European Journal of Operational Research 94 (1996) 517–526

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

A tree search algorithm for the crew scheduling problem

J.E. Beasley *, B. Cao

The Management School, Imperial College, London SW7 2AZ, United Kingdom

Received 7 October 1993; revised 7 March 1995

Abstract

In this paper we consider the crew scheduling program, that is the problem of assigning K crews to N tasks with fixed start and finish times such that each crew does not exceed a limit on the total time it can spend working.

A zero–one integer linear programming formulation of the problem is given, which is then relaxed in a lagrangean way to provide a lower bound which is improved by subgradient optimisation. Finally a tree search procedure incorporating this lower bound is presented to solve the problem to optimality.

Computational results are given for a number of randomly generated test problems involving between 50 and 500 tasks.

Keywords: Crew scheduling; Lagrangean relaxation; Optimisation

1. Introduction

Crew scheduling problems occur frequently in the real world, such as those found in the airline and mass transit (typically bus) industries. Although papers can be found in the literature dealing with ''crew scheduling'' problems they often present algorithms tailored to the constraints and conditions prevailing in a particular industry.

In our view, however, a generic industry independent ''crew scheduling'' problem is important in the sense that it conveys a clear view of the underlying structure of industry dependent crew scheduling problems. On top of this generic problem any particular industrial application may impose a number of additional constraints/conditions.

In this paper therefore we, following the approach of earlier workers [8,11,14], adopt the approach of defining the generic crew scheduling problem. We then present a zero–one integer linear programming formulation for this generic crew scheduling problem, which is relaxed to provide a lower bound for imbedding in a tree search procedure to solve the problem optimally.

1.1. The generic problem

The generic crew scheduling problem (henceforth the CSP) involves a number of distinct elements:

1. N tasks which have to be performed by K ($\leq N$) crews (each crew being identical and located at the same depot from which they start their work ''day'' and at which they end their work ''day'')
2. each task i has associated with it:
   - d_i the cost of performing task i,
   - a fixed start time s_i together with a fixed finish time f_i ($> s_i$, implying a fixed duration of $f_i - s_i$ for the task),

* Corresponding author.

- a travel time and cost (b_i and c_{0i} respectively) which are the time/cost incurred when a crew travels from the depot to task i (and vice-versa).

Without loss of generality we shall assume that the tasks have been numbered in ascending start time (s_i) order.

3. for any two tasks i and j ($j > i$) there is a *transition arc* of cost c_{ij} if it is possible for the same crew to perform task i and then to perform task j,
4. we define a crew path (or more simply a path) to be the (non-empty) set of tasks (arranged in ascending order) that will be performed by the same crew,
5. the problem therefore is to find crew paths (of minimum total cost) such that each task is performed exactly once and the total working time involved in each path (where by working time we mean the elapsed time between departure from the depot and arrival back at the depot) does not exceed the available working time T.

Conceptually the problem is best viewed graphically as in Fig. 1 (e.g. see [4,5,8,11,12,14]). In that figure we have plotted time along the horizontal axis. The tasks that have to be performed are the vertices (with "length" equal to their duration). Vertices are linked to each other by transition arcs. The depot is shown twice, as 0 to represent the start of a crew path and as $N+1$ to represent the end of a crew path. Because of the time dimension Fig. 1 is acyclic, i.e. there are no cycles in the graph.

The problem therefore is to find K vertex (task) disjoint paths from 0 to $N+1$ in Fig. 1 such that:

1. all tasks are on a path,
2. the working time involved in each path does not exceed T,
3. the total cost of the paths is a minimum.

