The Application of Semidefinite Programming
to the
Aircrew Scheduling Problem

Leander Quiring

April 22, 2008

Abstract

It is said that the world is shrinking every day. Passenger air travel, along with electronic communication, is undoubtedly one of the key factors responsible for this shrinking. To allow air traffic in the volumes we see today, quite a few extremely difficult problems had to be solved in the fields of structural engineering, aerodynamics, even marketing.

In this paper, we examine one such problem, the aircrew scheduling problem, and how it is has been solved in industry. We then examine a Semi-Definite Programming (SDP) relaxation for the two phases of this problem and test it on three test cases. The results of these test cases are promising, and suggest that further research in this area might be very rewarding. We also suggest several practical areas in which this research might be directed and where SDP could be used in the airline industry.

INTRODUCTION 1

AIRCREW SCHEDULING PROBLEM 1

LITERATURE REVIEW 2

SET PARTITIONING 3

General Formulation 3

Side Constraints 4

HEURISTICS 5

Gershkoff's Heuristic 5

Rubin's Heuristic 6

SEMIDEFINITE PROGRAMMING RELAXATION 6

Formulation 6

Numerical Results 7

PRACTICAL APPLICATIONS 9

CONCLUSIONS 10

REFERENCES 11

APPENDIX A: SMALL EXAMPLE CODE 12

SDP Relaxation 12

Exact Formulation 13

APPENDIX B: MATLAB FUNCTION CODE 14

Introduction

It is said that the world is shrinking every day. Passenger air travel, along with electronic communication, is undoubtedly one of the key factors responsible for this shrinking. To allow air traffic in the volumes we see today, quite a few extremely difficult problems had to be solved in the fields of structural engineering, aerodynamics, even marketing. In recent years, operations research (OR) has become another asset for airlines to keep a competitive edge.

Despite its growing importance, however, anecdotal evidence suggests that the airline industry uses near- but non-optimal solutions to their daily problems. This is, in large part, due to the inherent intractability of a key problem facing airlines: aircrew scheduling. The most prevalent model for aircrew schedule requires a two-stage set-partitioning problem (SPP). It is well known that each stage of problem is NP-complete, so either efficient and effective heuristics must be used or immense computing power must be thrown at it.

Aircrew Scheduling Problem

Let us consider the first stage of an aircrew scheduling problem (ASP), whose stages will be discussed in more detail later, where we wish to divide up the set of possible flights between a set of locations over the course of a period of time into tours of duty (ToDs). These will later be assigned to particular crewmembers.

Consider the case with flights based out of Toronto, flying to and from Ottawa and Montreal. A typical morning might involve the following schedule:

	Flight	Departure Time
A-B	Toronto-Ottawa	8 AM
A-D	Toronto-Montreal	8 AM
D-E	Montreal-Ottawa	9 AM
E-F	Montreal-Toronto	9 AM
B-G	Ottawa-Montreal	9 AM
B-F	Ottawa-Toronto	9 AM
C-E	Toronto-Ottawa	9 AM
F-I	Toronto-Montreal	10 AM
E-H	Ottawa-Toronto	10 AM
G-H	Montreal-Toronto	10 AM
I-J	Montreal-Toronto	11 AM

Fig 1. Small Flight Schedule in Table Form

If we wish to pick an optimal set of TODs using the SPP, we must first enumerate all possible TODs and then select an optimal subset. To envision this, consider the beginning city in each flight as a node, with an arc to the ending city. This results in the following graph:

