Average Case Complexity Analysis of Algorithms for the Linear Assignment Problem

Tor Sørevik
Dept. of Informatics
University of Bergen

May 14, 1993

Abstract

In this paper we present a probabilistic analysis of average case running time for augmenting path algorithms for solving the Linear assignment problem.

Our analysis may be used to estimate the running time provided the data have a uniformly random distribution. Contrary to the standard analysis our analysis explains how the running time depends not only on the problem size, but also on the structure of the input data. Our analysis is validated by numerical experiments.

The new insight gained leeds to a proposed modification of the algorithm.

Keywords: Linear Assignment Problem, Average Case Analysis, Augmenting Path Algorithms

1 Introduction

The Linear Assignment problem (LAP) may be defined as minimizing the cost of assigning n people to n jobs or tasks. A specific problem is defined through its cost matrix where the element c_{ij} is the cost associated with assigning person i to job j. LAP is of great importance as problem in its own right as well as a kernel routine for more complex problems (Quadratic assignment problem, TSP, etc) [Law76] [PS82].

Numerous algorithms for solving the LAP have appeared after the first one by Kuhn [Kuh55] in 1955. The problem may be formulated as a linear programming problem. In this case the special structure of the problem may be used to design polynomial time simplex routines [BGK77], [Bal86], [Hun83]. Alternatively we may describe the LAP as a matching problem in a weighted bipartite graph. Letting the n people represent one half of the nodes and the n jobs the other. The weights being defined by the cost matrix. Solving the LAP is then equivalent to finding the minimum cost perfect matching in this graph. We refer to the technique used for solving this matching problem as augmenting path techniques [JV87], [CMT88]. There also exist an iterative technique for solving LAP called the auction algorithm [Ber81], [Ber88].

In this paper we investigate different variants of augmenting path routines. Correctly implemented these methods all have complexity $O(n^3)$. Experiments do, however, reveal great differences in running time, depending on the input data as well as seemingly minor differences in the algorithm.

The purpose of this paper is to try to explain some of these differences. In particular how the running time is influented by properties of the data as well as by the problem size. Having no parameter related to the quality of the data, the bounds given by the standard worst case analysis give no indication of the observed datadependency. By making some assumptions on the input data, allowing us to parameterize these, we are able to perform an average case analysis which indicates how the running time of the algorithm is expected to vary with the quality of the input data. We have also experienced that the worst case analysis is far to pessimistic for predicting actual running time. But it is not our primary interest to make more accurate prediction of running time since we believe that 'the average case' is just as unlikely to take place as 'the worst case'.

Because we are primarily interested in saying something on the qualitatively behavior of the algorithms, we do not hesitate to sacrifice rigorous mathematics for simplicity.

2 The Algorithm

In this section we first give an overview of the computational framework in which all augmenting path routines fit. Then we focus in on some of the details, where we follow the description given in Carpaneto & al. [CMT88].

All augmenting path algorithms rely on the following fundamental properties of the problem:

> Any constant may be added to or subtracted from each element in a row or a column of the cost matrix without affecting the optimal solution.

The idea behind the algorithm is to use this property to construct a reduced cost matrix. If this matrix contains only nonnegative values, then if a solution of zero cost is feasible, this solution is optimal.

The reduced cost matrix is define as

$$c^*_{ij} = c_{ij} - u_i - v_j \tag{1}$$

where u_i and v_j are the dual variables in the Linear programming formulation of the problem.

Throughout the process dual feasibility is maintained, i.e. $c^*_{ij} \geq 0$, while, in each step, the primal infeasibility is reduced, until a primal feasible solution is found.

The entire family of augmenting path algorithms based on this philosophy fits in to the following general framework:

algorithm LAP;
begin
    initialize; (* compute initial values of dual variables
        and find a dual feasible assignment *)
    repeat
        select an unassigned row, i;
        findpath; (* find an augmenting path from i to unassigned
            column j. If necessary update dual variables *)
        reassign; (* reorganize assignment to also include i and j *)
    until (all rows are assigned);
end.

