Theory of Algorithms

Report: Modifying the Hungarian Method to Handle Uncertain or Changing Costs

Peter Racioppo

Survey

Given some sets of agents and tasks, the assignment problem is to find the set of agent-task pairs that maximizes or minimizes some objective function. In the linear version of the problem, a fixed cost is assigned to each possible agent-task pair; this situation can always be represented by a complete bipartite graph, since dummy tasks or agents can be assigned. The Hungarian method is a popular algorithm for solving this problem. Though guaranteed to always find a solution in polynomial time, costs in real-world scenarios may be time dependent or uncertain. In these situations, it would be helpful to develop algorithms that determine under what conditions the algorithm will have to be run again. If possible, it would also be helpful to minimize computations by identifying related subsets of the set of all possible agent-task pairs, "cliques," so that computations can be done as locally as possible. Research on these topics has included the exploration of algorithms in the following paradigms:

- Greedy algorithms operated independently by each robot, as in the case of the ALLIANCE architecture, where each robot uses observations of the other robots along with "motivation" heuristics [1].
- Optimization techniques. In [2], the authors define an objective function in order to determine optimal regions in cooperative motion planning problems.
- The distributed auction algorithm employed in [3], run in parallel among groups of agents, as well as an extension of this algorithm to the case where information is only available locally and pricing updates are communicated locally [4].
- Market based algorithms, where each robot bids for a task to produce near-optimal solutions with quick run times; a modification of this idea was presented in [5], allowing for swapping of tasks between adjacent robots to bring final solutions closer to the globally optimal solution.
- Embedding the "discrete space of admissible points in a continuous space" and applying classical techniques for continuous optimization problems, such as gradient descent [6].

Applications of sensitivity analysis to the assignment problem include:

- Treatments of methods to find (1) the range over which the optimal assignment remains optimal, (2) the range of admissible perturbations on costs, and (3) ranges in which changes to the optimal assignments occur at constant rates [7].

Work has also been done on reoptimizing an optimal assignment after a change in costs:

- A "dynamic Hungarian algorithm" was presented in [8] to repair an optimal solution without repeating the full Hungarian algorithm. This algorithm progressively improves assignments using heuristics on swapping a task between pairs of robots.
- Liu and Shell, the authors of the first paper discussed below, propose a method in [9] to "coarsen" the cost matrix, with the aim of partitioning the matrix into subsets in which optimal assignments can be computed independently. This is exactly the same aim discussed in the second of the below papers.

Assessing Optimal Assignment under Uncertainty (L. Liu and D. Shell)

This paper develops algorithms for dealing with uncertain costs in the assignment problem. Given an optimal matching for some set of costs, and assuming only a single cost changes, define the tolerance interval of this cost as the interval in which the cost can take on any value without changing the matching. Identifying tolerance intervals may save computation time in the case of varying costs, since the algorithm does not need to be run again unless one or more cost moves outside its interval. I will begin this discussion with a succinct explanation of the Hungarian method, after which I will discuss the methods for determining the tolerance intervals in the cases of matched and unmatched edges. Finally, I will summarize the paper's discussion of the more general case of interrelated edges.

Algorithm II.1. The Hungarian Method

Definitions (note that this is the maximization version):

- Feasibility condition: $\ell(x)+\ell(y)\geq w(x,y)$
- Feasibility graph: $G_e=(v,E_\ell)$: $E_\ell=\{(x,y):\ell(x)+\ell(y)=w(x,y)\}$.
- Neighbors of *u*: $N(u)=\{v:e(u,v)\in G_e\}$ (the set of all $v\in Y$ connected to some $u\in X$ in the feasibility graph) and $N(S)=\bigcup_{v\in S}N(u)$ (the set of all vertices in *Y* that are connected by some edge in the feasibility graph to some vertex in *S*).

Theorem: If ℓ is feasible and *M* is a perfect matching in E_ℓ, *M* is a max cost matching.

