A Greedy Approxmation Algorithm for the Linear Assignment Problem

Garrett Lewellen

March 20, 2017

1 Introduction

The Linear Assignment Problem (LAP) is concerned with uniquely matching an equal number of workers to tasks, n, such that the overall cost of the pairings is minimized. A polynomial time algorithm was developed in the late fifties by [6], and further refined by [9], called the Hungarian method. Named so after the work of Hungarian mathematicans König and Egerváry whose theorems in the 1930s form the basis for the method. While the Hungarian Method can solve LAP instances in $\mathcal{O}\left(n^3\right)$ time, we wish to find faster algorithms even if it means sacraficing optimality in the process. Here we examine a greedy α-approximation algorithm with $\mathcal{O}\left(n^2 \log n\right)$ runtime in terms of its approximation factor and compare it empircally to the Hungarian method.

2 Linear Assignment Problem

$$\begin{aligned} C_n = \min & \sum_{i=1}^{n}\sum_{j=1}^{n} M_{i,j}x_{i,j} \\ s.t. & \sum_{i=1}^{n} x_{i,j} = 1, \quad j = 1,\dots,n \\ & \sum_{j=1}^{n} x_{i,j} = 1, \quad i = 1,\dots,n \end{aligned} \tag{1}$$

The above linear program has cost, $M \in \mathbb{Z}_{+}^{n\times n}$, and assignment, $x \in \{0,1\}^{n\times n}$, matrices that specify the terms of the LAP. This is equivalent to finding a perfect matching in a weighted bipartite graph. A minimal cost may have several possible assignments, but we are only interested in finding just one. It is assumed that no one worker can do all jobs more efficiently by themself than the distributing work across all workers. Likewise, if the costs are thought of durations, then the minimum cost is the minimum sequential rather than parallel time taken to complete the tasks.

From a practical point of view, we may relax the integral constraint on M and allow all positive real-valued costs. For instances where there are more jobs than workers, and vice versa, dummy entries valued greater than the existing maximum may be added. Minimizing the cost is the default objective, but the maximum cost can be found by finding the optimal assignment for $M'_{i,j} = M_{max} - M_{i,j}$, then finding the cost relative to M.

3 Algorithms

Brute Force Rather than using the mathematical programming or graph theoretic representation of the problem, we can instead view the problem as finding the assignment that minimizes the cost

out of all possible assignments:

$$\pi^* = \arg\min_{\pi \in \Pi_n} \sum_{i=1}^{n} M_{i,\pi_i} \tag{2}$$

There are $n!$ such assignments which can be produced using an iterative version of Heap's algorithm [5] in $\mathcal{O}(n!)$ time assuming one does differential scoring (opposed to calculating the score for each permutation which would result in an $\mathcal{O}\left(n^2(n-1)!\right)$ algorithm.)

Random The random algorithm selects a permutation $\pi \in \Pi_n$ uniformly from the set of all possible assignment permutations in $\mathcal{O}(n)$ time using the Fisher-Yates shuffle [4]. This obviously does not produce an optimal or near-optimal solution, but serves as a strawman to compare other results.

Greedy The greedy heuristic continues to cover the row and column of the smallest uncovered entry in the cost matrix until all entries are covered. The resulting set of entries then consitutes the assignment of workers to jobs. An inefficient $\mathcal{O}\left(n^3\right)$ algorithm can be used to find the smallest entry every iteration, or a more efficient result of $\mathcal{O}\left(n^2 \log n\right)$ can be obtained through the use of a sorted, array indexed hybrid mesh and queue. Let `QNode` represent a tuple consisting of row, column, and value; the previous entry in the matrix $\leq$ this value, and the next entry in this matrix $\geq$ this value; and the `QNode`s (left, above, right, below) that are adjacent to this node.

