A MATCHING ALGORITHM WITH APPLICATION TO BUS OPERATIONS

M.A. FORBES, J.N. HOLT, P.J. KILBY AND A.M. WATTS

Department of Mathematics, University of Queensland, Australia

ABSTRACT. The problem of constructing shifts of work for busdrivers from given workblocks is shown to be a minimum cost matching problem. A branch and bound algorithm for this problem is derived which, while not having the guaranteed polynomial time complexity of the well known efficient labelling methods , is shown in extensive computational testing to be (1) extremely fast on the class of problem where the costs do not satisfy the triangle inequality, and (2) very easy to implement.

Keywords: Matching, Branch and Bound, Bus driver scheduling

1. Constructing Busdriver Shifts. Large mass transit bus operations in metropolitan areas involve many hundreds of bus driver shifts (or duties) daily. For example, the Brisbane City Council operates approximately 220 shifts at each of 3 depots. The final stage in the construction of these shifts at the Brisbane City Council currently involves combining two pieces of work or workblocks which are designed in advance to fit either side of a mealbreak in accordance with the industrial award under which drivers are employed. Associated with each workblock is a starting and ending time and a starting and ending location. The method of construction of the pieces of work is irrelevant to the present paper, but is an important topic in its own right and is the subject of considerable current work.

Australasian Journal of Combinatorics 4(1991), pp 71-85

The objective facing the scheduler is to combine the workblocks in such a way as to minimize the total cost. To this end, it is possible to compute the number of "pay minutes" associated with all feasible pairings of workblocks. This computation takes account of waiting time, meal breaks and total spread.

Because the workblocks overlap in time throughout the day, it is not possible simply to divide them into two sets and assign a member of each set to a member of the other. In the next section, we define the perfect matching problem and show how the shift construction problem is an instance of it. In section 3 we review the existing algorithms for this problem. Section 4 introduces the algorithm we have used to solve the Brisbane City Council's problem, and finally we give some computational experience with the algorithm in section 5.

2. Formulation of the matching problem. A formal statement of the minimum cost perfect matching problem (MP) is as follows. Given an undirected graph $G = (V, E)$, where $|V|$ is even, a perfect matching M is a subset of edges of G such that every node $i \in V$ is incident to exactly one edge in M. If a cost c_{ij} is associated with each edge $(i, j) \in E$, then MP is to find a perfect matching for which the sum of the costs associated with the edges in M is minimized.

In the context of the busdriver shift construction problem, the set of nodes V is the set of workblocks, and the edges E define the feasible connections between workblocks which may be used to define a day's work. Infeasible connections are caused for example by insufficient time to fit a mealbreak, by the total duration of the shift exceeding some maximum value or by the gap between the workblocks exceeding some limit. In addition, there is overlap in time between many pairs of workblocks, and for other pairs there is insufficient time to travel from the endpoint of one block to the starting point of the other. The costs on the edges are the number of pay minutes associated with the resulting shift.

We can formulate the problem as an integer programming problem in the following way. Let $V = \{1, \ldots, n\}$ where n is even. For each $(i,j) \epsilon E$, define $c_{ji} = c_{ij}$. For all $(i,j) \notin E$, define $c_{ji} = c_{ij} = \infty$. This completes a symmetric cost matrix. Then we have

$$MP: \qquad \text{minimize} \sum_{i,j=1}^{n} c_{ij}\, x_{ij} \tag{1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n \tag{2}$$

$$x_{ij} = 0 \text{ or } 1, \qquad 1 \leq i, j \leq n, \tag{3}$$

$$x_{ij} = x_{ji}, \qquad 1 \leq i, j \leq n. \tag{4}$$

This formulation is the same as that of the well-known assignment problem (AP), except that the cost matrix is symmetric and the solution is required to be symmetric. Eqns (2) are the usual origin availability constraints, and substituting Eqns (4) into them gives the destination requirements constraints of the AP. The differences from the AP are such that the convex set of feasible solutions to the problem obtained by relaxing Eqns (3) to $x_{ij} \geq 0$ does not in general have integer valued extreme points, so that the simplex method cannot be used to solve MP. In [5], it is shown how to introduce cut-inequalities to produce the desired properties if the integrality is dropped.