The idea of the algorithm is to grow an alternating tree (the Hungarian tree) in *G*e, which will be max-cost when it is a perfect matching, by the previous theorem.

1.) Begin by generating a labeling ℓ for each vertex and some initial matching *M* in Ge. Initially, we can set $\ell(y)=0$ for all $y\in Y$ and $\ell(x)=\max_{y\in Y}\{w(x,y)\}$ (the max edge cost of the edges touching *x*). This is feasible.
2.) At a given step, the sets $S\subseteq X$ and $T=N_\ell(S)\neq Y$ are the sets of vertices in *X* and *Y* that have edges between them in the Hungarian tree. Initialize *S* = {*u*} for some random *u* in *S* and initialize $T=\varnothing$. Do steps (3) and (4) until *M* is perfect.
3.) If $N(S)=T$, set $\delta=\min_{x\in S,y\notin T}\{\ell(x)+\ell(y)-w(x,y)\}$ and update labels $\ell'(v)\rightarrow\begin{cases}\ell(v)-\delta \text{ if } v\in S\\ \ell(v)+\delta \text{ if } v\in T\\ \ell(v) \text{ otherwise}\end{cases}$

   This forces $N_\ell(S)\neq T$. In particular, if all of the edges coming out of the vertices in *S* have already been added to the Hungarian tree, this operation finds the minimum slack *δ* of these edges and subtracts it from the corresponding vertices, so that the minimum slack edge coming out of *S* will be added to the Hungarian tree. All vertices outside of *S* and *T* are left alone, to prevent the addition of edges that are not incident to the current tree. The slack *δ* is added to the vertices in *T* to prevent the edges in the Hungarian tree from being removed from *G*e.
4.) If there are vertices in *N*(*S*) not in *T*, pick some such vertex $y\in Y-T$ at random. There are now two possible situations:
   (a.) If *y* is matched to some vertex $z\in X-S$, extend the Hungarian tree to $S=S\bigcup\{z\}$, $T=T\bigcup\{y\}$. Two new edges are added to the tree.
   (b.) If *y* is exposed (that is, it is only incident to a single edge in *G*e), $u\rightarrow y$ is an augmenting path. Augment *M* along the tree.

A point of confusion for me was that one would think that we're progressively making progress in improving the matching as the algorithm progresses, but matchings in the Hungarian tree in subsequent steps are disjoint, and dependent on the root vertex *u*. However, I think that, in non-pathological situations, there's some average sense in which the max-cardinality matching is improving as the algorithm runs. For instance, if we were to compare the optimal matching with the assignment at the second to last step plus the edge between the two remaining vertices at that step, it seems that the total costs of the two matchings should probably not be much different, provided that the costs of the edges are fairly evenly distributed.

Correctness

If $N(S)=T$, we can always create a new feasible matching and *M* remains in the new E_ℓ. If $N(S)\neq T$, either the Hungarian tree is grown or *M* in G_e is augmented. The algorithm thus terminates exactly when it reaches a perfect matching in E_ℓ. The theorem implies optimality.

- Definition of the tolerance margin ε_m of a matched edge: given an optimal assignment, a matched edge $e_m(r_\alpha,t_\beta)$ between $r_\alpha \in S$ and $t_\beta \in T$ with cost $w_{m\alpha\beta}$ and tolerance ε_m can take on any value in the interval $[w_{m\alpha\beta}-\varepsilon_m, +\infty)$ without sacrificing optimality.

Lemma 4.1

To find ε_m for an edge $e_m(r_\alpha,t_\beta)$ in the original optimal matching M_0, remove $e_m(r_\alpha,t_\beta)$ (by setting it to a very low number) and run a final iteration of the Hungarian method rooted at r_α using the rest of the tree. All vertices other than r_α and t_β are already in *S* or *T* so the sum of their costs will not change, and the vertex t_β will be added to the Hungarian tree only when the algorithm terminates. Thus, the difference in costs m_0–m' of the old and new assignments M_0 and M' is equal to the change in the vertex label of r_α: $\ell(r_\alpha)-\ell'(r_\alpha)=m_0-m'$.