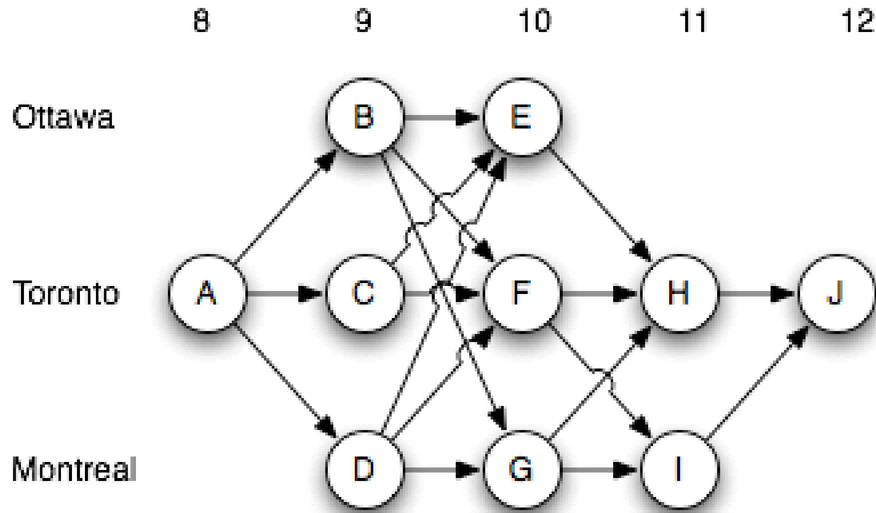

Fig 2. Small Flight Schedule in Graph Form

To enumerate all possible ToDs, we can consider all possible paths from the left-most node to the right-most one. Clearly, this problem will explode in size if we increase either the length of the schedule or the number of cities, or both. It is easy to see that, when using a real-life dataset, the number of variables and constraints simply becomes unmanageable. As we add more crew bases (though we consider only the single crew base case throughout this paper), we add additional complexity.

Literature Review

Because of the usefulness of the SPP to many real life applications, it has been studied extensively in the literature. Of particular interest to this study is the work by Mingozzi et al. (1998) on applying the SPP to aircrew scheduling. They discuss the difficulties of modelling real-life constraints, such as union rules. They then exposit a heuristic for calculating a lower bound of a general crew scheduling problem based on the dual of the linear programming relaxation to the SPP.

Wolkowicz and Zhao (1996) describe a method of using a semidefinite programming relaxation to provide information about the SPP. In particular, they provide a very compact formulation for both the relaxation and its dual by introducing a 'gangster operator,' a function that both guarantees a 0-1 solution and maintains solution feasibility. They then provide numerical and theoretical results that show that the semidefinite relaxation is, at worse, no worse than the linear programming relaxation, and is often superior.

In a very accessible paper, Butchers et al. (2001) discuss how Air New Zealand has profited from applying operations research in general and the SPP in particular to its crew roster software. They describe a tight alliance with local academia at the University of Auckland that has resulted in huge cost savings to the airline while providing a more desirable crew schedule to its personnel. The authors discuss both the size of typical

problems as well as real-life issues such as union rules, federal safety regulations and crew preferences and needs. They also describe how these issues affect the structure of the model for both domestic and international flights.

Ryan (1992), written by one of the co-authors of the previous paper, covers much of the same ground, but gives special emphasis on dealing with large-scale problems. He discusses how the relative sparseness of the constraint matrices in the set-partitioning formulation of the ASP can be used to find efficient solution methods. In fact, his methods provide relatively rapid (approximately three hours) runtimes to solve his large examples on what would now be considered to be ancient hardware.

Of course, finding an exact solution to the SPP is hardly the only solution methodology for the aircrew scheduling problem. As was alluded to previously, there are many heuristics in use by airlines today. One such example, used by the canonically successful American Airlines is described in Gershkoff (1989). Here, the author describes a heuristic algorithm to find approximate solutions to the first phase of the ASP by selecting the "best" ToDs for consideration, as opposed to an exhaustive search through all possible ToDs. The author then shows how this heuristic has been used in practice to reduce American Airlines' costs.

Rubin (1973) describes another heuristic that has been said to have met with wide adoption in the industry. This heuristic focuses on breaking the full set of ToDs into smaller, more manageable sub-problems. Additionally, it presents several methods of utilizing the structure inherent in crew scheduling problems to generate reasonable sub-problems by considering such factors as geographic separation of flights within a tour of duty.

Set Partitioning

General Formulation

Let us now examine how to model the aircrew scheduling problem as an SPP. As mentioned previously, the ASP has two stages. In the first, we must select from all physically possible sequences of flights a set of ToDs that ensures that each flight is in exactly one ToD. In practice, this is slightly modified to permit "passengering" (when an aircrew travels on a flight as a passenger to crew an upcoming flight departing from their destination). This is known as the Planning phase.

