7-2007

The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs

G. Ayorkor Mills-Tettey
Carnegie Mellon University

Anthony Stentz
Carnegie Mellon University

M. Bernardine Dias
Carnegie Mellon University

The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs

G. Ayorkor Mills-Tettey Anthony Stentz
M. Bernardine Dias

CMU-RI-TR-07-27

July 2007

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

In this paper, we present the dynamic Hungarian algorithm, applicable to optimally solving the assignment problem in situations with changing edge costs or weights. This problem is relevant, for example, in a transportation domain where the unexpected closing of a road translates to changed transportation costs. When such cost changes occur after an initial assignment has been made, the new problem, like the original problem, may be solved from scratch using the well-known Hungarian algorithm. However, the dynamic version of the algorithm which we present solves the new problem more efficiently by repairing the initial solution obtained before the cost changes. We present proofs of the correctness and efficiency of our algorithm and present simulation results illustrating its efficiency.

Contents

1 Introduction

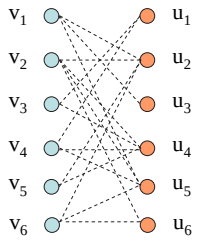
The assignment problem, also known as the maximum weighted bipartite matching problem, is a widely-studied problem applicable to many domains [2]. It can be stated as follows: given a set of workers, a set of jobs, and a set of ratings indicating how well each worker can perform each job, determine the best possible assignment of workers to jobs, such that the total rating is maximized [5]. More generally, given a bipartite graph made up of two partitions V and U, and a set of weighted edges E between the two partitions, the problem requires the selection of a subset of the edges with a maximum sum of weights such that each node $v_i \in V$ or $u_i \in U$ is connected to at most one edge [6]. The problem may also be phrased as a minimization problem by considering, instead of edge weights w_{ij}, a set of non-negative edge costs, $c_{ij} = W - w_{ij}$, where W is at least as large as the maximum of all the edge weights. Unless otherwise stated, this paper considers the minimization formulation of the problem.

The classical solution to the assignment problem is given by the Hungarian or Kuhn-Munkres algorithm, originally proposed by H. W. Kuhn in 1955 [3] and refined by J. Munkres in 1957 [5]. The Hungarian algorithm solves the assignment problem in $O(n^3)$ time, where n is the size of one partition of the bipartite graph. This and other existing algorithms for solving the assignment problem assume the a priori existence of a matrix of edge weights, w_{ij}, or costs, c_{ij}, and the problem is solved with respect to these values. In the many domains, however, things are dynamic and, given an optimal solution to an assignment problem, the problem itself may change before or during the execution of the computed solution: edge weights may change, new nodes may be added to the graph, or nodes may be deleted. For example, consider a problem in which the nodes in the graph represent workers and jobs to be performed by these workers, and the edges represent transportation costs between the worker locations and job locations. In this domain, a given road may be unexpectedly closed, significantly increasing the costs of reaching some jobs by some workers. While the optimal solution for the new problem could be obtained by solving it from scratch using the Hungarian algorithm, a significant computational overhead will result, especially in large problems, if such changes to the edge weights occur frequently. This paper presents a dynamic version of the Hungarian algorithm which, given an initial optimal solution to an assignment problem, efficiently repairs the assignment when some of the edge weights change. A generalization of Toroslu and Üçoluk's incremental assignment algorithm [8], the new algorithm is provably correct and optimal, with a computational complexity of $O(kn^2)$, where k is a measure of the number of cost changes. As shown in the results section, the presented algorithm can solve the modified problem orders of magnitude more efficiently by repairing the old solution than can be done by solving the problem from scratch using the original Hungarian algorithm.

2 Background

2.1 Terminology and Notation

With a few exceptions, this paper employs the terminology and mathematical notation of Papadimitriou and Steiglitz [6]. The Hungarian algorithm assumes the existence of a bipartite graph, $G = \{V, U, E\}$ as illustrated in Figure 1(a), where V and U are the sets of nodes in each partition of the graph, and E is the set of edges. The edge weights may be stored in a matrix as shown in Fig. 1(b). Missing edges are assumed to have zero weight. The minimization form of the problem assumes a matrix of edge costs, $c_{ij} = W - w_{ij}$ where $W \geq max(w_{ij})$. Missing edges may be given a large cost ($\geq W$), as illustrated in Figure 1(c).