Theorem 4.2. Matched Edge Interval

Suppose that the best matching excluding e_m has some cost $m'=m_0-\varepsilon_m$. Then the old optimal cost m_0 can't be decreased by more than ε_m or it wouldn't be optimal $\Rightarrow$ $\varepsilon_m=m_0-m'=\ell(r_\alpha)-\ell'(r_\alpha)$.

Algorithm IV.2: All Matched Edge Intervals

Remove a matched edge and run a single iteration of the Hungarian method to determine tolerance for this edge. Repeat for all edges. The time complexity for matched edges is $O(n^3)$, since the Hungarian method is $O(n^2)$ per iteration and there are *n* edges to consider.

Instead, per Dr. Raghvendra's suggestion, run Dijkstra's algorithm to find the shortest path from r_α to t_β, or equivalently, find the shortest edges connecting the two trees formed by the removal of e_m. But it seems that there are still $O(n^2)$ edges to consider, so this wouldn't improve the run time. We would get cost savings if we could avoid running Dijkstra's algorithm *n* times.

Lemma 4.3

The weight of any unmatched edge can be increased to the sum of its two associated labeling values without affecting the optimum assignment, since this operation doesn't change the old feasibility graph and doesn't violate feasibility.

- Define the auxiliary graph G_a for the edge $e_u(r_\alpha,t_\beta)$ (with cost $w_{u\alpha\beta}$) as the graph formed by starting from G_e and removing all edges incident on r_α and/or t_β (including $e_u(r_\alpha,t_\beta)$ itself). This forces $e_u(r_\alpha,t_\beta)$ to be in any matching. Define M' as the optimal matching in $(G_a + e_u(r_\alpha,t_\beta))$ with total cost m' and define $m_a = m' - w_{u\alpha\beta}$.

Theorem 4.4. Unmatched Edge Interval

In order for M_0 to be optimal, $m_0 > m' = m_a + w_{u\alpha\beta} \Rightarrow w_{u\alpha\beta} < m_0 - m_a$. The tolerance margin for $e_u(r_\alpha,t_\beta)$ is thus $(-\infty, m_0-m_a]$. Originally, $w_{u\alpha\beta}=\ell(r_\alpha)+\ell(r_\beta)$, so the tolerance margin for $e_u(r_\alpha,t_\beta)$ is $\varepsilon_u = m_0 - m_a - \left(\ell(r_\alpha)+\ell(r_\beta)\right)$.

A more natural way to put this would be: $\varepsilon_u = \left(m_0 - m'\right) - \left(\ell(r_\alpha)+\ell(r_\beta) - w_{u\alpha\beta}\right)$.

Algorithm IV.2. All Unmatched Edges Intervals

For a given unmatched edge $e_u(r_\alpha,t_\beta)$, run an iteration of the Hungarian method to find the optimal matching M' with total cost m' in the corresponding $(G_\alpha + e_u(r_\alpha,t_\beta))$. The tolerance for this edge is $\varepsilon_u = (m_0 - m') - (\ell(r_\alpha) + \ell(r_\beta) - w_{u\alpha\beta})$. The run time is $O(n^2)$ per iteration for each of the $O(n^2)$ unmatched edges, so altogether it's $O(n^4)$.

Let N be the number of edges in the Hungarian tree incident on r_α and t_β; removing these edges splits the Hungarian tree into N+1 subtrees, one of which is the isolated edge e_u. We can repair the Hungarian tree with Dijkstra's Algorithm by finding the minimum distance between all pairs of subtrees. Dijkstra's Algorithm would take $O(n^2)$ time to join the subtrees back into a single tree and there are $O(n^2)$ unmatched edges, so it's still $O(n^4)$ altogether. Since each r_α and t_β has to be considered n–1 times, it looks like, in non-pathological situations, the majority of the subtrees will usually be mostly the same, but I don't see a general way to take advantage of this.