After this phase is completed, we now enter the second phase, known as Rostering. In this step, we wish to pair individual aircrews with ToDs. Typically, this is done separately for pilot crews and hospitality crews (stewards and stewardesses), as well as for domestic and international flights, because each of these groups tends to have substantially different regulations.

Now, let us recall the basic SPP formulation:

$$\min c^T x$$
$$s.t. Ax = 1$$
$$x \in \{0,1\}^n$$

In the Planning phase, each column of A, indexed by i up to n represents a possible tour of duty. Correspondingly, each row of A, indexed by j up to m represents a flight. Therefore, we have:

$$a_{ij} = 1 \text{ if flight } j \text{ is in ToD } i,$$
$$a_{ij} = 0 \text{ otherwise.}$$

We can add side constraints on x to accommodate constraints such as minimum flying time on a ToD, maximum breaks between flights, etc. We can accommodate "passengering" by changing some or all of the equality constraints to "greater than" constraints. Note, however, that this acts more like the set-covering problem. We can also encode the possibility of a crew staying in a particular city by including arcs such as B-E in figure 2. This graph results in (2^{11}-1) tours of duty. If we include passengering, it reaches (2^{18}-1) tours. The "-1" is included to eliminate the null ToD composed of all zeros. We have reduced the size of this toy problem from its full size for solution later on by considering temporal and geographic restrictions that make some ToDs unrealistic.

The SPP is well known to be NP-Complete due to its combinatorial nature. In this particular case, if we have m flights, we have the maximum tours of duty being:

$$n^{\max} = \sum_{i=1}^{m} \binom{m}{i}$$

This, of course, includes all possible combinations, regardless of their realism.

In the Rostering phase, each row of A is represents a selected tour of duty. The columns are divided amongst the workers (each worker will have one or more columns assigned to them. Therefore, we have:

$$a_{ij} = 1 \text{ if employee } i \text{ works ToD } j,$$
$$a_{ij} = 0 \text{ otherwise.}$$

As before, we can add side constraints such as minimum levels of crew preference, must satisfy all laws (not all legal requirements may be satisfied in the first phase as we may assign more than one ToD to a given employee).

Side Constraints

Now that we have gone over the basic problem, let us consider some side constraints that might be found in a typical application of the aircrew scheduling problem. One constraint, suggested by Rubin (1973) involves minimum and maximum worked hours in

the final plan, with the former likely based on union requirements and the latter based on government regulations. This constraint takes the form:

$$H_{\min} \le \sum_{j=1}^{n} h_j x_j \le H_{\max}$$

where H^{min} is the minimum required worked hours, H^{max} is the maximum permitted, and h_j is the hours required to work ToD j.

We may also have a maximum number of aircrews, based on available personnel.

$$\sum_{j=1}^{n} x_j \le P$$

where P is the number of crews available.

Our last example constraint eliminates the possibility of having a ToD that has many short flights. In this way, we spread long and short flights across crews. While it is possible to enforce this constraint during the generation of the matrix *A*, we show here a way to do it after the fact. Doing it this way allows us to perform "what-if" analysis without having to regenerate the main constraint matrix.

$$\sum_{i=1}^{m} a_{ij} x_j \le F, \forall j$$

where F is the maximum number of flights in a ToD.

It should be noted, of course, that there are reasonable constraints for the Rostering phase as well. For example, we may wish to limit the number of different tours of duty that a particular aircrew is assigned (in the case that aircrews are assigned multiple variables) to provide more or less variety for the crews by bounding their sum from either below or above.

Other previously discussed constraints may be best handled in the generation of the matrix *A*. For example, limiting the number of hours between active flights could be implemented by requiring that any column of A must have no more than a specified number of zeros between flights arriving and departing from a particular location. Clearly, this constraint would be more difficult to handle after A is generated.

Heuristics

Gershkoff's Heuristic

Because of their importance to the airline industry, we will now briefly examine some of the most successful heuristics. As mentioned previously, the heuristic in Gershkoff (1989) has met with significant success in American Airlines. The fundamental idea of this heuristic is to solve a series of problems of reduced size, with the intent of finding a solution to the full problem. A subset of *A* is generated, and with a corresponding subset

of c and x, is solved to completion. Here, the size of the subset is dependent on available computational capacity.