(a)

w_{ij}

	u_1	u_2	u_3	u_4	u_5	u_6
v_1	0	8	9	0	6	0
v_2	0	4	0	5	5	8
v_3	7	0	0	9	0	0
v_4	3	0	0	8	8	0
v_5	0	6	0	7	0	0
v_6	0	8	0	0	9	3

(b)

c_{ij}

	u_1	u_2	u_3	u_4	u_5	u_6
v_1	∞	2	1	∞	4	∞
v_2	∞	6	∞	5	5	2
v_3	3	∞	∞	1	∞	∞
v_4	7	∞	∞	2	2	∞
v_5	∞	4	∞	3	∞	∞
v_6	∞	2	∞	∞	1	7

(c)

Figure 1: (a)A bipartite graph, (b) A matrix of edge weights, (c) An alternative representation showing edge costs

Each node in the graph may be *matched* (*assigned*) or *unmatched* (*unassigned*). Unmatched nodes are also called *exposed*. Edges likewise may be matched or unmatched. An edge (v_i, u_j) is matched if v_i is matched to u_j and unmatched otherwise. For clarity, we designate matched edges with solid lines and unmatched edges with dotted lines, as shown in Fig. 2(a). If v_i is matched to u_j, we call u_j the *mate* of v_i, and vice-versa. An *alternating path* is a path through the graph such that each matched edge is followed by an unmatched edge and vice-versa. In Fig. 2(a), $(v_5, u_2, v_1, u_1, v_3)$ is an example of an alternating path. An augmenting path, such as (v_5, u_2, v_1, u_3) in Fig. 2(a), is an alternating path that begins and ends with an exposed node. All alternating paths originating from a given unmatched node form a *Hungarian tree*. Searching for an augmenting path in a graph involves exploring these alternating paths in a breadth-first manner, and the process can be called *growing* a Hungarian tree. Figure 2(b) illustrates the process of growing a Hungarian tree rooted at node v_5, based on the graph in Figure 2(a).

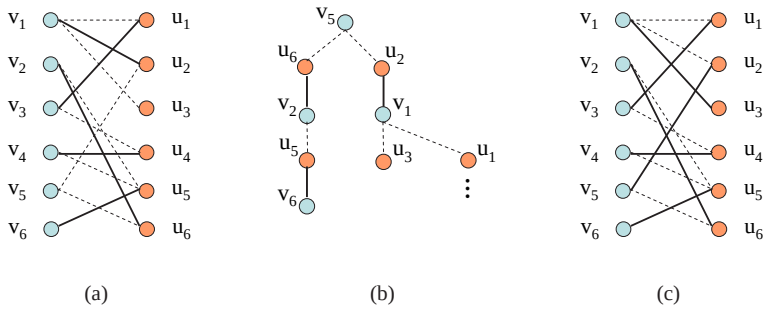


Figure 2: (a) A bipartite graph showing matched and unmatched edges, (b) A Hungarian tree rooted at v_5, (c) The bipartite graph from (a), with matched and unmatched edges flipped along the augmenting path (v_5, u_2, v_1, u_3)

2.2 Hungarian Algorithm

A high-level outline of the Hungarian or Kuhn-Munkres algorithm ([3], [5]) for the assignment problem is shown in Fig. 3. The algorithm assigns dual variables α_i to each node v_i and dual variables β_j to each node v_j. It exploits the fact that the dual of the minimization version of the assignment problem is feasible when $\alpha_i + \beta_j \leq c_{ij}$ [6]. The Hungarian algorithm maintains feasible values for all the α_i and β_j from initialization through termination. An edge in the bipartite graph is called *admissible* when $\alpha_i + \beta_j = c_{ij}$. The subgraph consisting of only the currently admissible edges is called the *equality subgraph*. Starting with an empty matching, the basic strategy employed by the Hungarian algorithm is to repeatedly search for augmenting paths in the equality subgraph. If an augmenting path is found, the current set of matches is augmented by flipping the matched and unmatched edges along this path, as illustrated in Fig. 2(c). Because there is one more unmatched than matched edge, this flipping increases the cardinality of the matching by one, completing a single stage of the algorithm. If an augmenting path is not found, the dual variables are adjusted to bring additional edges into the equality subgraph by making them admissible, and the search continues. n such stages of the algorithm are performed to determine n matches, at which point the algorithm terminates.

If the size of the two partitions of the graph are not equal, a typical strategy is to insert into the relevant partition, dummy nodes with zero-weight edges to all nodes in the opposite partition [6]. As such, the Hungarian algorithm always returns a complete matching, but this matching may include some zero-weight edges, representing "no assignment".

Each stage of the Hungarian algorithm takes $O(n^2)$ arithmetic operations (if implemented with the appropriate data structures [4, 6]), and the computational complexity of the entire algorithm involving n stages is thus $O(n^3)$. The Hungarian algorithm is provably complete and optimal [1, 6].