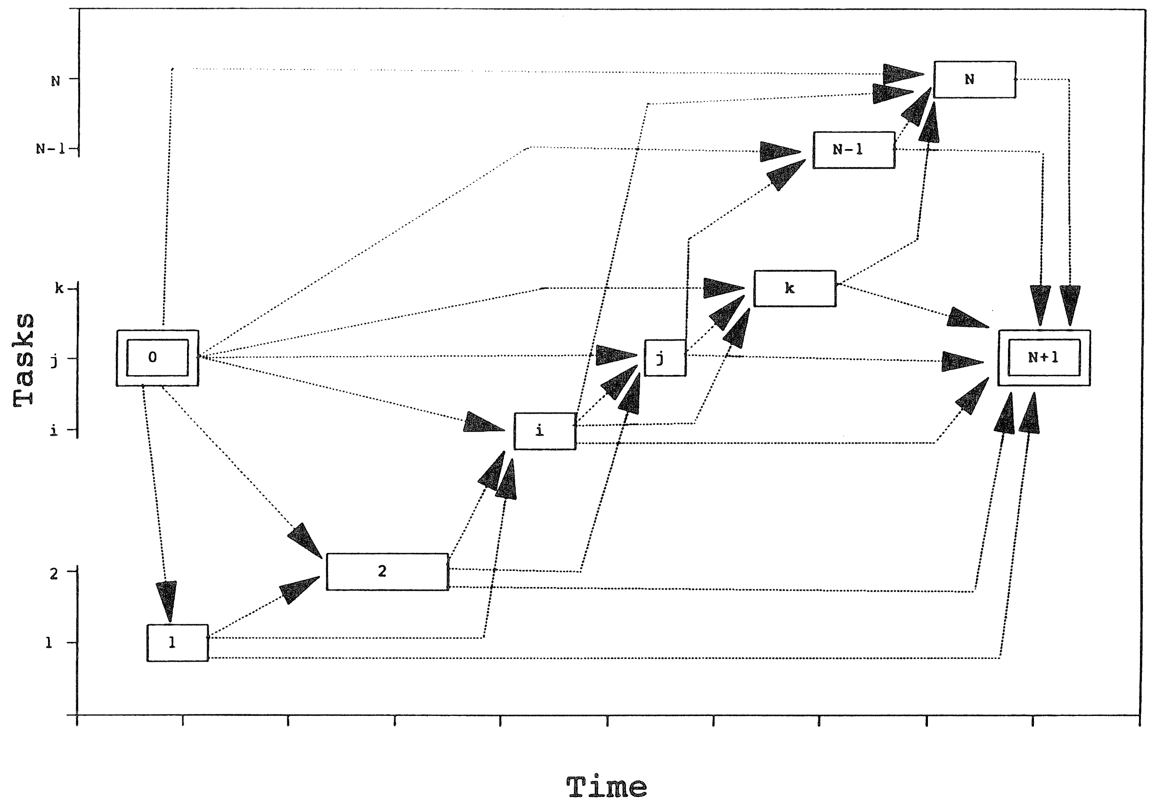

Fig. 1. Note: Although 0 and $N+1$ are shown above there are no fixed times associated with them (unlike tasks).

Since each task is performed exactly once the total cost associated with performing the tasks $\sum_{i=1}^{N} d_i$ is the same for all feasible solutions and hence can be neglected. The only cost term therefore comes from the costs associated with the transition arcs.

Transition arcs play a key role in our formulation of the CSP. Conceptually it is helpful to think of Fig. 1 as initially including the complete set of transition arcs $[(i,j) | \forall i, \forall j > i, f_i \leq s_j]$. Transition arcs that are infeasible are then deleted from that figure. For example any transition arc (i, j) for which $b_i + (f_j - s_i) + b_j > T$ can be deleted.

In general the transition arcs that need to be deleted are deduced from some underlying knowledge of the situation being considered. For example we may require a minimum rest period τ for the crew between two successive tasks (i.e. we require $(s_j - f_i) > \tau$). Also if two tasks i and j are being performed at different geographic locations then it may simply not be physically possible for the crew to travel from the "i location" to the "j location" in the time $(s_j - f_i)$ available.

2. Literature survey

A number of the crew scheduling papers in the literature can be classified in general terms as:

1. generate and cost an initial set (S) of candidate feasible crew schedules
2. formulate the problem of selecting from S a minimum cost subset that performs all tasks *either* as a set covering problem (SCP) *or* as a set partitioning problem (SPP)
3. solve the SCP/SPP using an appropriate algorithm (e.g. linear programming (LP) based branch and bound).