Algorithm IV.3. Interval Hungarian Algorithm

Combine Alg. IV.1 & Alg. IV.2 to find intervals for all edges.

Theorem 5.1. Uncertainty of a Single Interval

Assuming only one edge cost is allowed to change, perfect matching solutions are identical if and only if this edge cost is within its tolerance interval. The "only if" part of this is true by construction of the edge tolerances and the previous theorems. The converse is true because we've guaranteed, by construction of the edge tolerances, that for any new optimal matching M', $M' \le M_0$ & $M' \ge M_0$. Though the authors didn't quite put it this way, it seems that this is all they're saying.

- Definition: Interrelated edges are edges with related costs. The authors assume interrelated edges are in the same row or column of the cost matrix. Notably, the authors actually seem to make the stronger assumption that interrelated edges are in *either* the same row or column, but not both.
- Define a matched edge e_m with interval $[w_m-\varepsilon_m, +\infty)$ in a set of n unmatched interrelated edges e_{ui}, $i \in \{1,\dots,n-1\}$, with intervals $(-\infty, w_m+\varepsilon_{ui}]$.

Theorem 5.2. Uncertainty of Interrelated Intervals

Assuming all other costs remain fixed, given some $\varepsilon' \le \varepsilon_m$, the cost w_m can be substituted with $w_m - \varepsilon'$ without changing the optimal assignment. The tolerance interval for each of the e_{ui} then becomes $(-\infty, w_{ui}+\varepsilon_{ui}-\varepsilon']$. The first part of this claim was shown previously. The second part follows immediately from Theorem 4.4.

- Define a probability density function $f(x)$ of an edge having a cost x. The authors assume this function is identical for all edges.

Algorithm V.1. Method for Measuring Uncertainties of Interrelated Edges

The authors now address the possible situation that the costs of interrelated edges change simultaneously. Shrinking the interval of a matched edge by $\varepsilon_m-\varepsilon'$ causes the interval of an unmatched edge to shrink by ε', and vice-versa. They suggest the following procedure:

1.) For a given interrelated set, determine the minimal edge tolerance $\varepsilon_{\min}$. Then set $w_m \rightarrow w_m - k\cdot\varepsilon_{\min}$ and $w_{ui} \rightarrow w_{ui}+\varepsilon_{ui} - k\cdot\varepsilon_{\min}$. Here, k is a gain that determines what percentage of $\varepsilon_{\min}$ we're willing to allow a cost to drift. For clarity, I'll call the corresponding interval for each edge a $k\cdot\varepsilon_{\min}$ interval.

2.) Determine the probability P_{Ii} that the ith edge cost has left its $k\cdot\varepsilon_{\min}$ interval by integrating $f(x)$ over this interval. Say that the assignment is reliable if P_{Ii} is less than or equal to some value T for all $i \in \{1,\dots,n\}$.

Experiments

The authors now test the foregoing methods in simulation and physical experiments of robots moving in a maze. Two of the robots have locations known to perfect accuracy at any given time, while the remaining "uncertain robot" uses a localization algorithm to estimate its position using a simulated sensor. Costs are set to be Euclidean distances in the plane. The uncertainties in the uncertain robot's location are input to Algorithm V.1. to determine the probability that each cost is inside its $k\cdot\varepsilon_{\min}$ interval. It's unclear whether the Interval Hungarian Algorithm is run continuously or only a single time at the beginning of the simulation.

Analyzing Uncertainties in Cost: Mitigating Centralization in Multi-Robot Task Allocation (C. Nam and D. Shell)

This paper develops a method of factorizing a set of robots into cliques by perturbing the hyperplanes bounding a hyper-rectangle containing a set of admissible costs. Changes in assignment can then be computed hierarchically, with the goal of saving computation time, by first checking whether a change in costs violates the assignment of a clique and progressively proceeding to larger cliques only if this is the case.