3. Review of Algorithms for matching problems. The non-bipartite minimum cost perfect matching problem is one of a number of combinatorial optimization problems for which polynomial time algorithms are known. The idea of blossoms, as developed first by Edmonds [5], was used by Lawler [8] to develop an algorithm which can be implemented in computer time of order n^3, where n is the total number of objects to be matched. Derigs' [2] shortest augmenting path method is a further extension of these ideas leading to an algorithm which is also of order

n^3. Derigs and Metz [3] have described how an optimal fractional matching can be used to start the shortest augmenting path method for the matching problem and thereby dramatically reduce the amount of computer time required. These methods are also referred to as efficient labelling methods. In addition, Grötschel and Holland [7] introduced a cutting plane algorithm based on the simplex algorithm which is competitive with the best of the efficient labelling algorithms.

The implementation of the algorithms mentioned above is far from trivial. It is worthwhile, in view of this, to investigate the practical performance of other algorithms which may not have the theoretical computational bounds but which are significantly easier to implement. In the next section, we present a Branch and Bound method which is shown in extensive numerical tests to be highly effective for the class of matching problem we are considering. It is not the first Branch and Bound algorithm for this problem. Devine and Glover [4] presented a somewhat different branch and bound algorithm.

The method we propose is similar to that of [3] in that the relaxation step of the Branch and Bound approach uses the related assignment problem to obtain a partial matching and hence a lower bound on the solution of the current candidate matching problem. The branching process removes the odd cycles which may occur in the solution of the assignment problem.

While there are no performance guarantees for this method, it is found in the numerical experiments reported later that the matching problem can generally be solved in a time only slightly greater than that needed for the assignment problem resulting from the initial relaxation. Furthermore, the method is easy to implement.

4. The new branch and bound algorithm. The Branch and Bound approach to be described uses an AP relaxation of MP. In order to describe the steps in this

approach, it is necessary first to review the main features of the algorithm we use to solve the AP, which is the method of Engquist [6].

4.1 Assignment Algorithm.

As usual for the assignment problem, use is made of the reduced costs c'_{ij}. These are obtained by row and column operations on the original matrix of costs c_{ij}. That is, for some set of values u_i, v_j, the reduced costs are given by

$$c'_{ij} = c_{ij} - u_i - v_j.$$

It is well known that an assignment which is optimal with respect to the reduced costs is also optimal with respect to the original costs. Hence, if the reduced costs are non-negative and the assignment has a zero value with respect to them, then it is an optimal assignment for the original problem. The assignment problem is thus solved if values u_i, v_j, x_{ij} can be found such that

$$\begin{aligned}
&c'_{ij} \geq 0, \\
&x_{ij} = 0 \text{ or } 1, \\
&\sum_{j=1}^{n} x_{ij} = 1, \\
&\sum_{i=1}^{n} x_{ij} = 1, \\
&\sum_{i,j=1}^{n} c'_{ij}\, x_{ij} = 0.
\end{aligned}$$

The Engquist algorithm is an iterative one. At the end of each iteration there is a mapping $i \mapsto A_i, 1 \leq i \leq n$ such that if $j = A_i$ then $c'_{ij} = 0$. In terms of the reduced costs, the mapping has a zero cost.

The node j is said to be deficient if there is no i such that $j = A_i$. If there is more than one such i, then j is said to be abundant. During each iteration, the mapping

A_i is changed using an associated shortest path network as described by Engquist so that the number of deficient nodes and the abundance of one of the abundant nodes each decreases by one and the reduced costs are adjusted so that the new mapping has zero cost. The solution of the assignment problem is complete when there are no deficient nodes and is given by:

$$x_{ij} = \begin{cases} 1 & \text{if } j = A_i \\ 0 & \text{otherwise.} \end{cases}$$

4.2. Branch and Bound. The ingredients of a general Branch and Bound method for an integer programming problem are outlined by Parker and Rardin [9]. Balas and Toth [1] present a comprehensive account of Branch and Bound for the Travelling Salesman Problem, which also includes AP relaxations. The terminology used here is standard for Branch and Bound.