Hungarian:

Input: A bipartite graph, $\{V, U, E\}$ (where $|V| = |U| = n$) and an $n \times n$ matrix of edge costs C

Output: A complete matching, M

1. Perform initialization:
   (a) Begin with an empty matching, $M_0 = \emptyset$.
   (b) Assign feasible values to the dual variables α_i and β_j as follows:
   $$\forall v_i \in V, \quad \alpha_i = 0 \tag{1}$$
   $$\forall u_i \in U, \quad \beta_j = \min_i(c_{ij}) \tag{2}$$
2. Perform n *stages* of the algorithm, each given by the routine **Stage**.
3. Output the matching after the n^{th} stage: $M = M_n$.

Stage:

1. Designate each exposed (unmatched) node in V as the root of a Hungarian tree.
2. Grow the Hungarian trees rooted at the exposed nodes in the equality subgraph. Designate the indices i of nodes v_i encountered in the Hungarian tree by the set I^*, and the indices j of nodes u_j encountered in the Hungarian tree by the set J^*. If an augmenting path is found, go to step (4). If not, and the Hungarian trees cannot be grown further, proceed to step (3).
3. Modify the dual variables α and β as follows to add new edges to the equality subgraph. Then go to step (2) to continue the search for an augmenting path.
   $$\theta = \frac{1}{2} \min_{i \in I^*, j \notin J^*} (c_{ij} - \alpha_i - \beta_j)$$
   $$\alpha_i \leftarrow \begin{cases} \alpha_i + \theta & i \in I^* \\ \alpha_i - \theta & i \notin I^* \end{cases}$$
   $$\beta_j \leftarrow \begin{cases} \beta_j - \theta & j \in J^* \\ \beta_j + \theta & j \notin J^* \end{cases}$$
4. Augment the current matching by flipping matched and unmatched edges along the selected augmenting path. That is, M_k (the new matching at stage k) is given by $(M_{k-1} - P) \cup (P - M_{k-1})$, where M_{k-1} is the matching from the previous stage and P is the set of edges on the selected augmenting path.

Note: θ can be efficiently computed in $O(n)$ rather than $O(n^2)$ time by maintaining "slack variables" during the search for an augmenting path in step (2). That is, during the search, we maintain for each node u_j, $slack(u_j) = \min_{i \in I^*}(c_{ij} - \alpha_i)$. Then, $\theta = \frac{1}{2} \min_j(slack(u_j))$. As implementation details are described in many references [4, 6], we will not repeat them here. However, the $O(n^3)$ complexity of the Hungarian algorithm is dependent on this implementation efficiency.

Figure 3: The Hungarian Algorithm

2.3 Related Work

2.3.1 The incremental assignment problem.

Recently, Toroslu and Üçoluk [8] examined what they referred to as the incremental assignment problem and presented an elegant algorithm for its solution. In the incremental assignment problem, given a weighted bipartite graph and its maximum weighted matching, we wish to determine the maximum weighted matching of the graph extended with a new pair of vertices, one on each partition, and weighted edges connecting these new vertices to all the vertices on their opposite partitions. While this problem addresses one type of change that may occur in some real-world scenarios, it does not address the important class of changes in edge weights or costs which motivates our current work and is required for a fully dynamic algorithm. Like the regular assignment problem, the incremental assignment problem may be described in terms of a minimization of costs rather than a maximization of weights. The incremental assignment algorithm which Toroslu and Üçoluk presented to solve this problem involves first determining feasible values for the two new dual variables α_{n+1} and β_{n+1} (the other dual variables are still feasible from the solution of the $n \times n$ problem). It then essentially performs a single stage of the Hungarian algorithm to find an augmenting path between the two new vertices v_{n+1} and u_{n+1}, adjusting dual variables to add new admissible edges to the equality subgraph as needed. Flipping the matched and unmatched edges along the discovered augmenting path increases the cardinality of the matching by one, thus resulting in a complete matching. Note that the two new nodes v_{n+1} and u_{n+1} may not necessarily be matched to each other in the final matching. The incremental assignment algorithm of Toroslu and Üçoluk is illustrated in Fig. 4. We have reformulated the algorithm to present it as minimization rather than a maximization algorithm and also to highlight its relationship to the basic Hungarian algorithm as described in this paper. This reformulation does not however change the correctness and complexity of the algorithm which results in a provably optimal solution. Because the algorithm involves executing only one stage of the Hungarian algorithm after performing an $O(n)$ initialization step, the computational complexity of the incremental assignment algorithm is $O(n^2)$.