In the sequel we describe a simple initialization. It consists of two parts, column reduction and row reduction. In the column reduction each column is scanned for its minimum value. This value is assigned to the corresponding dual variable v_j, and the reduced costs are computed. As we proceed, assignments are made whenever a zero reduced cost appears in an unassigned row. The row reduction is equivalent, but is only applied to unassigned rows.

The following two vectors of indices are used:

$f \; : \; f_i$ = column assigned to row i; $(i = 1, ..., n)$
$\bar{f} \; : \; \bar{f}_j$ = row assigned to column j; $(j = 1, ..., n)$

procedure *initialize*;
$f := \bar{f} := 0$;
for $j := 1$ **to** n **do** (* column reduction *)
    find $c_{rj} = \min_i \{c_{ij}\}$;
    $v_j := c_{rj}$;
    if $(f_r = 0)$ **then**
        $\bar{f}_j := r$;
        $f_r := j$;
        $u_r := 0$;
    endif;
enddo;
for $i := 1$ **to** n **do** (* row reduction *)
    if $(f_i = 0)$ **then**
        find $c_{ir} = \min_j \{c_{ij} - v_j\}$;
        $u_i := c_{ir} - v_r$;
        if $(\bar{f}_r = 0)$ **then**
            $\bar{f}_r := i$;
            $f_i := r$;
        endif;
    endif;
enddo;
end(*initialize*);

The procedure *findpath* below is searching for a path of zero reduced cost starting from the unassigned row i. If no such path is found, the dual variables are updated and the search continues. Finally, when a path to an unassigned column, j, is found, the procedure stops. Details are as in Carpaneto e. al. [CMT88]. We use the following sets and vectors:

LR - set of labelled rows,
UC - set of unlabelled columns,
ptr - vector of row indices, ptr_j = row preceding column j in the current path
$temp$ - vector of reduced costs, $temp_j = \min \{c^*_{ij} : i \epsilon LR, i \neq \bar{f}_j\}$.

```
procedure findpath;
LR := {i};  UC := {1,...,n};
forall j∈{1,...,n} do temp_j := ∞;
repeat
        r := next element in LR;
        forall j∈UC do
              if (c_rj − u_r − v_j < temp_j) then
                 temp_j := c_rj − u_r − v_j;
                 ptr_j := r;
              endif;
        enddo;
        if ({temp_j = 0 : j∈UC} = ∅) then
        (* update dual variables *)
              d := min(temp_j |: j ∈ UC);
              forall k∈LR do u_k := u_k + d ;
              forall j∈{1,...,n} do
                 if temp_j = 0 then
                    v_j := v_j − d;
                 else
                    temp_j := temp_j − d;
                 endif;
              enddo;
        endif;
        j := first element in {temp_l = 0 |: l∈UC};
        if (f̄_j > 0) then
              LR := LR ∪ {f̄_j};
              UC := UC\{j};
        endif;
until (f̄_j = 0);
end(findpath);
```

Finally the procedure *reassign* reorganizes the assignments. The new assignments will include row i and column j in addition to all previously assigned rows and columns. The *ptr*-vector is used to backtrack the augmenting path which determines the appropriate updates of f and $\bar{f}$.

```
procedure reassign;
while(i ≠ ptr_j) do
        l := ptr_j;
        f̄_j := l;
        k := f_l;
        f_l := j;
        j := k;
```

enddo;
end(*reassign*);

The worst case complexity of the initialization procedure described here is easily seen to be $\Theta(n^2)$. Since the while loop of the main program and the repeat loop of *findpath* is iterated at most n times, the augmenting path part has worst case complexity $O(n^3)$. Giving an overall worst case complexity for the entire problem of $O(n^3)$.

3 Average case analysis

As will become evident by the following analysis the property of the data which is important, is whether multiple choices for assignments can occur. We therefore make the assumption that our data consist of uniformly distributed random integers on the interval $[0, C)$. The parameter C may then be used as a predictor of how many zero element we expect in a given row or column of the (reduced) cost matrix.