Algorithm 1 A greedy algorithm for the LAP.

```
procedure GREEDY(M)                                   ▷ n × n cost matrix
    A[i] ← ⊥ for i = 0 ... n − 1                      ▷ Assignment A[job] = worker
    Q[i] ← QNode for i = 0 ... n² − 1
    LINKMESH(Q)                                       ▷ Adjacent node left, above, right, below properties
    SORT(Q)                                           ▷ Sort in ascending order by node value
    LINKQUEUE(Q)                                      ▷ Adjacent node previous and next properties
    Q_min ← Q[0]
    while Q_min ≠ nil do
        A[Q_min → row] ← Q_min → col
        Q_min ← DELETENODE(Q, Q_min)                  ▷ Deletes row and col of Q_min
    end while
    return A
end procedure
```

Allocating and linking for assignment is $\mathcal{O}(n)$; mesh $\mathcal{O}\left(n^2\right)$; queue $\mathcal{O}\left(2n^2 \log n + n^2\right)$. Therefore, initialization requires $\mathcal{O}\left(n^2 \log n\right)$ time. The body of the loop requires a constant time assignment of worker to job, and $\mathcal{O}(2k-1)$ time to remove the row and column from a $k \times k$ matrix using a modified depth first search. Thus, the loop itself accounts for $\mathcal{O}\left(n^2\right)$ time. The resulting time complexity is therefore $\mathcal{O}\left(n^2 \log n\right)$ $\square$.

$$\begin{pmatrix} 62 & 31 & 79 & \boxed{6} & 21 & 37 \\ 45 & 27 & 23 & 66 & \boxed{9} & 17 \\ 83 & 59 & 25 & 38 & 63 & \boxed{25} \\ \boxed{1} & 37 & 53 & 100 & 80 & 51 \\ 69 & \boxed{72} & 74 & 32 & 82 & 31 \\ 34 & 95 & \boxed{61} & 64 & 100 & 82 \end{pmatrix} \quad \begin{pmatrix} 62 & 31 & 79 & \boxed{6} & 21 & 37 \\ 45 & 27 & 23 & 66 & \boxed{9} & 17 \\ 83 & 59 & \boxed{25} & 38 & 63 & 25 \\ \boxed{1} & 37 & 53 & 100 & 80 & 51 \\ 69 & 72 & 74 & 32 & 82 & \boxed{31} \\ 34 & \boxed{95} & 61 & 64 & 100 & 82 \end{pmatrix}$$

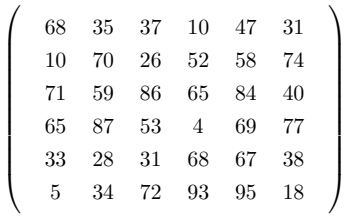
Breaking ties for the minimum uncovered value can result in different costs. This drawback is shown in the above example were choosing 25 at $(3, 6)$ yields a minimum cost of 174, where as the one at $(3, 3)$ gives a minimum cost of 167. The next progression in the design of the greedy algorithm would be to try all minimum positions and keep the top k performing paths.

Hungarian The general idea behind the Kuhn-Munkres algorithm is that if we are given an initial assignment, we can make further assignments and potentially retask workers until all workers have been tasked with a job. The high-level sketch of the algorithm starts with an initial assignment. While we have jobs that are unassigned, we look for qualified workers, ie, the zero entries. If a worker is already assigned to a job, but is also qualified for another, then we prime the alternative and continue to the next qualified worker, but if that is the only job the worker is qualified for, then we'd like to reassign any other worker already tasked to that job. This leads to a natural ripple effect represented by an alternating path of starred and primed entries. In Munkres' paper [9] "starred" zero's represent assignments of workers to jobs, and "primed" zero's are alternative assignments. By flipping the bits of the path, we retask workers to their alternative tasks while ensuring the assignment continues to be minimal by construction. After assigning as many workers as we have to, we then deduct the lowest cost to create a new qualified worker. Thus, every iteration we are guaranteed to make positive progress towards our goal of finding an optimal assignment. This scheme results in the worst case $\mathcal{O}\left(n^3\right)$ time to complete.