If a feasible solution is found, we know that it is feasible for the full problem. By comparing a number of these feasible solutions, we can hope to get a good, if not optimal, solution for the original problem. Unfortunately, it is stated in this paper that the authors could not find any methods for generating the subsets that were statistically superior to a completely random generation, so it may take a long time to search a reasonable proportion of the total space.

Rubin's Heuristic

Another heuristic, reported to be successful in industry can be found in Rubin (1973). The author presents an algorithm based on repeated solving of a set covering relaxation of the original problem. The solutions to these coverings provides an upper bound on the optimal value and claims that they provide an optimal solution more than 99% of the time for large problems. He also accelerates this search by restricting the search space based on geographic locality and storing intermediate solutions. It is also made more robust by including full passengering or, as Rubin and others in industry call it, 'deadheading.'

Semidefinite Programming Relaxation

Formulation

Wolkowicz and Zhao (1996) have proposed a semidefinite programming (SDP) relaxation to the SPP. This relaxation can be directly applied to the ASP. We refer the reader to the original paper for the justification of the formulation. The authors provided two primal formulations; we present below the first listed in the article as it provided for fewer problems while developing the code for our study.

$$
\begin{aligned}
&\min tr(CY) \\
&s.t. tr(diag([0,a_j])Y) = 1 \\
&(-1,a_j)Y(-1,a_j)^T = 0 \\
&arrow(Y) = e_0 \\
&G_J(Y) = 0 \\
&Y_{00} = 1 \\
&Y \in M^{PSD}
\end{aligned}
$$

where:

- C is the matrix formed by $(0\ c)(0\ c)^T$,
- Y is the matrix formed by $(1\ x)(1\ x)^T$, indexed from 0 to n.
- $arrow(Y)$ enforces that $Y_{ii}=Y_{0i}$
- $G_J(Y)$ is a so-called 'gangster' operator, to be discussed below.

The first two constraints correspond directly to $Ax=1$ from our original SPP formulation. The 'arrow' constraint is a relaxation of the {0,1} constraint. The 'gangster' operator forces all elements of Y belonging to a set J to be forced to 0. J is the set of elements of Y where $a_{ij}=a_{ik}$ for some row i, i.e. the set of ToDs that share a flight in common with some other ToD. This acts to ensure that only one ToD is selected for each flight.

It should be noted that the last n elements of the first row of Y directly correspond to the elements of our original x, as do the diagonal elements of Y. This means that we do not need to reformulate any side-constraints we had in our original problem. However, if the reader wishes to use the more elegant second formulation presented in Wolkowicz and Zhao (1996), this reformulation becomes more complicated. A reformulation of the minimum/maximum worker hour constraint for our original problem is:

$$H_{\min} \leq \sum_{j=1}^{n} h_j Y_{jj} \leq H_{\max}$$

In addition to the original constraints, the SDP relaxation permits the addition of more constraints on the interactions between the elements of Y, and therefore on x. One such example is to include a 'gangster'-style operator to ensure that each crew in the Rostering phase is assigned only on ToD.

If the reader is only interested in the optimal value of the relaxation (say to evaluate another solution method), one can also solve the dual dual given in Wolkowicz and Zhao (1996). This dual may provide computational benefits in the form of memory usage, as well as using fewer constraints. The authors show that the primal and dual satisfy Slater's constraint qualification, and thus the duality gap is zero.

Lastly, the authors show that the SDP relaxation optimal value is at least as good as the one from the traditional LP relaxation. In fact, in the numerical results below, we find that it is often significantly stronger.

Numerical Results

To test the quality of the results given by the SDP relaxation, we performed three sets of tests. Firstly, we generated a subset of the columns for the small problem defined in figures 1 and 2 and solved its relaxation and compared it to the exact solution found using the Simple Branch & Bound (SBB) solver provided by NEOS. The relaxation provided a tight bound on the optimal value, but returned a fractional solution of 6. This solution had 4 elements with value 1, which also appeared in the exact optimal solution, and 4 elements with value ~0.5. By rounding one of these values to 1, however, we were able to resolve and get an integer optimal solution.

The SDP was coded and run in MatLab on a standard Nexus computer, using the *cvx* interface to the SDPT3 solver. The exact formulation was coded in GAMS and, as mentioned previously, submitted to the SBB solver on NEOS. The code for these problems can be found in Appendix A.