2.3.2 The dynamic assignment problem.

Spivey and Powell [7] introduce the problem of dynamically assigning resources to tasks, described as the *dynamic assignment problem*. An example application is a transportation domain in which a driver can transport one load at a time – drivers call in over time to a dispatcher asking to be assigned to loads, and customers call in over time asking for loads to be moved by drivers. This is similar to Toroslu and Üçoluk's incremental assignment problem considered on a rolling horizon. However, the modeling approach taken by Spivey and Powell is significantly different. They model three classes of time-lagged information processes namely, (1) the arrival of drivers and loads to the system, (2) information on whether a driver-to-load assignment is feasible and (3) the contribution (weight) of the assignment. The processes are modeled as Markov Decision Processes, capturing the behavior of knowing about the possible arrival of a resource or task before the arrival actually occurs. Their solution strategy is an ap-

Incremental Assignment:

Input:

- An assignment problem comprising a bipartite graph, $\{V, U, E\}$ (where $|V| = |U| = n + 1$) and an $n + 1 \times n + 1$ matrix of edge costs C
- An optimal solution to the $n \times n$ sub-problem of the above assignment problem, comprising a matching M^* of the first n nodes of V to the first n nodes of U, and the final values of the dual variables α_i and β_j for $i \in 1...n$ and $j \in i...n$

Output: An optimal matching, M, for the $n + 1 \times n + 1$ problem.

1. Perform initialization:
   (a) Begin with the given matching, $M_0 = M^*$.
   (b) Assign feasible values to the dual variables α_{n+1} and β_{n+1} as follows:

$$\beta_{n+1} = \min(\min_{1 \leq i \leq n} (c_{i(n+1)} - \alpha_i), c_{(n+1)(n+1)}) \quad (3)$$

$$\alpha_{n+1} = \min_{1 \leq j \leq n+1} (c_{(n+1)j} - \beta_j) \quad (4)$$

2. Perform one iteration of **Stage** from the basic Hungarian algorithm detailed in Fig. 3.
3. Output the resulting matching M.

Figure 4: The Incremental Assignment Algorithm

proximate dynamic programming strategy that requires iteratively solving sequences of assignment problems. By considering the time dimension, their work focuses on determining a "non-myopic" solution that incorporates the expectation of future events. Our focus in this work, however, is to efficiently determine the optimal solution at a given point in time, given all currently information, including information about changed costs.

3 The Assignment Problem with Changing Costs

In this paper, we are concerned with the assignment problem with changing costs, in which we are given an optimal solution to an $n \times n$ assignment problem and a row or column of changed edge weights or costs. The goal is to efficiently find an optimal solution to the new problem with changed costs. As individual costs in the changed row or column may increase, decrease or stay the same, this is a general problem that includes as special cases nodes being removed from the graph (all associated costs increasing to infinity), or nodes being added to the graph (all associated costs decreasing from infinity to a finite value). This problem generalizes the incremental assignment problem addressed by Toroslu and Üçoluk [8] to address the problem of changed edge

Dynamic Hungarian:

Input:

- An assignment problem comprising a bipartite graph, $\{V, U, E\}$ (where $|V| = |U| = n$) and an $n \times n$ matrix of edge costs C
- An optimal solution to the above assignment problem, comprising a complete matching M^*, and the final values of all dual variables α_i and β_j
- Either a row c_{i^*} or column c_{j^*} of changed edge costs.

Output: A new complete matching, M, representing an optimal solution to the problem with the changed edge costs.

1. Perform initialization:
   - If a row i^* of the cost matrix changed:
     (a) Remove the edge $(v_{i^*}, mate(v_{i^*}))$ from the matching M^*.
     (b) Assign $\alpha_{i^*} = \min_j(c_{i^*j} - \beta_j)$
   - Otherwise, if a column j^* of the cost matrix changed:
     (a) Remove the edge $(mate(u_{j^*}), u_{j^*})$ from the matching M^*.
     (b) Assign $\beta_{j^*} = \min_i(c_{ij^*} - \alpha_i)$
2. Perform one iteration of **Stage** from the basic Hungarian algorithm detailed in Fig. 3.
3. Output the resulting matching M.

Figure 5: The Basic Version of the Dynamic Hungarian Algorithm

weights or costs. We propose a dynamic version of the Hungarian algorithm, outlined in Fig. 5, to solve this problem.