Some papers use column generation techniques to expand the initial set S of candidate crew schedules (typically based upon a shortest path calculation utilising dual information from the solution to the LP relaxation of the problem).

A number of additional constraints may also be added to this SCP/SPP formulation of the problem (e.g. crew base constraints in airline crew scheduling).

In order to structure our literature survey we consider airline crew scheduling, mass transit crew scheduling and generic crew scheduling separately.

2.1. Airline crew scheduling

Graves, McBride, Gershkoff, Anderson and Mahidhara [22] present an algorithm that follows the general approach given above (SPP, no column generation) with the exception that the elastic SPP (which allows constraints to be violated provided a penalty cost is incurred) is used. Crew base constraints are also considered. Limited computational results are reported for problems involving up to 1716 tasks.

Hoffman and Padberg [23] present an algorithm that follows the general approach given above (SPP, no column generation). The SPP (with additional crew base constraints if necessary) is solved by a branch and cut algorithm (which involves solving the LP relaxation of the problem and incorporating cuts derived from polyhedral considerations). Computational results are given for a large number of real-world problems involving up to 825 tasks.

Gershkoff [21] discusses crew scheduling at American Airlines. They use a heuristic procedure for the SPP formulation of the problem (based upon the work of Rubin [29]) whereby from an initial crew schedule a (small) number of crew paths are chosen and the subproblem (set of tasks) corresponding to the selected crew paths is solved to optimality (via explicit enumeration of all possible feasible crew schedules and LP-based branch and bound). The initial crew schedule can then be updated using this optimal solution and the process repeated. Anbil, Gelman, Patty and Tanga [1] provide an update on the crew scheduling work at American Airlines.

Lavoie, Minoux and Odier [24] present an algorithm that follows the general approach given above (SCP, column generation). Column generation is based upon a shortest path calculation on an acyclic feasibility graph. Once the LP-based column generation procedure fails to generate any attractive candidate schedules it is suggested that the final SCP is optimally resolved via branch and bound. Computational results are given for a number of real-world problems involving up to 329 tasks.

Crainic and Rousseau [16] present a heuristic algorithm that follows the general approach given above (SCP, column generation). Column generation is based upon a heuristic procedure. Once the column generation procedure fails to generate any attractive candidate schedules the final SCP is solved using a heuristic algorithm. Computational results are given for three real-world problems involving between 34 and 120 tasks.

Ball and Roberts [5] present a heuristic algorithm based upon finding an initial solution by constructing feasible crew paths in Fig. 1 guided by the solution to a sequence of minimum cost matching problems. This initial solution is then improved by looking at the potential savings resulting from rearranging the tasks in all pairs of crew paths. A minimum cost matching problem is solved to determine which rearrangements to make. Computational results are given for four real-world problems involving between 264 and 1058 tasks.

For other work on airline crew scheduling see [2,3,25,26,29].

2.2. *Mass transit crew scheduling*

Sousa [31] presents an algorithm involving both man–machine interaction and a simple heuristic. Computational results are given for a real-world problem.

Desrochers and Soumis [17] present an algorithm that follows the general approach given above (SCP, column generation). Column generation is based upon a resource constrained shortest path calculation on an acyclic graph. Computational results are given for two real-world problems involving 167 and 235 tasks.

Smith [30] presents an algorithm that follows the general approach given above (SCP, no column generation).

Falkner and Ryan [18] present an algorithm that follows the general approach given above (SPP, no column generation). Computational results are given for two real-world problems.

Martello and Toth [27] present a heuristic algorithm for minimising the number of crews used based upon a zero–one formulation of the problem. Their algorithm is a greedy procedure that builds upon a partial solution, guided by the solution to a number of matching problems. Computational results are given for two real-world problems.

Ball, Bodin and Dial [4] present an algorithm that is similar to the algorithm presented in [5] for airline crew scheduling. Computational results are given for a real-world problem involving 1602 tasks.

For other work on mass transit crew scheduling see [9,10,13,28,32].

2.3. *Generic crew scheduling*

For the generic crew scheduling problem defined above Beasley and Cao [8] present an algorithm based upon dynamic programming together with computational results for randomly generated problems involving up to 500 tasks.