The algorithm, as described in Section 2 separates nicely into two disjoint parts which can be analyzed separately. In the analysis of the initialization we not only need an estimate of the average work, but also on how many assignments we might expect to do, as this determine the work left to be done in the augmenting path part.

3.1 Estimates for the initialization

The initialization falls into two separate parts. First we scan every column for its minimum and subtract this minimum from all entries in the column. Having n entries in a column, we expect there to be exactly one entry less than C/n in each column. With equal probability for each number to occur an estimate of the lowest entry is

$$C_{min} = C/2n. \tag{2}$$

The reminding $n-1$ entries in the column are assumed to be uniformly distributed on the interval $C/2n \leq x < C$. Thus the possibility for any of them to be equal to C_{min} is $1/(C - C/2n)$. The expected number of zero elements in each column thus becomes

$$1 + \frac{n-1}{C - C/2n} = 1 + \frac{n}{C}\left(1 - \frac{1}{2n-1}\right) \approx 1 + \frac{n}{C}. \tag{3}$$

After having performed the column reduction, we might view the situation from a row's perspective. For each entry in the row the possibility of finding a zero element is $(1 + n/C)/n$. Then the number of zeroes in a row is a stocastic variable with a binomial distribution. The possibility of exact s zero elements is then

$$p(s) = \binom{n}{s}\left(\frac{n+C}{nC}\right)^{s}\left(1 - \frac{n+C}{nC}\right)^{n-s}. \tag{4}$$

Setting

$$C = kn \tag{5}$$

and using

$$\lim_{n\to\infty}\left(1-\frac{a}{n}\right)^n = e^{-a}, \tag{6}$$

we find the expected number of rows without zeroes to be

$$np(0) = n\left(1-\frac{1+k}{k}\frac{1}{n}\right)^n \approx ne^{-\frac{1+k}{k}}. \tag{7}$$

When $C \to \infty$, we may assume that there is a unique minimum in each column. In this case, if a row has a zero, by construction, no other row has zero in the same column, and consequently we might assign the row to this column. The expected total of unassigned rows after column reduction

$$np(0) = n\left(1-\frac{1}{n}\right)^n \approx ne^{-1}. \tag{8}$$

For all other values of C (7) is a lower bound on unassigned rows and (8) an upper bound.

The row reduction is only applied to the non-assigned rows. Setting the unit cost of scanning a row or a column to ω_i, we get as an estimate for the initialization cost:

$$W_i(n,\infty) = \omega_i n(1+e^{-1}). \tag{9}$$

For $C < \infty$ the cost is bounded by

$$\omega_i n(1+e^{-\frac{1+k}{k}}) \leq W_i(n,C) \leq \omega_i n(1+e^{-1}). \tag{10}$$

In a sequential implementation ω_i of (10) is a linear function of n and a constant in our parallel one. In either case ω_i is independent of C.

The row reduction is applied to n' rows, only. The value of n' is bounded by (7) and (8). Analogous arguments shows that the expected number of columns which does not get a zero element in the row reduction is

$$np'(0) = n\left(1-\frac{1+k}{k}\frac{1}{n}\right)^{n'}. \tag{11}$$

When $C \to \infty$ then $n' \to ne^{-1}$ and we expect

$$n(1-e^{-e^{-1}}) \tag{12}$$

to be the number of columns with a new zero element. Of these new ones we expect a fraction of e^{-1} to appear in unassigned columns. We sum up the above discussion in the following theorem

Theorem 1 *If each column and row of the cost matrix has one element with the minimum value and these are random scattered over the matrix, then the expected number of assignments produced by the procedure "initialize" is*

$$n\left(1 - e^{-(1+e^{-1})}\right) \approx 0.745n. \tag{13}$$

Proof: By (8) assignments after the column reduction is $n(1 - e^{-1})$. Adding up the new assignments after row reduction and multiplying (12) by e^{-1} gives the result □