In the initialization phase of the dynamic Hungarian algorithm, the affected node is unmatched, decreasing the cardinality of the matching by one: if a row i^* of the cost matrix changed, the affected node is v_{i^*} whereas if column j^* changed, the affected node is u_{j^*}. A new feasible value is then computed for the corresponding dual variable α_i or β_j respectively, according to the equations below:

$$\alpha_{i^*} = \min_j(c_{i^*j} - \beta_j) \quad (5)$$

$$or: \quad \beta_{j^*} = \min_i(c_{ij^*} - \alpha_i) \quad (6)$$

Finally, a single stage of the Hungarian algorithm is executed to increase the cardinality of the matching by one, thus resulting again in a complete matching. Section 5 proves that this new matching is optimal. Because only one stage of the basic Hungarian algorithm is executed after an $O(n)$ initialization phase, the computational

complexity of the proposed algorithm is $O(n^2)$. This bound is proved in Sect. 5 and the practical efficiency of the algorithm is illustrated with numerical results in Sect. 6.

The dynamic Hungarian algorithm can be easily extended to the case where more than one row or column of the cost matrix changes. If k rows and columns change, the k affected nodes are unmatched, and new feasible values are computed for the k affected dual variables. k stages of the basic Hungarian algorithm are then executed, resulting in a computational complexity of $O(kn^2)$. Like in the original algorithm, Hungarian trees are grown from all unmatched nodes simultaneously when searching for an augmenting path in each stage. As such, no decisions need to be made about the order in which to re-match the unmatched nodes.

Finally, if a single entry c_{ij} rather than an entire row or column in the cost matrix changes, optimizations can be made such that nodes are only unmatched if necessary, making the algorithm even more efficient. Although these optimizations do not change the asymptotic complexity of the dynamic Hungarian algorithm, they do cause the algorithm to run faster in real time due to constant time reductions. The optimized version of the algorithm incorporating these features is shown in Fig. 6.

4 Example

Figure 7 illustrates an example of the dynamic Hungarian algorithm in action. We start off with the optimal solution to the assignment problem shown in Figure 7(a). In the graph on the left side of this figure, admissible edges are illustrated with dotted lines while matched edges are shown with solid lines. In the cost matrix on the right, the costs of matched edges are underlined. The optimal solution (with the least-cost matching) shown in Figure 7(a) has a cost of 1+2+3+2+4+1=13.

Suppose now that the costs in the fourth column of the matrix change. The corresponding node, u_4 is unmatched, as illustrated in Figure 7(b). Note that pending the re-computation of the dual variable β_4 corresponding to u_4, we are not sure which edges connected to u_4 are admissible and so no edges are shown connected to u_4 in the graph on the left side of this figure. Note, however, that all previously admissible edges that do not involve u_4 are still admissible. The value of β_4 is then re-computed according to equation 6, as shown in Figure 7(c). This results in a new admissible edge, (v_2, u_4), shown in that figure.

We now enter the main loop of the algorithm. Given that there is only one unmatched node, we perform one stage of the Hungarian algorithm. The search for an augmenting path begins from the only unmatched node in the V partition, v_2. The search fails, requiring modifications of the vertex labels, as shown in Figure 7(d). The search for an augmenting path then continues, yielding the path $(v_4, u_5, v_6, u_2, v_5, u_4)$. The current matching is then augmented along this path by flipping the matched and unmatched edges, as shown in Figure 7(e). The final matching is shown in Figure 7(f). It has an optimal cost of 1+2+3+2+2+2 = 12.

Optimized Dynamic Hungarian:

Input:

- An assignment problem comprising a bipartite graph, $\{V, U, E\}$ (where $|V| = |U| = n$) and an $n \times n$ matrix of edge costs C
- An optimal solution to the above assignment problem, comprising a complete matching M^*, and the final values of all dual variables α_i and β_j
- k cost changes, each of which can be a row c_{i^*}, a column c_{j^*}, or a single entry $c_{i^*j^*}$ of the cost matrix.

Output: A new complete matching, M, representing an optimal solution to the problem with the changed edge costs.

---

1. Perform initialization: For each of the k cost changes
   - If a single value $c_{i^*j^*}$ of the cost matrix changed from c_{old} to c_{new}:
     (a) If $c_{new} > c_{old}$ and v_{i^*} is matched to u_{j^*}, then remove the edge (v_{i^*}, u_{j^*}) from the matching M^*.
     (b) Otherwise, if $c_{new} < c_{old}$ and $\alpha_{i^*} + \beta_{j^*} > c_{new}$
     – Assign $\alpha_{i^*} = \min_j(c_{i^*j} - \beta_j)$
     – If v_{i^*} is not matched to u_{j^*}, remove the edge $(v_{i^*}, mate(v_{i^*}))$ from the matching M^*.