Cao [11] considered both the generic problem and various extensions to that problem (involving rest periods in a crew path, multiple depots, multiple vehicle types and task start time windows). His algorithms used a variety of methods – heuristics, lagrangean relaxation, lagrangean decomposition and dynamic programming.

Cheddad [14] proposed various linear programming and lagrangean relaxation based algorithms. Extensions to the generic problem were also considered.

3. Formulation and lower bound

In this section we illustrate how the CSP, as defined above, can be formulated as a zero–one integer linear programming problem. We then consider a lagrangean relaxation of this formulation which provides a lower bound for the CSP.

3.1. *CSP formulation*

Recall here that the CSP is the problem of finding K vertex (task) disjoint paths from 0 to $N+1$ in Fig. 1 such that:

1. all tasks are on a path,
2. the working time involved in each path does not exceed T,
3. the total cost of the paths is a minimum.

Define:

$$x_{ij} = \begin{cases} 1 & \text{if transition arc}(i,j)\text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

then the CSP is:

minimise:

$$\sum_{i,j} c_{ij} x_{ij} \tag{1}$$

subject to:

$$\sum_{k} x_{jk} = \sum_{i} x_{ij} \quad j = 1,\dots,N \tag{2}$$

$$\sum_{j} x_{ij} = 1 \quad i = 1,\dots,N \tag{3}$$

$$\sum_{j} x_{0j} = K \tag{4}$$

time limit constraints (5)

$$x_{ij} \in (0,1) \quad \forall\, i,j \tag{6}$$

Note here that although Eq. (5), representing the requirement that the working time involved in each path does not exceed T, could be written in a mathematical way we prefer it as presented above for simplicity.

In this formulation of the CSP Eq. (2) specifies that the number of arcs leaving a task is equal to the number entering the task. Eq. (3) specifies that one arc leaves each task (i.e. that each task is on exactly one path so that the paths are task disjoint). Eq. (4) specifies that K arcs originating at 0 are used (or to phrase it differently, K tasks are chosen to be the first task on some path). Eq. (6) specifies that each arc can be used at most once (i.e. that the paths are arc disjoint).

3.2. Lagrangean relaxation

Introducing multipliers (u_i) and relaxing Eq. (3) in the above CSP formulation in a lagrangean fashion (see [7,19,20] for a detailed description of lagrangean relaxation) we get:

minimise:

$$\sum_{i,j} c_{ij} x_{ij} + \sum_{i=1}^{N} u_i \left(1 - \sum_{j} x_{ij}\right) \tag{7}$$

subject to:

$$\sum_{k} x_{jk} = \sum_{i} x_{ij} \quad j = 1,\dots,N \tag{8}$$

$$\sum_{j} x_{0j} = K \tag{9}$$

time limit constraints (10)

$$x_{ij} \in (0,1) \quad \forall\, i,j \tag{11}$$

Eqs. (7)–(11) imply that:
1. there are to be K time limit constrained arc disjoint paths, each of which has a different first task,
2. the objective function is to minimise the total (lagrangean) cost of these K paths.

It is not easy to solve this problem, principally because of the requirement that the paths are arc disjoint ($x_{ij} \in (0,1) \forall i,j$, Eq. (11)). Hence to make the problem easier to solve we further relax Eq. (11) to:

$$x_{0j} \in (0,1) \quad j = 1,\dots,N \tag{12}$$

$$x_{ij} \geq 0 \text{ integer} \quad \forall\, i \neq 0, j \tag{13}$$

to give the complete lagrangean relaxation as:

minimise:

$$\sum_{i,j} c_{ij} x_{ij} + \sum_{i=1}^{N} u_i \left(1 - \sum_{j} x_{ij}\right) \tag{14}$$

subject to:

$$\sum_{k} x_{jk} = \sum_{i} x_{ij} \quad j = 1,\dots,N \tag{15}$$

$$\sum_{j} x_{0j} = K \tag{16}$$

time limit constraints (17)

$$x_{0j} \in (0,1) \quad j = 1,\dots,N \tag{18}$$

$$x_{ij} \geq 0 \text{ integer} \quad \forall\, i \neq 0, j \tag{19}$$

Eqs. (14)–(19) represent the problem of finding the K least (lagrangean) cost time limit constrained paths, each with a different first task. This problem can be easily solved as follows:

1. for each task j in turn:
   1.1. temporarily remove from Fig. 1 any tasks k which, from the working time constraint viewpoint, cannot be performed on the same crew path as j (i.e. any k for which $b_j + (f_k - s_j) + b_k > T$)
   1.2. find the least (lagrangean) cost path that goes directly from 0 to j and ends at $N+1$ in the resulting graph (this least cost path

calculation is easily accomplished (e.g. see [15]) since the graph (Fig. 1) is acyclic)
2. the K smallest of these N least (lagrangean) cost paths then constitute the solution to the lagrangean relaxation (Eqs. (14)–(19)) and provide a lower bound on the optimal solution to the original CSP.

3.3. Subgradient optimisation

The multipliers (u_i) in the above lagrangean relaxation are updated with the subgradient optimisation method (see [7,19,20] for a detailed description of subgradient optimisation). Given initial values (u_i^0) for the multipliers a sequence (u_i^n) can be produced by:

$$u_i^{n+1} = u_i^n + t_n\left(1 - \sum_j x_{ij}^n\right) \quad i = 1,\dots,N \tag{20}$$

where (x_{ij}^n) is the optimal solution of the lagrangean relaxation at the n^{th} iteration (of value $Z(u^n)$) and t_n is a positive scalar step size given by:

$$t_n = \lambda(Z^* - Z(u^n)) / \sum_{i=1}^{N}\left(1 - \sum_j x_{ij}^n\right)^2 \tag{21}$$

where Z^* is set to two times the best lower bound found. This is done because we did not, in many of the example test problems solved, have any initial feasible solution. λ is a scalar parameter which is set to 2 initially and divided by 2 after 30 iterations if there has been no increase in the best value of the lower bound $(Z(u^n))$ found so far.

If at any iteration the solution to the lagrangean relaxation is feasible for the original CSP then it is also optimal as we have relaxed equality constraints. Otherwise the subgradient optimisation procedure was artificially terminated when λ fell below 10^{-6} (in which case no solution, neither feasible nor optimal, will have been obtained).

4. Tree search procedure

As mentioned above the subgradient optimisation procedure may terminate without having found the optimal solution to the original CSP. In order to find the optimal solution we need to resort to a tree search (branch and bound) procedure. In this section we describe the procedure that we adopted.

4.1. Initial tree node

We first calculated a lower bound on the problem using lagrangean relaxation and subgradient optimisation as detailed above.

4.2. Other tree nodes

4.2.1. Forward branching – selection of a branching node

In forward branching we choose the tree node with the minimum lower bound value and not having been either branched or fathomed to branch on. The logic for this was essentially twofold:
1. as we did not have an upper bound for most of the problems we solved (and hence no means of pruning/curtailing the tree from lower bound considerations) the alternative of a depth-first tree search procedure may have led to excessively large trees,
2. early computational experience indicated that the maximum lower bound achieved by lagrangean relaxation and subgradient optimisation was very close to the optimal CSP solution.

Hence it seemed best to branch from the tree node with the minimum lower bound value as that tree node was most likely to lead to the optimal solution.

4.2.2. Forward branching – selection of a branching variable

In forward branching from a tree node we branched by examining the solution ((X_{ij}), say) associated with the best lower bound at that tree node.

Letting (D_i) represent the task outdegrees (i.e. $D_i = \sum_j X_{ij} i = 1,\dots,N$) we choose a branching variable in the following way:
1. let p be the task such that $D_p = \max[D_i \mid D_i \neq 1, i = 1,\dots,N]$ (ties broken arbitrarily),
2. let (p, j) be the transition arc for which $X_{pj} = \max[X_{pk} \mid k = 1,\dots,N]$ (ties broken arbitrarily),

then we branched by setting $X_{pj} = 1$. Computationally this can be achieved by amalgamating tasks p and j into a single "pseudo-task" in an obvious fashion.