Sec. 3.1. Multi-robot Task Allocation (MRTA) with Changeable Costs

Given *n* robots and *m* tasks, with *ij*th cost c_{ij}, the minimization version of the MRTA problem is to minimize

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \text{ subject to } \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \; ; \; \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \; ; \; 0 \le x_{ij} \le 1 \quad \forall\{i,j\} \; ; \; x_{ij} \in \mathbb{Z}^{+} \quad \forall\{i,j\}.$$

The first of these conditions requires that the *j*th task be done exactly once, and the second requires that the *i*th robot does exactly one task. The third and fourth conditions equate to $x_{ij} \in \{0,1\}$, expressing the fact that a robot can either be assigned to a task or not. We can assume $n = m$, since otherwise dummy tasks/agents can be assigned.

The additional assumption is made here that c_{ij} is restricted to an interval: $\underline{c}_{ij} \le c_{ij} \le \overline{c}_{ij}$. The authors define the matrices $\overline{C}$ and $\underline{C}$ to hold the cost bounds $\underline{c}_{ij}$ and $\overline{c}_{ij}$. Each matrix is composed of *n* vectors, which form *n* "linear boundaries," meaning hyperplanes, in the *n*-dimensional metric space of the costs. Originally, I thought that the matrix *C* was defined as a matrix containing a particular set of costs c_{ij}; I also thought it might be the matrix containing the admissible intervals for a particular assignment (in the sense of the previous paper), but in fact it's the matrix of the cost *intervals* $[\underline{c}_{ij}, \overline{c}_{ij}]$, that is the region enclosed by the *n*-dimensional polytope defined by $\overline{C}$ and $\underline{C}$. It's easy to verify that this polytope is convex. The authors never clearly state whether $\overline{C}$ and $\underline{C}$ are bounds on the possible costs under any conditions (that is bounds on *C* for all possible assignments) or bounds on the costs of a particular *C* (reflecting uncertainty), but the former can be inferred.

Sec 3.2. Background on Sensitivity Analysis

Define X^{*} as an optimal (min-cost) matching, or more specifically, the matrix containing the x_{ij} corresponding to an optimal matching. The authors define vectors of costs for optimal matchings $c \in \mathbb{R}^{s}$ where $s \in n^{2}$. It must be the case that each *c* in $\theta(X^{*})$ has exactly *n* nonzero elements, corresponding to the costs of the assigned robot-task pairs, and $n^{2}-n$ zero elements. Define $\theta(X^{*})$ as the set of all *c* vectors that would result in the optimal matching X^{*}: $\theta(X^{*}) = \{c \in \mathbb{R}^{s} \colon X^{*} \text{ is optimal}\}$.

Define the critical region R_k as the set of all cost vectors *c* that have the same solution for a particular assignment problem. The authors state that R_k is the region bounded by a polyhedral cone, which would mean that its hyperplane boundaries must intersect at a point, which would have to be the origin. A point of confusion for me was that I was originally imagining a polyhedral cone in the *n*-dimensional cost space, but they're referring to the n^{2} dimensional space of the *c* vectors.

The authors note that an X^* may admit degenerate solutions, so $\theta(X^*)$ should be defined as the union of all c vectors whose minimum cost would be given by the assignment X^*. The authors state this as:

$\theta(X^*) = \bigcup_{k \in H} R_k$, where $H = \{k : X^*_{J_k} = B_k^{-1}, X^*_{N_k} = 0\}$ where J_k and N_k, B_k and A_{N_k}, and c_{J_k} and c_{N_k} are, respectively, variables, constraint matrices, and costs of, respectively, basic and non-basic variables. This is apparently the same statement in terms of linear programming formalism.