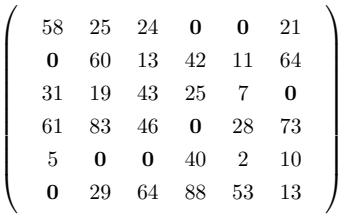
Algorithm 2 The Hungarian method for the LAP.

procedure HUNGARIANMETHOD(M) ▷ $n \times n$ cost matrix
  $M_{i,j} \leftarrow M_{i,j} - \min_j M_{i,j}$ for $i = 0 \dots n-1$
  $M_{i,j} \leftarrow M_{i,j} - \min_i M_{i,j}$ for $j = 0 \dots n-1$
  Star the first uncovered zero in row i, cover the corresponding column j for $i = 0 \dots n-1$
  while All columns not covered **do**
    while Uncovered zeros **do**
      Prime the current uncovered zero
      if There's a starred zero in this row **then**
        Uncover the starred zero's column and cover the row
      else
        Find an alternating augmented path from the primed zero
        Unstar the starred zeros on the path and star the primed zeros on the path
        Remove all the prime markings and cover all stared zeros
        break
      end if
    end while
    if Found path **then**
      continue
    end if
    $M^* = \min M_{i,j}$ over all uncovered i, j
    $M_{i,j} = M_{i,j} - M^*$ for all uncovered columns j
    $M_{i,j} = M_{i,j} + M^*$ for all covered rows i
  end while
  return Starred zeros ▷ These are all the assignments
end procedure

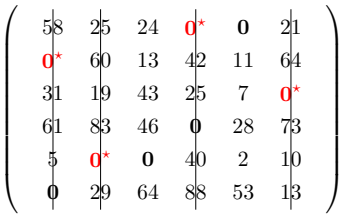
To further illustrate the algorithm, consider the following example where starred entries are denoted by red, and primed entries by green:



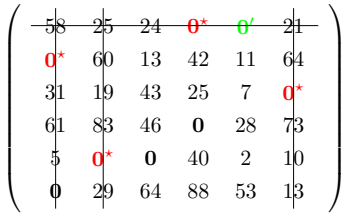
Input cost matrix



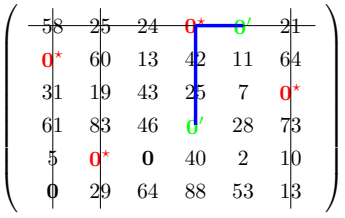
Deduct row, then resulting column minimums



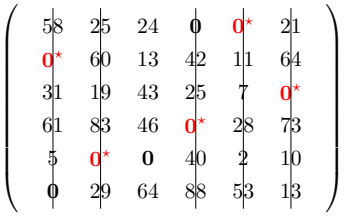
Cover columns of starred entries



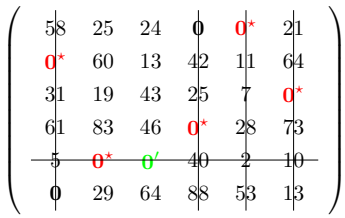
Starred zero at $(1, 4)$, prime $(1, 5)$



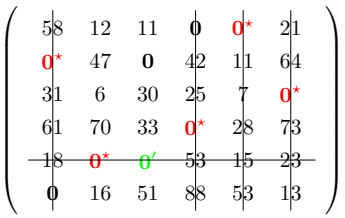
No starred zero in row 4, prime $(4, 4)$ and find augmented path



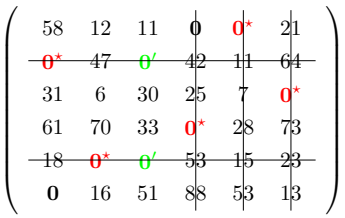
Unstar starred entries and star prime entries along path



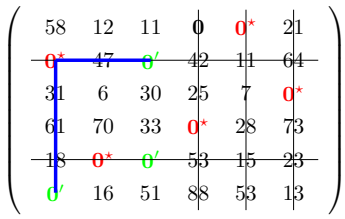
Starred zero at $(5, 1)$, prime $(5, 3)$



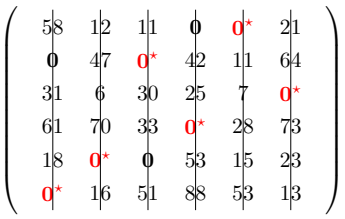
No uncovered zeros, deduct uncovered minimum and it to double covered entries