4.2.3. Lower bound

At each tree node we performed the same lower bound calculation as at the initial tree node except that λ was set to 0.005 initially and halved after five iterations had occurred without any increase in the maximum lower bound found. In addition, we did at most 150 subgradient iterations at a tree node.

The initial set of multipliers (u_i) at each tree node were the set associated with the best lower bound found at the predecessor tree node. Z^* was set to the value of a feasible solution when a feasible solution was first found and updated whenever a better feasible solution was found.

4.2.4. Backtracking

We can backtrack in the tree when a feasible solution is obtained or when $Z(u^n) > Z^*$.

5. Computational results

5.1. Test problem generation

The tree search algorithm described above was programmed in FORTRAN and run on a Silicon Graphics Iris Indigo workstation for a number of randomly generated test problems involving from 50 to 500 tasks. These test problems were the same test problems as were solved in our previous work [8] and were generated in the following manner:

1. set the time limit $T = 480$ (minutes),
2. set the start time s_i of each task i to be an integer randomly generated from [1,1440] and set $b_i = d_i = c_{0i} = c_{i,N+1} = 0$
3. the finish time f_i of task i was defined by producing an integer v randomly generated from

Table 1
Computational results – from 50 to 250 tasks

Number of tasks	Number of transition arcs	Number of crews	Initial tree node			Tree search		Optimal value
			Number of iterations	Duality gap (%)	Time (seconds)	Number of tree nodes	Time (seconds)	
50	173	31	249	-	1	-	-	1872
		30	357	-	1	-	-	2092
		29	795	0.000028	2	2	1	2399
		28	251	-	1	-	-	2706
		27	512	-	2	-	-	3139
100	715	48	430	-	7	-	-	3905
		47	490	-	8	-	-	4107
		46	558	-	9	-	-	4310
		45	347	-	6	-	-	4514
		44	495	-	8	-	-	4812
150	1355	73	461	-	18	-	-	5347
		72	893	0.009059	34	2	7	5551
		71	544	-	21	-	-	5754
		70	1279	0.166753	48	31	96	5999
		69	1040	0.406412	39	63	111	6275
200	2543	97	504	-	43	-	-	6288
		96	575	-	49	-	-	6430
		95	505	-	43	-	-	6583
		94	608	-	51	-	-	6747
		93	720	-	61	-	-	6914
250	4152	112	584	-	100	-	-	7707
		111	633	-	108	-	-	7863
		110	639	-	109	-	-	8023
		109	1686	0.110470	287	24	469	8212
		108	1199	-	205	-	-	8406

$[1, 240 - 15]$ and setting $f_i = s_i + 15 + v$ (hence ensuring that task durations were neither too long nor too short),

4. once all the task start and finish times had been generated the tasks were relabelled in increasing start time order and for all pairs of tasks (i,j) for which a transition arc potentially exists (i.e. $f_i \leq s_j$ and $b_i + (f_j - s_i) + b_j \leq T$):
   4.1. let v be an integer randomly generated from $[1, 240]$,
   4.2. the transition arc exists if $f_i + v \leq s_j$, else not,
   4.3. if the transition arc exists it is of cost $(1 + r)(s_j - f_i)$, rounded up to the nearest integer, where r is a real random number sampled from $U(0,1)$.

In order to decide the number of crews K to use we applied a simple crew path construction heuristic, namely:

1. let P_k be the set of tasks in crew path k, L be the number of crew paths,
2. set $L = 1$, $P_1 = [1]$ and recall that the tasks are labelled in increasing start time order,
3. consider each task i $(i = 2, \dots, N)$ in turn:
   3.1. if it is feasible to add i to the end of the crew path in P_k (for some $k \leq L$) then do so, ties broken by adding i to the end of the crew path that incurs the minimum transition cost from the task at the end of the crew path to i,
   3.2. if it is not possible to add i to any crew path then set $L = L + 1$ and $P_L = [i]$.

After all tasks have been considered we will have a feasible solution (of a certain cost) involving L crews.