As discussed above we expect the condition of the theorem to be satisfied when $C \rightarrow \infty$. Decreasing the value of C is expected to give more zero elements in the reduced cost matrix. Using (7) we obtain $n(1 - e^{-(1+k)/k})$ as an estimate on the number of rows with at least one zero. In this case we cannot assume that all these rows might be assigned. Thus (11) is an upper bound on the number of assignments after the column reduction. In cases where this bound is met, n' in (11) becomes $n' = e^{-(1+k)/k}$. The expected number of columns obtaining at least one zero element in the row reduction is

$$n\left(1 - e^{-\frac{1+k}{k}e^{-\frac{1+k}{k}}}\right). \tag{14}$$

Taking the proper fraction of this and adding up the total number of assignments give the following theorem.

Theorem 2 *On a uniformly random distributed cost matrix with elements in the interval $[0, C)$, $C = kn$. The expected numbers of assignments after initialization is bounded by:*

$$0.745n \approx \left(1 - e^{-(1+e^{-1})}\right)n \leq n - r \leq \left(1 - e^{-\frac{1+k}{k}(1+e^{-\frac{1+k}{k}})}\right)n \tag{15}$$

4 Expected running time for the Augmenting Path

Suppose our initialization leaves us with r unassigned rows. The augmenting path routine starts by picking an unassigned row. The zeroelements in the reduced cost matrix of this row occurs in a column assigned to another row (that is why it is not assigned in the initialization). We therefore pick a new column by finding the minimum value in the vector *temp*. The pointer *ptr* tells us from which row this value originates. The probability of the second column being applicable is

$$p_r(2) = \frac{r}{n-1} \tag{16}$$

As soon as an unassigned column is found we stop. If not we repeat the process, removing the inspected column from the set of applicable columns. If $p_r(i)$ is the probability for us to stop after i attempts we get the following recurrence

$$p_r(i) = \left(1 - \sum_{j=2}^{i-1} p_r(j)\right) \frac{r}{n - i + 1}. \tag{17}$$

Extracting the last term from the sum, followed by some minor algebraic manipulations, give the following recursion formula

$$p_r(i) = p_r(i-1) \frac{n - (i + r - 2)}{n - (i - 1)}. \tag{18}$$

Which applied recursively together with (16) gives

$$p_r(i) = \frac{(n - (r+1))!}{(n - (r + i - 1))!} \frac{(n-i)!}{(n-1)!} = \binom{n-i}{r-1} / \binom{n-1}{r}. \tag{19}$$

The expected path length is than defined as

$$E(p_r) = \sum_{i=1}^{n} i p_r(i) \tag{20}$$

which can be shown to be

$$E(p_r) = \frac{n + r + 1}{r + 1}. \tag{21}$$

The process analysed above assign one more row. To assign all the unassigned rows, the process has to be repeated for row number $r-1, r-2, ..., 2$ and 1. Assuming the construction of all these paths is independent of each other, an estimate of the overall work is obtained by summing over $E(p_r(j))$ for $j = r, ..., 1$. Approximating the sum by an integral we obtain the following estimate

$$W_a(n, r) = \omega_a \sum_{j=r}^{1} E(p_j) \approx \omega_a(n(\log r - \log 2) + (r - 1)). \tag{22}$$

The bounds on $n - r$ in Th. 2 applied to (22) indicates how the distribution of input data dictates r, the number of assignments left to do in the augmenting path routine. It has, however, no influence on the parameter ω_a, which similar to ω_i is a linear function of n in a sequential implementation and a constant in our parallel.

If one assumes (22) to be an estimate for the computational work of the augmenting path and compares this with the actual running time a significant discrepancy is revealed (See Figure 1 and 2). A closer look at the details of the computational experiments shows that the discrepancy is almost entirely due to differences occurring in the tail of the computation.

The explanation for this phenomena is that the underlying assumption that r succeeding calls to the augmenting path routine create r independent paths, is violated in practice. After a while the different rows are connected by paths. When searching for a new augmenting path one might easily links onto one of these. The algorithm will now follow this path and make an extension to it. This problem will, of course, be most apparent at the end of the computation.