Our algorithm for MP begins with MP itself on the candidate list. It is selected, and the AP relaxation obtained by omitting Eqns (4) is then solved by the Engquist algorithm. If the solution of this relaxed problem is symmetric, i.e. if the final mapping satisfies the condition $j = A_i \Longrightarrow i = A_j$ so that $x_{ij} = x_{ji}$ for all i and j, then the optimal solution to MP has been obtained.

In the case where the solution is not symmetric, there are cycles defined by the mapping. Let $n_1, n_2, \ldots, n_{p+1}$ be a sequence such that $n_{i+1} = A_{n_i}$ and $n_{p+1} = n_1$. If p is even, this is an even cycle, and in that case the assignment solution is not unique. For nodes in any such even cycle, the assignment can be replaced by

$$n_{2i} = A_{n_{2i-1}}, \qquad n_{2i-1} = A_{n_{2i}}, \qquad 1 \leq i \leq p/2,$$

with no change in the objective function, so that the assignments involving these nodes are symmetric.

In the case where there is an odd cycle, i.e. where p is odd, it is necessary to break the cycle and it is here that the branching process is used. For any odd cycle, it is clear that a matching between any two adjacent nodes in the cycle either occurs or does not occur in a feasible solution to MP. The branching step involves selecting such a pair of nodes and generating two new candidate matching problems. In the first of these the pair is matched, and in the second the pair is forced not to be matched. n_1 and n_2 are chosen from the shortest odd cycle.

The subproblems are bounded immediately they are generated, using one iteration of the Engquist algorithm as described below. This is a major advantage in using the Engquist algorithm over other assignment methods.

To consider the matching problem in which n_1 and n_2 are not matched, the condition $x_{n_1 n_2} = x_{n_2 n_1} = 0$ is imposed, which is equivalent to setting $c_{n_1 n_2} = \infty$. Now consider the optimal solution to the AP relaxation of the parent problem. This gives a lower bound on the optimal solution for the subproblem, but this bound can be tightened as follows. Let k be the node with the minimum reduced cost from n_1, i.e. $c'_{n_1 k} = \min_j c'_{n_1 j}$. Then set $A_{n_1} = k$ and increase u_{n_1} so that the reduced cost $c'_{n_1 k}$ is zero. The node n_2 is then deficient and k is abundant. One iteration of Engquist is therefore needed to recompute the solution of the assignment relaxation for this subproblem. The new lower bound is equal to that of the parent plus $c'_{n_1 k}$ plus the increase in objective computed by Engquist.

For the matching problem in which n_1 and n_2 are matched, these two nodes are removed from V. Considering again the optimal mapping for the parent's AP ,but with these nodes and their connections removed, node n_3 then becomes deficient since there is no longer any mapping to it, and the mapping from n_{p-1} must be diverted from n_1 to a suitable node k remaining in the set. k is chosen by the

condition

$$c'_{n_{p-1}k} = \min_{j \neq n_1, n_2} c'_{n_{p-1}j}$$

and the replacement mapping is given by

$$A_{n_{p-1}} = k.$$

Again, k is abundant and one iteration of the Engquist algorithm is required to solve the relaxed subproblem. A new bound for the subproblem is obtained in a similar manner to that described above for the other subproblem.

Once the subproblems are bounded (or solved if their relaxation produces a feasible matching), they can be fathomed in the usual manner by comparing their bounds with the value of an incumbent best known matching solution. If a problem has a bound smaller than the incumbent, it is placed on the candidate list according to its bound. If a solved candidate problem has a better solution than the incumbent it becomes the incumbent. The candidate list can then be purged of problems with bounds worse than the new incumbent. The selection from the candidate list is carried out according to best bound. The solution is complete when the candidate list is empty.