     [**Note:** We may stochastically decide to modify β_{j^*} rather than α_{i^*} in this case. If β_{j^*} is modified, then the edge $(mate(u_{j^*}), u_{j^*})$ should be removed from the matching instead of $(v_{i^*}, mate(v_{i^*}))$.]
   - Otherwise, if a row i^* of the cost matrix changed:
     (a) If v_{i^*} is matched, remove the edge $(v_{i^*}, mate(v_{i^*}))$ from the matching M^*.
     (b) Assign $\alpha_{i^*} = \min_j(c_{i^*j} - \beta_j)$
   - Otherwise, if a column j^* of the cost matrix changed:
     (a) If u_{j^*} is matched, remove the edge $(mate(u_{j^*}), u_{j^*})$ from the matching M^*.
     (b) Assign $\beta_{j^*} = \min_i(c_{ij^*} - \alpha_i)$
2. Let k^* be the number of edges removed from the matching in the initialization phase of the algorithm (note $k^* \leq k$). Perform k^* iterations of **Stage** from the basic Hungarian algorithm detailed in Fig 3.
3. Output the resulting matching M.

Figure 6: The Optimized Version of the Dynamic Hungarian Algorithm

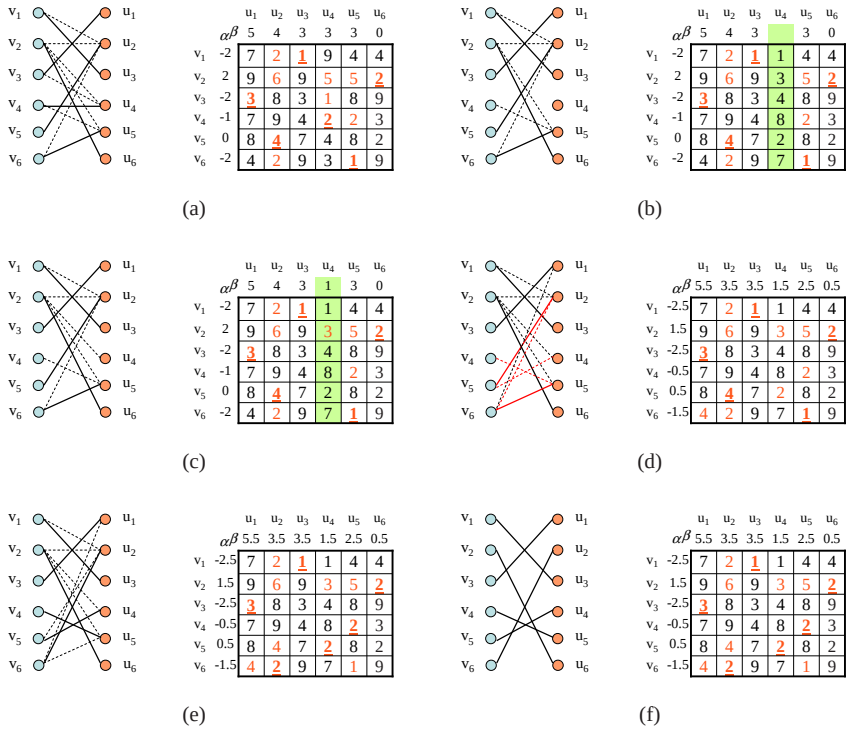


Figure 7: Dynamic Hungarian Algorithm Example

5 Proofs

Let us assume that we are working with a bipartite graph given by $G = \{V, U, E\}$. We will describe an assignment to the dual variables α_i, β_j $\forall i, j$ as a *vertex labeling* of G [1]. A feasible vertex labeling is one for which $\alpha_i + \beta_j \leq c_{ij} \forall i, j$. Given a feasible vertex labeling l, G_l denotes the equality subgraph of G: that is, the subgraph of G comprising all nodes and only the edges for which $\alpha_i + \beta_j = c_{ij}$.

Theorem 1. *Let l be a feasible vertex labeling of G. If G_l contains a perfect matching, M^*, then M^* is an optimal matching of G.*

Proof This theorem corresponds to Theorem 5.5 of Bondy and Murty [1], in which a feasible vertex labeling is defined directly in terms of the weights, $\alpha_i + \beta_j \geq w_{ij}$. In this paper, a feasible vertex labeling is defined in terms of the costs, $\alpha_i + \beta_j \leq c_{ij}$, and the theorem still holds, as shown by the following.