We then expanded our testing to examine 20 randomly generated dense A matrices with 20 flights and 100 ToDs each; a dense matrix is representative of a flight network with several airports in relatively close proximity. The probability of a_{ij} having value one was 0.5. We used a unit objective function. We then compared the results of the SDP relaxations with the LP and exact solutions.

As shown in figure 3 below, the SDP always detected infeasibility in the original problem, or provided a tight bound. In fact, in every case of a tight bound, an integer solution was also found. The LP provided a lower bound in every case, even the infeasible ones. Even for those problems that were feasible, the LP gives little information; telling us that at least two out of 100 columns will be needed is not exactly informative!

As before, the SDP was coded in MatLab, and both the LP and exact formulations were coded in GAMS and submitted to SBB. The MatLab code for the function used in these tests is included in Appendix B, though the test data, being randomized, is not. The GAMS code is not, as it does not differ significantly from that in Appendix A.

Run	LP	SDP	Exact
1	1.48	NaN	INF
2	1.56	NaN	INF
3	1.48	3	3
4	1.47	NaN	INF
5	1.54	NaN	INF
6	1.55	NaN	INF
7	1.57	NaN	INF
8	1.59	2	2
9	1.53	NaN	INF
10	1.65	NaN	INF
11	1.44	NaN	INF
12	1.56	NaN	INF
13	1.57	NaN	INF
14	1.55	NaN	INF
15	1.54	NaN	INF
16	1.50	NaN	INF
17	1.62	NaN	INF
18	1.51	NaN	INF
19	1.56	NaN	INF
20	1.51	NaN	INF

Fig. 3 20 20x100 random, dense matrices

Subsequent to this, we examined another set of matrices, this time with the probability of 1 being 0.33. This sparser network is more of a reasonable example of an airline network that is geographically diverse but still has a many flights going to each of its nodes. We ran 100 test cases of 20x100 randomly generated matrices. Figure 4, below, summarizes the results. Here, the LP also provides a tight bound or an indication of infeasibility. As well, both the LP and the SDP provided consistently integer solutions.

Result	LP	SDP	Exact
Infeasible	26	26	26
Tight Bound	74	74	74

Fig. 4 100 20x100 random, sparse matrices

While it appears that the SDP provides good information about the optimal solution, we wished to determine if the formulation could provide it in a timely fashion. To test this, we measured the time that a standard Nexus computer would take to solve the 100 test cases mentioned above, using both the exact (solved here using SBB on a standard GAMS installation) and SDP formulations. The exact solution took an average of 10.8s to solve, while the SDP took 23.8s. It doesn't take a sophisticated statistical tool to tell that the latter is significantly different from the former.

We postulate, however, that a large part of this difference comes from the relative maturity of the SBB solver as compared to SDPT3 and *cvx*. It should be noted, however, that the SDP formulation requires significantly more memory than the exact formulation to store its variables. This may become a major issue when attempting to apply it to real data sets.

Practical Applications

Let us now consider how the results of the SDP relaxation can be used in industry. Most basically, the lower bound on the size of the partition can be used in two ways. First, it can be used to evaluate the performance of either of the two heuristics mentioned previously, as a goal for others, including for exact solution methods.

Alternatively, if used in a traditional optimization formulation, this lower bound can be used to eliminate a portion of the optimization problem size. If the lower bound is a reasonable fraction of m, it will significantly cut down the search space; in our small example, we found a minimum of 6 tours of duty would be required, which effectively cuts the size by half.

Alternatively, since the SDP provides at least the same amount of information as the LP, it can be substituted in any existing branch and bound/cut algorithm. It can then use a solution of the SDP and then attempts to resolve it to an integer solution, feasible for the original problem.

A final potential application that we suggest here is that the SDP could be used to eliminate columns of A that will not be used in the final solution. While examining the optimal solutions provided by the SDP, we noted that, while some elements of x had fractional values, most had exactly zero. It is likely that these completely unused columns would not be used in a final optimal solution. We can then find an A' of reduced size and solve a much smaller SPP using a traditional integer programming method.