The process outlined develops a binary tree of candidate problems, each with certain sets of matchings included and excluded.

5. Computational Results. The algorithm discussed in the previous section was implemented in FORTRAN and tested on a PYRAMID 9810 computer. Test problems were generated with a view to investigating the computational performance of the method as a function of three key problem characteristics. These are the size of the matching problem, the density of feasible arcs and the range of costs on the arcs.

Problems with the number of nodes (n) in the range 100 to 500 were considered. For each choice of n, five approximate arc densities were selected, ranging from 20% to 100%. Twenty random problems were generated for each pair of the two parameters, and for each of the cost ranges 1 – 100, 1 – 1000 and 1 – 10000. The costs on the arcs in each problem were selected randomly from a uniform distribution over the appropriate range.

Tables 1 to 5 present the results of the tests. The numbers shown in the Tables are the average over the 20 problems considered for each set of three parameters.

The first thing to notice about the results is that the cpu times required to solve the problems are very small, making the method attractive even for large problems. Furthermore, the times to complete the matching solutions (Table 5) are generally small fractions of those needed to solve the initial AP relaxation (Table 3). There are two exceptions to this in Table 5. In group (a) of that table, there are two average times, 17.97 and 11.94 which are larger than the times for the initial relaxations. Each of these was caused by a single outlier problem which required a large number of extra Engquist iterations in the Branch and Bound (Table 2). For example, the 17.97 figure resulted because one of the problems required 662 iterations to move the objective function value from 51 for the assignment to 52. This implies that many alternative assignments existed with the same value of the objective function. It is interesting to note that no such cases were encountered in group (b) or group (c) of the results, where the spread of arc costs is greater. In any case, the computer times for these outlier problems are quite acceptable.

It is of interest to investigate the cpu time spent in branch and bound to complete the matching as a fraction of the total cpu time to solve the problems. A comparison can be made here with the results reported by Derigs and Metz[3] who also perform computational tests on randomly generated problems.

The fully dense problems considered by Derigs and Metz cover the same cost distributions and numbers of nodes as in Tables 1 to 5. Their average fraction of cpu time to complete the matching to total cpu time including assignment startup is 0.376 using their best startup procedure ASSIGN-2, with a minimum of 0.071 and a maximum of 0.846. The corresponding average fraction for the present method is 0.162 with a minimum of 0.08 and a maximum of 0.233 from the data in Tables 3 and 5 for the 100% dense case. It should be noted however, that neither of the two outliers mentioned earlier occur in this case. For the 80% dense case, which does include an outlier, the figures are :- average 0.212, minimum 0.094 and maximum 0.781. It is clear that, on this sample, the present method is performing better with respect to the ratio measure on the dense problems.

When considering sparse problems, Derigs and Metz observed a reduction in the fraction, using ASSIGN-1 startup. For example, for 200 node problems with 25% arc density, they observed an average of 0.055, taken over the three cost distributions. The average fraction for the present work, for 20% arc density and 200 nodes, is 0.17. In general , a reduction in the fraction with increased problem size or reduced arc density is not observed for the present method. It remains approximately the same over all the problems considered.

Table 1 shows the interesting result that for large, dense problems, the number of Engquist iterations for the initial AP relaxation decreases significantly as the cost range is increased. This implies that there is less degeneracy as the spread of costs increases. On the other hand, the average number of matchings obtained from the initial relaxation (Table 4), is not dependent on density or on cost range.

The ratio of the number of extra iterations to the number of initial iterations from Tables 2 and 1 respectively is less than the ratio of cpu times from Tables 5 and 3 respectively. This is because the extra Engquist iterations in the Branch and

Bound are all hard. By this, we mean that there is only one abundant and one deficient node, and so long paths through the shortest path network will possibly be required. In the initial stage, there are in general several abundant nodes , some of which will have short paths to the deficient node.

