocation in Multi-Robot Formations. *J. of Physical Agents 4(2)*, 2010.

[Alonso-Mora *et al.*, 2011] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Roland Siegwart, and Paul Beardsley. Multi-robot system for artistic pattern formation. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, pages 4512–4517, 2011.

[Ayanian *et al.*, 2012] Nora Ayanian, Daniela Rus, and Vijay Kumar. Decentralized Multirobot Control in Partially Known Environments with Dynamic Task Reassignment. In *IFAC Workshop on Distributed Estimation and Control in Networked Systems*, pages 311–316, 2012.

[Bertsekas, 1990] D. P. Bertsekas. The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces*, 1990.

[Bertsekas, 1991] Dimitri P. Bertsekas. An Auction Algorithm for Shortest Paths. *SIAM Journal on Optimization*, 1(4):425–447, 1991.

[Burkard *et al.*, 2009] R.E. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems*. Society for Industrial and Applied Mathematics, New York, NY, 2009.

[Diestel, 2005] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, third edition, 2005.

[Elkaim *et al.*, 2006] G.H. Elkaim, R.J. Kelbley, and Aydan M. Erkmen. A Lightweight Formation Control Methodology for a Swarm of Non-Holonomic Vehicles. In *IEEE Aerospace Conference*, Big Sky, MT, 2006.

[Elmaliach *et al.*, 2009] Yehuda Elmaliach, Noa Agmon, and Gal A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Math and Artificial Intelligence*, pages 3–4, 2009.

[Fox, 2001] Dieter Fox. KLD-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems (NIPS-14)*, pages 713–720, 2001.

[Gerkey and Matarić, 2004] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research*, 23(9):939–954, September 2004.

[Gerkey *et al.*, 2003] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proc. Intl. Conf. on Advanced Robotics*, 2003.

[Giordani *et al.*, 2010] Stefano Giordani, Marin Lujak, and Francesco Martinelli. A distributed algorithm for the multi-robot task allocation problem. In *Proceedings of the 23rd international conference on Industrial engineering and other applications of applied intelligent systems - Volume Part I*, IEA/AIE'10, pages 721–730, Berlin, Heidelberg, 2010. Springer-Verlag.

[Howard *et al.*, 2002] Andrew Howard, Maja J. Matarić, and Gaurav S. Sukhatme. An Incremental Self-Deployment Algorithm for Mobile Sensor Networks. *Autonomous Robots 13(2)*, pages 113–126, 2002.

[Karmani *et al.*, 2007] R.K. 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*, pages 173–182, 2007.

[Kuhn, 1955] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2(1):83–97, 1955.

[Liu and Shell, 2011] Lantao Liu and Dylan Shell. Assessing Optimal Assignment under Uncertainty: An Interval-based Algorithm. *Intnl. Journal of Robotics Research 30(7)*, pages 936–953, 2011.

[Liu and Shell, 2012a] Lantao Liu and Dylan Shell. Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots 33(3):291–307*, 2012.

[Liu and Shell, 2012b] Lantao Liu and Dylan A. Shell. Tunable routing solutions for multi-robot navigation via the assignment problem: A 3D representation of the matching graph. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4800–4805. IEEE, 2012.

[Liu *et al.*, 2011] Lantao Liu, Benjamin Fine, Dylan A. Shell, and Andreas Klappenecker. Approximate characterization of multi-robot swarm shapes in sublinear-time. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2886–2891, Shanghai, China, May 2011.

[Lovász and Plummer, 1986] László Lovász and Michael D. Plummer. *Matching Theory*. North-Holland, 1986.

[Michael *et al.*, 2008] Nathan Michael, Michael M. Zavlanos, Vijay Kumar, and George J. Pappas. Distributed multi-robot task assignment and formation control. In *IEEE Intl. Conf on Robotics and Automation*, pages 128–133, 2008.

[Mills-Tettey *et al.*, 2007] G. Ayorkor Mills-Tettey, Anthony Stentz, and M. Bernardine Dias. The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs. Technical Report CMU-RI-TR-07-27, Carnegie Mellon University, 2007.

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

[Papadimitriou and Steiglitz, 1998] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, 1998.

[Ravichandran *et al.*, 2007] Ramprasad Ravichandran, Geoffrey Gordon, and Seth Copen Goldstein. A Scalable Distributed Algorithm for Shape Transformation in Multi-Robot Systems. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2007.

[Ren and Sorensen, 2008] Wei Ren and Nathan Sorensen. Distributed coordination architecture for multi-robot formation control. *Robotics and Autonomous Systems 56(4)*, pages 324–333, 2008.

[Shen and Salemi, 2002] Wei-Min Shen and B. Salemi. Distributed and dynamic task reallocation in robot organizations. In *IEEE Intl. Conf. on Robotics and Automation*, pages 1019–1024, 2002.

[Topal *et al.*, 2009] Sebahattin Topal, Ismet Erkmen, and Aydan M. Erkmen. Morphing a Mobile Robot Network to Dynamic Task Changes over Time and Space. In *Intl. Conf. on Automation, Robotics and Control Sys.*, pages 192–199, 2009.

[Toroslu and Üçoluk, 2007] Ismail H. Toroslu and Göktürk Üçoluk. Incremental Assignment Problem. *Information Sciences*, 177(6):1523–1529, March 2007.

[Ulrich and Borenstein, 1998] Iwan Ulrich and Johann Borenstein. VFH+: reliable obstacle avoidance for fast mobile robots. In *Proc. Intl. Conf. on Robotics and Automation*, pages 1572–1577, Leuven, Belgium, 1998.

[Wurm *et al.*, 2008] Kai M. Wurm, Cyrill Stachniss, and Wolfram Burgard. Coordinated multi-robot exploration using a segmentation of the environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1160–1165, 2008.