Conclusions

In this paper, we have examined what the aircrew scheduling problem is and how it is has been solved in industry. We then examined an SDP relaxation for the two phases of this problem and tested it on three test cases. The results of these test cases were promising, and suggest that further research in this area might be very rewarding. Particularly in the dense test cases, the additional computation time and space that SDP requires definitely results in a more useful solution. We also suggested several practical areas in which this research might be directed and where SDP could be used in the airline industry.

Unfortunately, there is a relative immaturity in solvers for SDP problems, at least with respect to the one used here, that makes it difficult to wholeheartedly recommend SDP to replace LP as the relaxation of choice for aircrew schedulers. It may be that, as these solvers improve, schedulers will be able to leverage the additional forms of constraints that SDP allows as well as the stronger results that SDP derives.

References

1. Butchers, E. Rod et al. "Optimized Crew Scheduling at Air New Zealand." Interfaces 31 (1) pp 30-56. (2001)
2. Gershkoff, I. "Optimizing Flight Crew Schedules" Interfaces 19 (4) pp 29-43. (1989)
3. Mingozzi, A. et al. "A Set Partitioning Approach to the Crew Scheduling Problem." Operations Research 47 (6) pp 873-888. (1999).
4. Ryan, D. M. "The Solution of Massive Generalized Set Partitioning Problems in Aircrew Rostering." The Journal of the Operational Research Society 43 (5 pp 459-467. (1992)
5. Rubin, J. "A Technique for the Solution of Massive Set Covering Problems with Application to Airline Crew Scheduling." Transportation Science 7 (1) pp 34-48. (1973)
6. Wolkowicz, H. and Zhao, Q. "Semidefinite Programming Relaxations for Set Partitioning Problems." Unpublished. (1996)
7. Wolkowicz, H. and Zhao, Q. "Semidefinite Programming Relaxations for the Graph Partitioning Problem." Discrete Applied Mathematics 96-97 pp 461-479. (1999)
8. Zhao, Q. "Semidefinite Programming for Assignment and Partitioning Problems." Thesis. University of Waterloo. (1996)

Appendix A: Small Example Code

SDP Relaxation

```
clear;
m=11;
n=25;
setToOne=[1];
A=[
1  1   0   0   0   0   0   1   1   0   0   1   0   0   0   1   1   0
1   0   0   0   0   0   0;
0  0   1   1   0   0   1   0   0   0   1   0   1   1   1   0   0   0
0   1   0   0   0   0   0;
0  0   0   1   0   0   0   0   0   0   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0;
0  0   1   0   0   0   1   0   0   0   0   0   0   0   1   0   0   0
0   0   0   0   1   0   0;
1  0   0   0   0   0   0   0   0   0   0   1   0   0   0   1   0   0
0   0   0   0   0   0   0;
0  1   0   0   0   0   0   1   0   0   0   0   0   0   0   0   1   0
0   0   0   1   0   0   0;
0  0   0   0   1   0   0   0   0   1   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0;
0  0   0   0   0   1   1   1   0   0   0   0   0   0   1   0   1   1
0   0   1   0   0   0   0;
0  0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0;
0  0   0   0   0   0   0   0   0   0   0   1   1   0   0   0   0   0
0   0   0   0   0   0   1;
0  0   0   0   0   0   0   0   0   0   0   0   0   1   1   1   1   1
0   0   0   0   0   1   0];
]
c=ones(n,1);


%Calculate C
C=diag([0 c']);

%Now, let's construct Ghat, a matrix that will act like the Gangster
Operator
Jtemp=zeros(n,n);
for i=1:m
    %Now, for each row we just found, find the non-zero elements
    for j=1:n
        for k=1:n
            if A(i,j)==1 & A(i,k)==1 & j~=k,
                Jtemp(k,j)=1;
                Jtemp(j,k)=1;
            end
        end

    end
end

Ghat= [0 zeros(1,n);zeros(n,1) Jtemp];
```

```
%Now, lets try to do the optimization using CVX
%perhaps we should do a small example first, if this doesn't work.
cvx_begin
    %cvx_quiet(true);
    variable Y(n+1,n+1) symmetric;
    minimize(trace(C*Y));
    subject to
        for i=1:m
            trace(diag([0 A(i,:)])*Y)==1;
            [-1 A(i,:)]*Y*[-1 A(i,:)]'==0;
        end
        diag(Y)-Y(1,1:1:n+1)'==[0;zeros(n,1)];
        Ghat.*Y== zeros(n+1);
        for i=1:length(setToOne)
            Y(setToOne(i),setToOne(i))==1;
        end
        Y == semidefinite(n+1);
cvx_end
X=Y(1,2:1:n+1)
```