It should be noted that the problems considered here are not Euclidian. The algorithm presented would not be expected to perform as well on problems for which the arc costs satisfied the triangle inequality. The same comments apply to the blossom algorithms. The numerical testing of earlier codes is based on randomly generated problems as described here. There appears to be no comparison between performance on the two types of problem, and this could be an area for future work.

6. Practical application. The algorithm as presented here has been used successfully in the practical application involving matching blocks of work for bus drivers for the Brisbane City Council Transport Department. It is only applied when doing major reschedules which necessitate a new roster. Minor changes occur over the duration of the roster. Trips are added or deleted in response to new peak passenger demand, new routes are implemented, old routes rationalized or titmetables changed. Over the life of the roster, the effective savings due to the system therefore decline.

With three bus depots, each with around 220 shifts, the use of the program has produced savings estimated by Council officers to be in the order of $250,000 annually over the last four years. This figure has been calculated using the average saving as half the initial saving for the reasons stated above.

In the actual implementation the optimization code has been combined with code developed by the Council to compute the cost matrix taking into account the pay rates, industrial award constraints etc., and to output the optimal matching to other software systems.

Table 1: The average number of Engquist iterations to solve the initial AP relaxations. Costs uniformly distributed in the ranges (a) 1 – 100; (b) 1 – 1000; (c) 1 – 10000.

		Arc Density				
	n	20%	40%	60%	80%	100%
	100	37.40	37.80	37.50	37.40	41.50
	200	71.35	76.35	77.80	79.00	105.50
(a)	300	112.70	117.35	123.65	132.50	183.35
	400	147.40	162.05	176.45	191.25	267.45
	500	190.30	210.10	236.10	263.05	356.65
	100	37.30	36.75	35.50	36.50	36.05
	200	74.70	72.85	74.65	72.65	74.05
(b)	300	111.25	111.25	112.15	109.75	112.35
	400	146.10	150.45	146.30	147.45	148.25
	500	183.10	183.60	184.40	186.85	189.55
	100	36.25	36.45	35.10	36.50	36.10
	200	72.85	73.85	74.10	74.45	75.10
(c)	300	109.85	111.35	110.45	111.30	110.65
	400	144.05	147.90	149.85	148.10	148.25
	500	180.15	184.85	182.75	183.95	183.80

Table 2: The average number of extra Engquist iterations needed in the Branch and Bound . (a) 1 – 100; (b) 1 – 1000; (c) 1 – 10000.

		Arc Density				
	n	20%	40%	60%	80%	100%
	100	1.10	1.30	2.10	1.10	2.10
	200	0.90	1.60	3.90	2.70	3.70
(a)	300	1.60	2.40	4.80	38.20	4.20
	400	3.70	4.60	5.40	3.70	4.00
	500	5.10	17.30	4.80	5.60	4.00
	100	0.70	2.40	1.50	1.10	1.50
	200	2.20	1.30	1.80	1.80	0.80
(b)	300	1.60	1.20	1.40	1.90	2.50
	400	1.70	2.90	1.90	1.40	1.30
	500	1.40	2.30	2.50	2.30	2.40
	100	1.00	1.60	1.60	1.80	0.90
	200	1.70	2.00	0.70	1.20	1.80
(c)	300	1.90	2.50	2.20	2.80	2.30
	400	1.50	2.40	3.20	3.30	1.80
	500	2.70	2.10	3.20	2.20	2.00

Table 3: The average number of cpu seconds to solve the initial AP relaxations. Costs uniformly distributed in the ranges (a) 1 – 100; (b) 1 – 1000; (c) 1 – 10000.