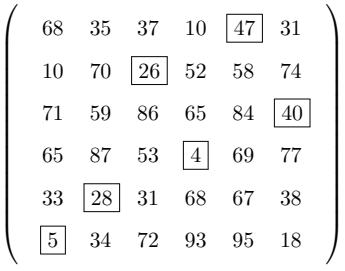
Starred zero at $(2, 1)$, prime $(2, 3)$



No starred zero in row 6, prime $(6, 1)$ and find augmented path



Unstar starred entries and star prime entries along path



Output cost of 150

4 Analysis

The prevailing convention in the literature is to look at the approximation factor, α, to determine how close the results of an approximation algorithm are to optimal [10]. Here this ratio is the expected minimum cost assignment of the algorithm under test to the same quantity given by the expected minimum assignment cost. Let $M_{i,j} \sim \mathrm{Exp}\,(1)$ be an $n \times n$ a standard exponential random

cost matrix. We resort to the exponential distribution for its ease of analyis and prominence in related literature. Cf. the works of [7], [8] for analysis based on $M_{i,j} \sim \mathcal{U}(0,1)$.

Exponential Distribution Properties Let $X \sim \text{Exp}(\lambda)$ have cumulative distribution function $F_X(x) = 1 - \exp(-\lambda x)$ and expectation $\mathbb{E}(X) = \lambda^{-}$. The distribution demonstrates the memoryless property for expectations $\mathbb{E}(X|X > a) = \mathbb{E}(X) + a$. Define the order statistic $X_{1:n} = \min\{X_1, \ldots, X_n\}$ to be the minimum of n draws from $\text{Exp}(\lambda)$. $X_{1:n} \sim \text{Exp}(n\lambda)$ [2] with expectation $\mathbb{E}(X_{1:n}) = (n\lambda)^{-}$. If $Y_n = \sum_{i=1}^{n} X_i$ then $Y_n \sim \text{Gamma}(n, \lambda)$ with expectation $\mathbb{E}(Y_n) = n\lambda^{-}$.

Expected Minimum Cost The expected minimum assignment cost for M is given by [1]:

$$\mathbb{E}(C_n) = \sum_{k=1}^{n} \frac{1}{k^2} = H_n^{(2)} \tag{3}$$

Which is the generalized harmonic number of order two and converges to $\zeta(2) = \pi^2/6$. For the generalized harmonic numbers, $H_n^{(k)}$, $\lim_{k\to\infty} H_n^{(k)} = \zeta(k)$ for $k > 1$.

Greedy The minimum value of an $n \times n$ matrix is given by the order statistic $M_{1:n^2}$ with expectation $\mathbb{E}(M_{1:n^2}) = n^{-2}$. The expected value of the minimum cost assignment is not just $\sum_{i=0}^{n-1}(n-i)^{-2}$ because the expectation doesn't take into account the previous iteration's minimum value. To accomplish this we make use of the memoryless property of the exponential distribution to observe that the expected difference in minimums betweens iterations is the expected minimum value given by $M_{i:k^2}$. If we add up all these differences we get the expected minimum value of the k'th iteration; summing all these expectations then yields the expected minimum cost assignment:

$$\mathbb{E}(C_n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} \frac{1}{(n-j)^2} = \sum_{j=0}^{n-1} \frac{(n-j)}{(n-j)^2} = \sum_{j=0}^{n-1} \frac{1}{n-j} = H_n \tag{4}$$

This is the harmonic number of order one which does not converge. The resulting approximation factor is:

$$\alpha_n = \frac{H_n}{H_n^{(2)}} \tag{5}$$

Random The random algorithm will simply select an assignment permutation, so we are just adding up n $\text{Exp}(1)$ distributed random variables leading to an expected cost of:

$$\mathbb{E}(C_n) = \sum_{i=1}^{n} \mathbb{E}(M_{i,\pi_i}) = n \tag{6}$$

And approximation factor:

$$\alpha_n = \frac{n}{H_n^{(2)}} \tag{7}$$

From this analysis one concludes that the greedy algorithm has an unbounded approximation factor that grows signifigantly slower than that of randomly selecting assignments.