In the computational results presented below we, for each of the test problems, first found a value for L using the above heuristic and then solved the

Table 2
Computational results - from 300 to 500 tasks

Number of tasks	Number of transition arcs	Number of crews	Initial tree node			Tree search		Optimal value
			Number of iterations	Duality gap (%)	Time (seconds)	Number of tree nodes	Time (seconds)	
300	6108	133	785	-	237	-	-	9026
		132	2225	0.000114	672	51	644	9200
		131	692	-	209	-	-	9378
		130	860	-	260	-	-	9580
		129						infeasible
350	7882	148	1438	0.004873	622	29	831	10378
		147	1123	0.000038	488	22	508	10525
		146	992	0.000049	430	6	63	10677
		145	1520	0.000159	659	4	112	10833
		144	1059	0.000010	458	2	1	10991
400	10760	163	730	-	489	-	-	11696
		162	752	-	504	-	-	11848
		161	2315	0.004422	1561	16	884	12006
		160	792	-	532	-	-	12163
		159	1970	0.084993	1316	209	15860	12341
450	13510	186	2034	0.025153	1918	126	13739	12232
		185	1217	0.000055	1149	6	218	12357
		184	1841	0.033843	1739	603	39661	12497
		183	1810	0.031439	1714	95	8885	12639
		182	1736	0.034019	1631	127	16204	12785
500	16695	208	1358	0.006114	2463	12	2540	12772
		207	685	-	1251	-	-	12899
		206	925	-	1683	-	-	13032
		205	1024	0.015231	1880	25	4893	13169
		204	856	-	1545	-	-	13302

problem using $K = L + 2$, $K = L + 1$, $K = L$, $K = L - 1$ and $K = L - 2$ i.e. each test problem was solved five times, each time with a different value for the number of crews K. This implies that, except for $K = L$, we do not have an initial feasible solution, providing an upper bound on the problem, available. Indeed, for $K = L - 1$ and $K = L - 2$, there may not be a feasible solution (i.e. the problem may be infeasible, although this only happened for one of our test problems – see below).

Note here that all of these test problems are now publically available via e-mail from OR-Library [6].

5.2. Results

Tables 1 and 2 give the results for the algorithm presented in this paper when applied to these test problems. In those tables we give, for each problem:

1. the number of tasks, the number of transition arcs (excluding arcs to/from the depot) and the number of crews,
2. the number of subgradient iterations at the initial tree node,
3. the percentage duality gap between the best lower bound at the initial tree node and the optimal solution (as measured by 100(optimal value - best lower bound)/(optimal value)),
4. the computation time in Silicon Graphics Iris Indigo cpu seconds for the initial tree node,
5. the number of tree search nodes,
6. the total computation time in Silicon Graphics Iris Indigo cpu seconds for the tree search nodes (note here that this means that the total algorithm time is found by adding the time for the initial tree node to the time for the tree search nodes),
7. the value of the optimal solution.

Examining these tables we find that:

- most of the test problems were solved without branching being necessary, indeed only 19 of the 49 test problems required branching
- for those problems requiring branching the number of tree nodes was reasonable in most cases
- over all 49 test problems the average duality gap was 0.018919% and the average (total) computer time was 2701 cpu seconds.

Comparing these results with the results obtained in our previous work [8] (in which the average duality gap was 0.019558% and the average (total) computer time was 13685 cpu seconds) it is clear that the algorithm presented in this paper is (computationally) superior to the algorithm presented in [8].

6. Conclusions

In this paper we presented an algorithm for the crew scheduling problem based upon lagrangean relaxation of a zero–one integer linear programming formulation of the problem, together with subgradient optimisation and a tree search procedure for optimally resolving the problem.

Computational results indicated that the algorithm was able to solve relatively large problems.

References

[1] R. Anbil, E. Gelman, B. Patty and R. Tanga, Recent advances in crew-pairing optimization at American Airlines. *Interfaces* 21(1) (1991) 62–74.

[2] E.K. Baker, L.D. Bodin, W.F. Finnegan and R.J. Ponder, Efficient heuristic solutions to an airline crew scheduling problem. *AIIE Transactions* 11 (1979) 79–85.