4.1 A Modified augmenting path algorithm

The problem described at the end of the previous section relates to the selection criterion used for choosing the next column. We note that even in the extreme case ($C \to \infty$) there is more than n zero elements in the reduced cost matrix and consequently occasional multiple choices exists. Always choosing the first zero element (as in CMT's algorithm) does not take advantage of these multiple choices, and we are therefore always repeating and extending previous computation. To minimize this effect we introduce a global counter on each column, which is incremented each time a column is examined. Then if multiple choices occur we choose the column least used so far. This does not eliminate the problem described, but should improve on it.

The proposed modification makes the augmenting path algorithm similar to that of Jonker and Volgenant [JV87].

Because lowering the cost range has the effect of making multiple zeroes occurring more frequently, we expect this modification to make a larger differences as we decrease C.

5 Numerical experiments

n	C	experiments	lower bound	upper bound
100	10000	75.1	74.5	74.8
200	80000	149.0	149.1	149.2
400	160000	298.7	298.1	298.4
600	360000	448.2	447.2	447.5
800	640000	593.5	596.3	596.5
1000	1000000	747.1	745.4	745.6
1200	1440000	892.2	894.4	894.7
1400	1940000	1048.1	1043.5	1043.8
1600	2560000	1195.7	1192.7	1192.8
1800	3240000	1344.4	1341.6	1341.9
2000	4000000	1488.3	1490.7	1491.0

Table 1 *The numbers reported are assignments made by the initialization routine. The first two column defines the problem. The third gives experimental results averaged over 10 samples. The last two column gives lower and upper bound as defined by (15).*

n	C	experiments	lower bound	upper bound
100	100	75.7	74.5	89.6
200	200	151.6	149.0	179.3
400	400	303.2	298.1	358.7
600	600	461.4	447.2	538.0
800	800	610.4	596.2	717.4
1000	1000	763.5	745.3	896.7
1200	1200	914.1	894.4	1076.1
1400	1400	1068.9	1043.4	1255.4
1600	1600	1214.9	1192.5	1434.8
1800	1800	1371.0	1341.6	1614.1
2000	2000	1521.6	1490.7	1793.5

Table 2 *The numbers reported are assignments made by the initialization routine. The first two column defines the problem. The third column gives experimental results averaged over 10 samples. The last two column gives lower and upper bound as defined by (2).*

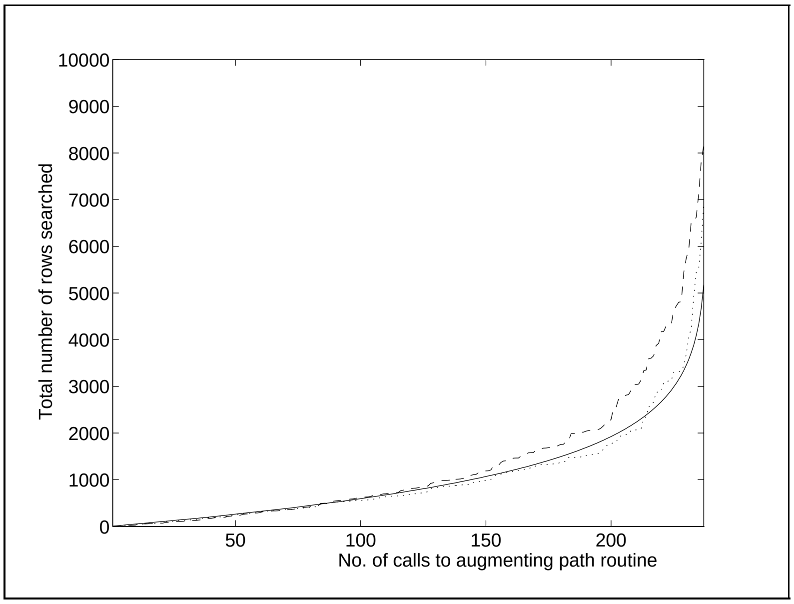

Figure 1: *Accumulated length of path on a 1000 by 1000 example, with Cost range 1000. 237 rows were unassigned after initialization. Solid line is model, dashed line is standard version and dotted line is modified version*

We have run extensive experiments on a massively parallel computer, MasPar MP-2, using the code developed in [SS92]. All data is uniformly random integer generated by MasPar's own random generator. Only dense problems are considered.