Exact Formulation

```
Sets
i /1*11/
j /1*25/;

Parameters
c(j) /1*25  1/;

Table A(i,j)
      1     2     3     4     5     6     7     8     9     10    11
      12    13    14    15    16    17    18    19    20    21    22
      23    24    25
1     1     1     0     0     0     0     0     1     1     0     0
      1     0     0     0     1     1     0     1     0     0     0
      0     0     0
2     0     0     1     1     0     0     1     0     0     0     1
      0     1     1     1     0     0     0     0     1     0     0
      0     0     0
3     0     0     0     1     0     0     0     0     0     0     1
      0     0     0     0     0     0     0     0     0     0     0
      0     0     0
4     0     0     1     0     0     0     1     0     0     0     0
      0     0     0     1     0     0     0     0     0     0     0
      1     0     0
5     1     0     0     0     0     0     0     0     0     0     0
      1     0     0     0     1     0     0     0     0     0     0
      0     0     0
6     0     1     0     0     0     0     0     1     0     0     0
      0     0     0     0     0     1     0     0     0     0     1
      0     0     0
7     0     0     0     0     1     0     0     0     0     1     0
      0     0     0     0     0     0     0     0     0     0     0
      0     0     0
8     0     0     0     0     0     1     1     1     0     0     0
      0     0     0     1     0     1     1     0     0     1     0
      0     0     0
```

```
9       0       0       0       0       0       0       0       0       1       1       1
        0       0       0       0       0       0       0       0       0       0       0
        0       0       0
10      0       0       0       0       0       0       0       0       0       0       0
        1       1       0       0       0       0       0       0       0       0       0
        0       0       1
11      0       0       0       0       0       0       0       0       0       0       0
        0       0       1       1       1       1       1       0       0       0       0
        0       1       0;

Variables
X(j)  1 if TOD j is selected
z           objective value;

Binary Variables
x(j);

Equations
obj         objective value
Ax(i) each flight in one TOD (i.e. partition constraint);

obj .. z =e= sum(j, c(j)*x(j));
Ax(i) .. sum(j,A(i,j)*x(j)) =e= 1;

Model Small /all/;
option mip=sbb;

Solve Small using mip minimizing z;
```

Appendix B: MatLab Function Code

```
function [X, obj] = setpart(A,c)
%Takes a matrix A and vector c for constraints and objective function
%The solves the set partition, and returns an optimal solution X and
optimal
%value
obj
[m n]=size(A);

setToOne=[1];

%Calculate C
C=diag([0 c']);

%Now, let's construct Ghat, a matrix that will act like the Gangster
Operator
Jtemp=zeros(n,n);
for i=1:m
    %Now, for each row we just found, find the non-zero elements
    for j=1:n
        for k=1:n
            if A(i,j)==1 & A(i,k)==1 & j~=k,
                Jtemp(k,j)=1;
                Jtemp(j,k)=1;
            end
        end

    end
```

```
end

Ghat= [0 zeros(1,n);zeros(n,1) Jtemp];


%Now, lets try to do the optimization using CVX
%perhaps we should do a small example first, if this doesn't work.
cvx_begin
    cvx_quiet(true);
    variable Y(n+1,n+1) symmetric;
    minimize(trace(C*Y));
    subject to
        for i=1:m
            trace(diag([0 A(i,:)])*Y)==1;
            [-1 A(i,:)]*Y*[-1 A(i,:)]'==0;
        end
        diag(Y)-Y(1,1:1:n+1)'==[0;zeros(n,1)];
        Ghat.*Y== zeros(n+1);
        for i=1:length(setToOne)
            Y(setToOne(i),setToOne(i))==1;
        end
        Y == semidefinite(n+1);
cvx_end
X=Y(1,2:1:n+1);
obj=trace(C*Y);
```