[3] E. Baker and M. Fisher, Computational results for very large air crew scheduling problems. *OMEGA* 9 (1981) 613–618.

[4] M. Ball, L. Bodin and R. Dial, A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Science* 17 (1983) 4–31.

[5] M. Ball and A. Roberts, A graph partitioning approach to airline crew scheduling. *Transportation Science* 19 (1985) 107–126.

[6] J.E. Beasley, OR–Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41 (1990) 1069–1072.

[7] J.E. Beasley, Lagrangean relaxation. In *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed.), Blackwell Scientific Publications, Oxford (1993).

[8] J.E. Beasley and B. Cao, An algorithm for the crew scheduling problem. Working paper, The Management School, Imperial College, London SW7 2AZ (1993).

[9] L.D. Bodin and B.L. Golden, Classification in vehicle routing and scheduling. *Networks* 11 (1981) 97–108.

[10] L. Bodin, B. Golden, A. Assad and M. Ball, Routing and scheduling of vehicles and crews – the state of the art. *Computers & Operations Research* 10 (1983) 63–211.

[11] B. Cao, Algorithms for crew scheduling problems. PhD thesis, Imperial College, London (1992).

[12] P. Carraresi and G. Gallo, Network models for vehicle and crew scheduling. *European Journal of Operational Research* 16 (1984) 139–151.

[13] P. Carraresi, G. Gallo and J.-M. Rousseau, Relaxation approaches to large scale bus driver scheduling problems. *Transportation Research* 16B (1982) 383–397.

[14] H. Cheddad, Algorithms for crew scheduling problems. PhD thesis, Imperial College, London (1987).

[15] N. Christofides, Graph theory: an algorithmic approach. Academic Press, London (1975).

[16] T.G. Crainic and J.-M. Rousseau, The column generation principle and the airline crew scheduling problem. *INFOR* 25 (1987) 136–151.

[17] M. Desrochers and F. Soumis, A column generation approach to the urban transit crew scheduling problem. *Transportation Science* 23 (1989) 1–13.

[18] J.C. Falkner and D.M. Ryan, A bus crew scheduling system using a set partitioning model. *Asia-Pacific Journal of Operational Research* 4 (1987) 39–56.

[19] M.L. Fisher, The lagrangian relaxation method for solving integer programming problems. *Management Science* 27 (1981) 1–18.

[20] M.L. Fisher, An applications oriented guide to lagrangian relaxation. *Interfaces* 15(2) (1985) 10–21.

[21] I. Gershkoff, Optimizing flight crew schedules. *Interfaces* 19(4) (1989) 29–43.

[22] G.W Graves, R.D. McBride, I. Gershkoff, D. Anderson and D. Mahidhara, Flight crew scheduling. *Management Science* 39 (1993) 736–745.

[23] K.L. Hoffman and M. Padberg, Solving airline crew scheduling problems by branch-and-cut. *Management Science* 39 (1993) 657–682.

[24] S. Lavoie, M. Minoux and E. Odier, A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research* 35 (1988) 45–58.

[25] R.E. Marsten, M.R. Muller and C.L. Killion, Crew planning at Flying Tiger: a successful application of integer programming. *Management Science* 25 (1979) 1175–1183.

[26] R.E. Marsten and F. Shepardson, Exact solution of crew scheduling problems using the set partitioning model: recent successful applications. *Networks* 11 (1981) 165–177.

[27] S. Martello and P. Toth, A heuristic approach to the bus driver scheduling problem. *European Journal of Operational Research* 24 (1986) 106–117.

[28] J.-M. Rousseau, *Computer Scheduling of Public Transport 2*. North-Holland, Amsterdam (1985).

[29] J. Rubin, A technique for the solution of massive set covering problems with application to airline crew scheduling. *Transportation Science* 7 (1973) 34–48.

[30] B.M. Smith, IMPACS – a bus crew scheduling system using integer programming. *Mathematical Programming* 42 (1988) 181–187.

[31] J.F. de Sousa, A computer based interactive approach to crew scheduling. *European Journal of Operational Research* 55 (1991) 382–393.

[32] A. Wren (ed.), *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam (1981).