		Arc Density				
	n	20%	40%	60%	80%	100%
	100	0.17	0.26	0.38	0.40	0.49
	200	0.73	1.13	1.50	2.06	2.44
(a)	300	1.81	2.78	3.62	5.04	6.50
	400	3.26	5.21	7.32	9.92	12.37
	500	4.87	8.73	12.65	15.75	21.87
	100	0.17	0.25	0.38	0.45	0.52
	200	0.80	1.24	1.74	2.02	2.40
(b)	300	1.93	3.11	4.25	4.93	5.68
	400	3.74	6.05	7.64	10.27	10.32
	500	6.08	10.35	13.09	14.72	17.51
	100	0.16	0.26	0.36	0.47	0.55
	200	0.77	1.30	1.65	2.19	2.70
(c)	300	2.02	3.25	4.48	5.99	6.57
	400	3.52	6.97	8.63	10.35	12.43
	500	6.22	10.08	14.44	17.77	21.18

Table 4: The average number of matchings in the solutions of the initial AP relaxations. Costs uniformly distributed in the ranges (a) 1 – 100; (b) 1 – 1000; (c) 1 – 10000.

		Arc Density				
	n	20%	40%	60%	80%	100%
	100	49.5	49.5	49.4	49.6	49.5
	200	99.6	99.6	99.4	99.3	99.1
(a)	300	149.5	149.3	149.4	148.9	148.9
	400	199.1	199.2	199.3	199.0	199.2
	500	249.3	249.3	248.9	248.9	249.0
	100	49.7	49.5	49.5	49.5	49.5
	200	99.3	99.7	99.5	99.5	99.7
(b)	300	149.5	149.5	149.5	149.5	149.3
	400	199.5	199.4	199.5	199.6	199.6
	500	249.6	249.3	249.5	249.3	249.3
	100	49.6	49.5	49.5	49.5	49.6
	200	99.3	99.4	99.7	99.5	99.4
(c)	300	149.3	149.3	149.4	149.3	149.4
	400	199.6	199.4	199.3	199.3	199.5
	500	249.3	249.4	249.0	249.3	249.3

Table 5: The average number of cpu seconds required to complete the solutions of the matching problems. Costs uniformly distributed in the ranges (a) 1 – 100; (b) 1 – 1000; (c) 1 – 10000.

		Arc Density				
	n	20%	40%	60%	80%	100%
	100	0.03	0.05	0.12	0.08	0.14
	200	0.10	0.23	0.65	0.59	0.74
(a)	300	0.32	0.71	1.62	17.97	1.84
	400	1.29	1.89	2.85	2.33	2.61
	500	2.25	11.94	3.48	4.54	4.25
	100	0.03	0.10	0.09	0.07	0.12
	200	0.23	0.18	0.34	0.38	0.21
(b)	300	0.32	0.36	0.53	0.82	1.41
	400	0.59	1.49	1.27	1.20	1.40
	500	0.74	1.86	2.68	2.79	3.26
	100	0.03	0.07	0.09	0.11	0.07
	200	0.16	0.29	0.14	0.27	0.47
(c)	300	0.35	0.72	0.88	1.38	1.27
	400	0.51	1.26	2.01	2.81	1.80
	500	1.34	1.66	3.41	2.76	3.06

The authors would like to thank the Transport Department of The Brisbane City Council for their support in bringing this work to fruition.

References

[1] E. Balas and P. Toth, *Branch and bound methods* in *The traveling salesman problem, A guided tour of combinatorial optimization, Eds E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys*, John Wiley & Sons, (1985) 361–401.

[2] U. Derigs, *A shortest path method for solving minimal perfect matching problems*, Networks, 11, (1981) 379–390.

[3] U. Derigs and A. Metz, *On the use of optimal fractional matchings for solving the (integer) matching problem*, Computing, 36, (1986) 263–270.

[4] M. D. Devine and F. Glover, *Computational study of the symmetric assignment problem*, presented at the 41st National ORSA Meeting, New Orleans, 1972.

[5] J. Edmonds, *Maximum matching and a polyhedron with 0.1 vertices*, J. Res. Natl. Bur. Stand., 69B, (1965) 125–130.

[6] M. Engquist, *A successive shortest path algorithm for the assignment problem*, INFOR, 20, (1982) 370–384.

[7] M. Grötchel and O. Holland, *Solving matching problems with Linear programming* , Mathematical Programming, 33, (1985) 243–259.

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

[9] R. G. Parker and R. L. Rardin, *Discrete Optimization*, Academic Press, San Diego, (1988).