5 Evaluation

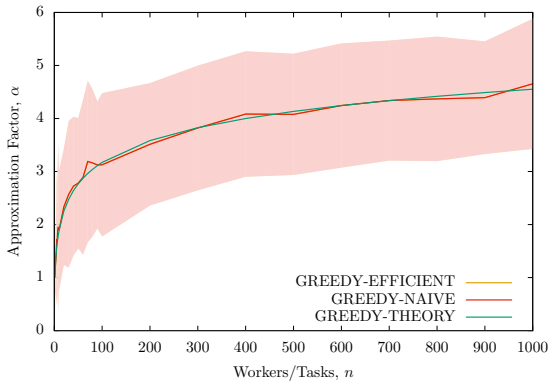


Figure 1: Greedy analytical and empirical approximation factor with 95% confidence interval.

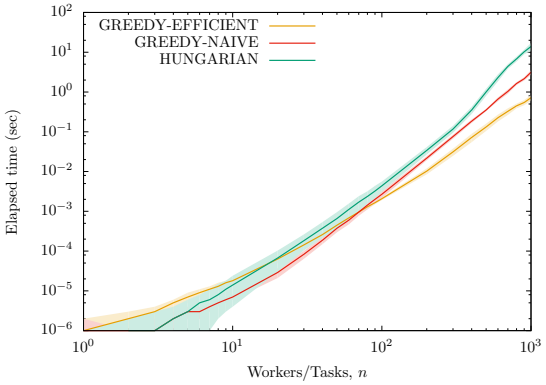


Figure 2: Elapsed wall time with 95% confidence interval.

To illustrate the preceeding results, Figure 1 shows the approximation factor for the greedy algorithm implementations against the derived approximation factor. The simulated results are based on 120 $n \times n$ standard exponentially distributed matrices for $1 \leq n \leq 1000$. Using the same conventions for the approximation factor, Figure 2 illustrates the runtime characteristics of the algorithms after rejecting outliers due to system fluctuations. Results obtained from source code compiled with -O3 flags and ran on a Xeon E3-1245 v5 3.5 Ghz system with 32 GBs of 2133Mhz DDR4 RAM. The algorithms coincide with the theoretical time complexities as shown in Table 1.

Solver	MSE
GREEDY-EFFICIENT	0.002139
GREEDY-NAIVE	0.014161
HUNGARIAN	0.232998

Table 1: Mean square error of fitted model to mean runtime for each solver. Models given by the corresponding time complexity. Fit by Levenberg-Marquardt.

6 Summary

	Brute	Random	Greedy	Hungarian
Complexity	$\mathcal{O}(n!)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^3)$
α_n	1	$n/H_n^{(2)}$	$H_n/H_n^{(2)}$	1

Table 2: Merits of each approach

Exact solutions can be delivered by the brute method when a handful of workers are being considered, and the Hungarian method should be considered for all other instances. Approximate solutions can be provided by the greedy algorithm with logarithmic degeneracy while providing a linear factor improvement over the Hungarian method. For inputs greater than those considered, the parallel Auction algorithm [3] is a suitable alternative and the subject of future work.

References

[1] Aldous, D. J. The ζ (2) limit in the random assignment problem. *Random Structures & Algorithms 18*, 4 (2001), 381–418.

[2] Balakrishnan, N., and Rao, C. Handbook of statistics 16: Order statistics-theory and methods, 2000.

[3] Bertsekas, D. P. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research 14*, 1 (1988), 105–123.

[4] Durstenfeld, R. Algorithm 235: random permutation. *Communications of the ACM 7*, 7 (1964), 420.

[5] Heap, B. Permutations by interchanges. *The Computer Journal 6*, 3 (1963), 293–298.

[6] Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2 (1955), 83–97.

[7] Kurtzberg, J. M. On approximation methods for the assignment problem. *Journal of the ACM (JACM) 9*, 4 (1962), 419–439.

[8] Michael Steele, J. Probability and statisitics in the service of computer science: illustrations using the assignment problem. *Communications in Statistics-Theory and Methods 19*, 11 (1990), 4315–4329.

[9] Munkres, J. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics 5*, 1 (1957), 32–38.

[10] Williamson, D. P., and Shmoys, D. B. *The design of approximation algorithms*. Cambridge university press, 2011.