We're evidently interested in starting with X^* (using e.g. the Hungarian method) and identifying the set of all c that leave this assignment unchanged – hence the need for sensitivity analysis.

Sec. 3.3. Problem Statement

Given a factorization of the robots, the authors now give an overview of how to decide whether to change assignments. It's never clearly stated, but I believe the situation is that we are given an initial cost matrix C_0 , we then compute the corresponding X^*_0 , and we then compute a corresponding set of admissible cost intervals for this assignment. Also, it should be noted that an iteration of the factorization algorithm will possibly alter the assignment, so C_0 and X^*_0 are static with respect to an iteration of the algorithm, but dynamic in general.

Since the costs can vary continuously, but the assignments are discrete, the authors apparently suggest a factorization algorithm in which every possible assignment X^*_q is enumerated, with $q = 1,\ldots,N$, where N is the number of possible assignments. For each of the $N \leq$ (n^2 choose n) assignments, define C_q as the matrix of the admissible cost intervals for which this assignment is optimal. The computation of C_q is not discussed, but you could use the method in the previous paper. The authors also introduce the matrix $C_{\min_q}$, which I gather is the minimum set of costs in C_q, which would mean that it's an extremum on the corresponding polytope. The authors now define the worst case cost difference between the initial assignment X^*_0 and all other possible assignments as $\max\{\bar{C}X^*_0 - C_{\min_q}X^*_q\}$, $q \in [1,\ldots,N]$, the first term being the worst the robots can do by not changing assignments and the second term being the best they can do by changing assignments. It originally seemed a bit odd that we would use $\bar{C}$ and $C_{\min_q}$ rather than $\bar{C}$ and $\underline{C}$, but of course their definition does give the maximum possible difference. It seems that $\max\{\bar{C}X^*_0 - C_{\min_q}X^*_q\}$ = $\bar{C}X^*_0$–$\min\{C_{\min_q}X^*_q\}$ and it should probably be the case that $\min\{C_{\min_q}X^*_q\}$ = $\min\{\underline{C}X^*_q\}$. On the other hand, it doesn't make much sense to me to use $\bar{C}$, since we should have access to (and be interested in) actual costs and not just their upper bounds.

Sec 4.2. Factorizing a Team of Robots

The authors note that enumerating all of the possible assignments is factorial in the number of inputs, and then seem to make a vague statement that it may not take this long due to degeneracy. I'm not sure what they're saying here.

I originally believed that the factorization algorithm was along the lines of the following: starting with an initial cost matrix C_0 (or more generally, a matrix of admissible intervals), which defines a polytope, perturb each of its hyperplane boundaries outward toward an extremum **c**' on the polytope defined by C, whose bounds are given by $\bar{C}$ and $\underline{C}$. After each perturbation, the optimal assignment may change, and if so, a subset of the robots will change assignment. These subsets are the cliques.

I think their algorithm is essentially the above, but defined in $\mathbb{R}^{n^2}$. The authors consider an assignment X^*, which they now call X^* rather than X^*_q. They define l as an arbitrary linear boundary in $\theta(X^*)$, by which they must mean a hyperplane in $\mathbb{R}^{n^2}$. Also, in $\mathbb{R}^{n^2}$, I think C will form a hyper-rectangle. The authors define **c**' as "an extreme point of C outside of the current $\theta(X^*)$" (so this is the same as the definition above, but in $\mathbb{R}^{n^2}$). The algorithm is to perturb l (perpendicular to its hypersurface) toward **c**' until l bounds C (more precisely, I think this equates to l not intersecting C at more than one point). So this is a different picture than the one I described above, where we were perturbing each of the n-dimensional hyperplane boundaries of a polytope toward the extrema of a larger bounding polytope. Instead, I think the algorithm is to perturb an n^2 dimensional hyperplane boundary of $\theta(X^*)$ until it bounds the hyper-rectangle defined by C. Something that isn't said explicitly is that this operation will be repeated for every l in $\theta(X^*)$.