Assume that G_l contains a perfect matching, M*. Since G_l contains all the vertices from G, M^* is also a perfect matching for G. The cost of this matching is:

$$c(M^*) = \sum_{(i,j) \in M^*} c_{ij} = \sum_{v_i \in V} \alpha_i + \sum_{u_j \in U} \beta_j \tag{7}$$

The cost of any perfect matching, M, of G is:

$$c(M) = \sum_{(i,j)\in M} c_{ij} \geq \sum_{v_i \in V} \alpha_i + \sum_{u_j \in U} \beta_j \tag{8}$$

because a feasible vertex labeling requires that $\alpha_i + \beta_j \leq c_{ij}$ for all $v_i \in V, u_j \in U$. It follows that $c(M^*) \leq c(M)$, and M^* is, as such, an optimal matching.

Theorem 2. *The dynamic Hungarian algorithm presented in Fig. 6 results in an optimal assignment for the new problem with modified costs.*

Proof Assume that the previous optimal matching is M^*. When a cost change occurs, the algorithm presented in Fig. 6 first computes a new feasible vertex labeling of G by recomputing α_{i^*} or β_{j^*} as needed. There are three cases:

Case 1: An entire column in the cost matrix changes. Suppose that the vertex corresponding to the changed column is $u_{j^*} \in U$. Thus, the cost c_{ij^*} may have increased, decreased, or remained the same for each $v_i \in V$. As such, the inequalities required for feasibility, $\alpha_i + \beta_{j^*} \leq c_{ij^*}$ may no longer hold for this column with the current vertex labels. For all other vertices $u_j \in U$, no costs change and so the old labels are still feasible: $\alpha_i + \beta_j \leq c_{ij}$. The algorithm adjusts β_{j^*} by setting it equal to $\min_i(c_{ij^*} - \alpha_i)$. This now guarantees that $\alpha_i + \beta_{j^*} \leq c_{ij^*}$, resulting in a feasible vertex labeling.

Case 2: An entire row in the cost matrix changes. Suppose that the vertex corresponding to the changed row is $v_{i^*} \in V$. Thus, the cost c_{i^*j} may have increased, decreased, or remained the same for each $u_j \in U$. As such, the inequalities required for feasibility, $\alpha_{i^*} + \beta_j \leq c_{i^*j}$ may no longer hold for this row with the current vertex labels. For all other vertices $v_i \in V$, no costs change and so the old labels are still feasible: $\alpha_i + \beta_j \leq c_{ij}$. The algorithm adjusts α_{i^*} by setting it equal to $\min_j(c_{i^*j} - \beta_j)$. This now guarantees that $\alpha_{i^*} + \beta_j \leq c_{i^*j}$, resulting in a feasible vertex labeling.

Case 3: A single value $c_{i^*j^*}$ in the cost matrix changes, corresponding to the vertices $v_{i^*} \in V$ and $u_{j^*} \in U$. The cost $c_{i^*j^*}$ may have increased, decreased or remained the same. If $c_{i^*j^*}$ decreased, the inequality $\alpha_{i^*} + \beta_{j^*} \leq c_{i^*j^*}$ required for feasibility may no longer hold for the current vertex labels, while all other inequalities $\alpha_i + \beta_j \leq c_{ij}$ hold for all $v_i \in V, i \neq i^*$ and $u_j \in U, j \neq j^*$. The algorithm fixes this problem by either setting α_{i^*} to $\min_j(c_{i^*j} - \beta_j)$ or setting β_{j^*} to $\min_i(c_{ij^*} - \alpha_i)$, resulting again in a feasible vertex labeling.

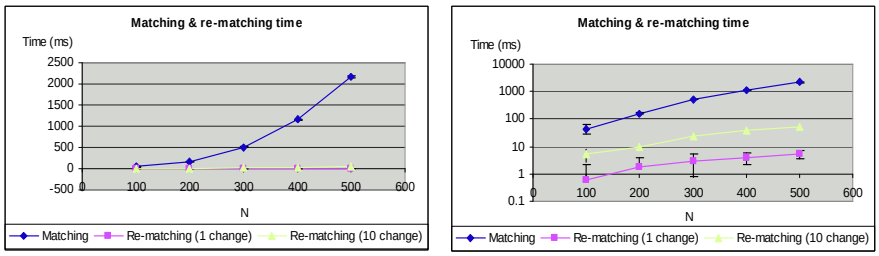
Each cost change results in at most one change in the value of a dual variable, as described above, corresponding to a node $x, x \in V \cup U$. Assume that the node x was matched to node y (if $x \in V$, then $y \in U$, and if $x \in U$, then $y \in V$), then the algorithm removes the pair (x, y) from the matching. If there are k cost changes, there are at most k unmatched pairs. The algorithm then performs a stage of the Hungarian algorithm for each unmatched pair, increasing the cardinality of the matching by one each time, and maintaining feasible vertex labels for all nodes in the graph. At the termination of the algorithm, we have a perfect matching within the equality subgraph and as such, by Theorem 1, we have an optimal matching.