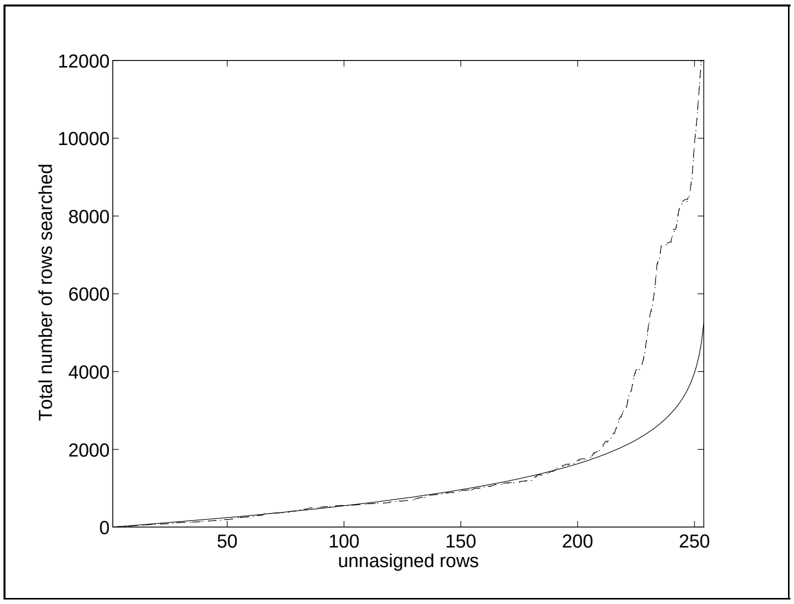

Figure 2: *Accumulated length of path on a 1000 by 1000 example, with Cost range 100000. 254 rows were unassigned after initialization. Solid line is model, dashed line is standard version and dotted line is modified version*

In Table 1 and 2 we report on the behavior of the initialization routine and compare it with the bounds given in (2).

We observe that the number of assignments obtained in practice is almost perfect predicted by our model in the case of high cost range. For moderate size cost range the observed numbers always stays within the bounds we give. We also notice that it stays closer to the lower bound than to the upper bound.

In Figure 1 we illustrate the typical behavior of the augmenting path routine. There is little discrepancy between the model and the actual for the first half of the iterations. Then the standard version starts picking up old paths, making the accumulated pathlength longer than the predicted one. The proposed modification does not prevent this from happening, it only postpone it. In this example the picking up of old paths does not seem to start before more than 4/5 of the calls to the augmenting path routine are done.

The proposed modification gives significant improvements while C is in the low cost range. Hardly any improvement is achieved in the high cost range (Figure 2).

References

[Bal86] M. Balinski. A competitive (dual) simplex method for the assignment problem. *Mathematical Programming*, 34:125–141, 1986.

[Ber81] D. Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21:152–171, 1981.

[Ber88] D. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14:105–123, 1988.

[BGK77] R. Barr, F. Glover, and D. Klingman. The alternating basis algorithm for the assignment problem. *Mathematical Programming*, 13:1–13, 1977.

[CMT88] G. Carpaneto, S. Martello, and P. Toth. Algorithms and codes for the assignment problem. *Annals of Operations Research*, 13:193–223, 1988.

[Hun83] M. Hung. A polynomial simplex for the assignment problem. *Operations Research*, 31:595–600, 1983.

[JV87] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[Kuh55] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, N. J., 1982.

[SS92] S. Storøy and T. Sørevik. An simd, fine-grained, parallel algorithm for the dense linear assignment problem. Technical Report Report 72, Dept. of Informatics, University of Bergen, 1992.