The authors say that l only needs to be maximized, since we're perturbing outward toward the bounding polytope, but it seems that this wouldn't guarantee that the hyperplane sweeps through all of C.

Anyway, the following theorem gives a method to perturb l such that no assignments are missed.

Theorem. How to perturb the *l*'s

"Let l be an arbitrary linear boundary in $\theta(X^*)$. The magnitude of a perturbation ε to perturb an arbitrary point $\mathbf{p}$ on l of $|\mathbf{p}|/n$ will not miss any $\theta(X^*)$." The proof is some simple geometry. The important observation is that although the cost space is continuous, which means that a hyperplane boundary l of $\theta(X^*)$ is continuous in the directions parallel to its hypersurface, perturbing an l perpendicular to its hypersurface requires that at least one of its nonzero elements becomes zero and vice versa. I'm not sure that's it's quite this, but the general idea is that the l's must move in discrete steps on account of the finite number of possible assignments.

After completing the above process, the authors say that the total set of cliques can be found by summing all of the assignments, resulting in a block-diagonal assignment matrix. There's some information that isn't discussed here though, such as whether we can be sure that this summing operation will preserve all of the clique information.

Sec 4.3. Choosing to Persist with the Initial Assignment

Algorithm 1 now summarizes the procedure in Sec. 4.2. and Algorithm 2 summarizes the procedure in Sec. 3.3.

Algorithm 1. Find Theta.

- Start with initial cost matrix C_0 and find X_0^* with the Hungarian method. Find $\theta(X_0^*)$ with some sensitivity analysis function SA, which has not been discussed. For this, we can use something like the Interval Hungarian Algorithm in the previous paper. Set $C = C_0$, $X^* = X_0^*$.
- Input the arbitrary hyperplane boundary l_i in $\theta(X_0^*)$ and the bounds $\overline{C}$ and $\underline{C}$ to the function linprog, which returns $\mathbf{c}_i'$ and obj$_i$, the latter of which determines whether l_i bounds C using a linear programming formula.
- If l_i doesn't bound C, perturb l toward $\mathbf{c}_i'$. After a perturbation, determine the largest subset of C, C_q, that is enclosed by l_i. Using the Hungarian method, determine the optimal assignment X_q^* for C_q. Using SA again, determine $\theta(X_q^*)$.
- Output X_q^* and $\theta(X_q^*)$ for this iteration.
- Set $X^* = X^* \bigcup X_q^*$, for use in the next iteration.
- Once l_i bounds C, repeat for every other l_i in $\theta(X_0^*)$.
- At the end of the algorithm, output the full sets $\{X_0^*,...,X_N^*\}$ and $\{\theta(X_0^*),...,\theta(X_N^*)\}$.

Algorithm 2. Max Loss.

- Input the $\{X_0^*,...,X_N^*\}$ and $\{\theta(X_0^*),...,\theta(X_N^*)\}$ computed by Alg. 1.
- For every $\theta(X_q^*)$, compute C_q.
- Compute the extremum $C_{\min_q}$ in C_q.

- Return $C_{\min_q} X_q^*$. After returning this value for all q, compute the minimum of the set, $\min\{C_{\min_q} X_q^*\}$.
- Compute $c_{\text{worst}} = \max\{\bar{C}X_0^* - C_{\min_q} X_q^*\} = \bar{C}X_0^* - \min\{C_{\min_q} X_q^*\}$. Return c_{worst}.

The following algorithm determines whether a team of robots with assignment X^* violates its $\theta(X^*)$ after some changes to the costs.

Algorithm 3. Incremental Communication.