Theorem 3. *The asymptotic complexity of the dynamic Hungarian algorithm presented in Fig. 6 is $O(kn^2)$, where k is the number of cost modifications (columns, rows, or cells) since the last solution.*

Proof The initialization phase of the dynamic Hungarian algorithm recomputes the values of at most k dual variables. Each re-computation is an $O(n)$ computation, resulting in a complexity of $O(kn)$ for the initialization phase. The next phase executes k stages of the Hungarian algorithm, and each stage of the Hungarian algorithm requires $O(n^2)$ arithmetic operations [6]. As such, the asymptotic complexity of the dynamic Hungarian algorithm is $O(kn^2)$.

6 Numerical Results

According to Theorem 3, the Dynamic Hungarian Algorithm has an asymptotic complexity of $O(kn^2)$, compared to the $O(n^3)$ complexity of the basic Hungarian algorithm. This suggests that when few rows or columns in the cost matrix change, it is much more efficient to use this algorithm to repair the matching than it is to solve the problem from scratch using the regular Hungarian algorithm. This is illustrated by the numerical results below, which are provided to give the reader a sense for the "constants" in the computational complexity for an example problem with random costs. Figure 8 compares the amount of time in milliseconds required to compute a new matching using the dynamic algorithm ("re-matching") to the amount of time required to solve the problem from scratch with the regular Hungarian algorithm ("matching") for various problem sizes. These experiments were run on a 2.13 GHz Pentium M workstation. The values shown are averaged over 20 problems with random costs, each cost ranging between 1 and 100. Re-matching times are plotted for a scenario in which costs changed in only 1 column (k=1) and for a scenario in which costs changed in ten columns (k=10) before re-matching. Figure 8(a) plots the results on a linear scale, emphasizing the significant gains that are obtainable by using the dynamic algorithm, particularly as the problem size increases. Figure 8(b) plots the same results on a log-linear scale, enabling a distinction between the re-matching time for 1 changed column versus 10 changed columns. Similar re-matching times are obtained when cost changes occur in rows rather than columns, and the re-matching times are even lower when single entries of the cost matrix change, rather than entire rows or columns.



(a) (b)

Figure 8: A comparison between time required for dynamic re-matching versus matching from scratch, plotted (a) on a linear scale and (b) on a logarithmic scale

7 Conclusions

In this paper, we presented the dynamic Hungarian algorithm for the assignment problem with changing costs. The goal of the algorithm is to efficiently repair an optimal assignment when changes in the edge costs occur, as can happen in many real-world scenarios. The algorithm results in a provably optimal solution and has a computational complexity of $O(kn^2)$ where n is the size of one partition of the bipartite graph, and k is the number of changed rows or columns in the cost matrix. This complexity compares favorably with the $O(n^3)$ operations that would be required for solving the problem from scratch, and we presented simulation results illustrating the practical efficiency of the algorithm on various problem sizes. The dynamic Hungarian algorithm is useful in any domain that requires the repeated solution of the assignment problem when costs may change dynamically. In future work, we will apply this algorithm to various transportation-related problems.

Acknowledgments

This work was sponsored by the Jet Propulsion Laboratory, under contract "Reliable and Efficient Long-Range Autonomous Rover Navigation" (contract number 1263676, task order number NM0710764). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA or the U.S. Government.

References

[1] BONDY, J. A., AND MURTY, U. S. R. *Graph Theory with Applications*. Elsevier Science Publishing Co., Inc, 1976.

[2] BURKARD, R. E., AND ÇELA, E. *Handbook of Combinatorial Optimization, Supplement Volume A*. Kluwer Academic Publishers, 1999, ch. Linear assignment problems and extensions, pp. 75–149.

[3] KUHN, H. W. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2* (1955), 83–97.

[4] LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids*. Holt, Reinehart and Winston, New York, 1976.

[5] MUNKRES, J. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics 5*, 1 (March 1957), 32–38.

[6] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[7] SPIVEY, M. Z., AND POWELL, W. B. The dynamic assignment problem. *Transportation Science 38*, 4 (November 2004), 399–419.

[8] TOROSLU, I. H., AND ÜÇOLUK, G. Incremental assignment problem. *Information Sciences 177*, 6 (March 2007), 1523–1529.