- Start with initial cost matrix C_0 and find X_0^* with the Hungarian method. Find $\theta(X_0^*)$ with SA.
- Some function TA outputs tolerance intervals for each cost. Details aren't given, but it seems that tolerance intervals are assumed to be independent for each cost. The following is run individually by each robot.
- Check whether $c_{ij} \pm \tau_{ij}$ violates the bounds $\underline{c}_{ij}$ and $\bar{c}_{ij}$. (I think each robot just checks its row of the cost matrix.) Collect all of the violated costs into a set C_{v_i}.
- If there is at least one violation, check the robot against adjacent robots to see if cost changes for the adjacent robots may countervail the violation. I'm not sure if adjacent here means belonging to the same clique or just physically adjacent. An indicator variable $V_i = \{0,1\}$ keeps track of whether some robot/set of robots has a violation.
- After all such adjacencies have been checked, if one more of the $V_i = 1$, global communication is required.

Complexity Analysis

According to the authors, computing $\theta(X^*)$ has $O\left(|k|n^2\right)$ time complexity, where $|k|$ is the number of degenerate solution sets, and this dominates each iteration of Alg. 1. Note that $|k| \le n^2$, so this is no worse than the $O\left(n^4\right)$ run time of the Interval Hungarian Algorithm in the previous paper. The authors say that the time complexity of Alg. 1 is then $O\left(\left(|k|n^2\right)^N\right) = O\left(\left(|k|n^2\right)^{e^{an}}\right)$, $a \in \mathbb{R}$. I'm not sure why it's this rather than $O\left(\left(|k|n^2\right)N\right)$, but either way the algorithm is exponential or worse.

Algorithm 2 includes Algorithm 1 and takes an additional $O\left(n^3\right)$ time to compute C_q in each of the N iterations, so its run time is dominated by Algorithm 1.

Algorithm 3 includes Algorithm 1 and is dominated by it. The authors claim that the remainder of the procedure is $O(n)$, but it seems to me that it should be $O\left(n^2\right)$, since there are n robots making $O(n)$ comparisons of numbers.

The authors mention implementing an "anytime algorithm" in real-world scenarios that would involve only computing a subset of each $\theta(X^*)$ to reduce runtime. Evidently, computations of $\theta(X^*)$ have diminishing returns, so the majority of the set can usually be computed much more quickly than all of it.

Experiments

Simulation experiments are run using scenarios of robot teams moving in two dimensions, with costs given by Euclidean distances to stationary goals. The robots also encounter debris or traffic lights, the significance of which I'm not sure of. The anytime algorithm, the details of which are not discussed, is implemented to show that computing subsets of a $\theta(X^*)$ is sufficient in the tested scenarios. Another experiment is run comparing the performance of the Hungarian method, a 1D interval method probably along the lines of the algorithm in the previous paper, and a sensitivity analysis algorithm, which accounts for simultaneous cost changes, but which isn't discussed. The factorization and worst-case cost difference algorithms are tested in simulation and give useable results. A final experiment is run on the incremental communication algorithm, showing that non-global communication was usually sufficient in the tested scenarios.

References

[1] L. Parker. “ALLIANCE: an architecture for fault tolerant multi-robot cooperation.”

[2] N. Atay and B. Bayazit. “Mixed-Integer Linear Programming Solution to Multi-Robot Task Allocation Problem.”

[3] D. Bertsekas. “The Auction Algorithm for the Transportation Problem.”

[4] M. Zavlanos, L. Spesivtsev, and G. Pappas. “A distributed auction algorithm for the assignment problem.”

[5] S. Trigui, A. Koubaa, O. Cheikhrouhou, H. Youssef, H. Bennaceur, M. Sriti, Y. Javed. “A Distributed Market-based Algorithm for the Multi-robot Assignment Problem.”

[6] R. Brockett, W. Wong. “A Gradient Flow for the Assignment Problem.”

[7] C. Lin, U. Wen, P. Lin. “Advanced sensitivity analysis of the fuzzy assignment problem.”

[8] W. Shen and B. Salemi. “Distributed and dynamic task reallocation in robot organizations.”

[9] L Liu and D Shell. “Large-scale multi-robot task allocation via dynamic partitioning and distribution.”