Imperial College of Science and Technology
(University of London)
Department of Management Science

ALGORITHMS FOR CREW SCHEDULING PROBLEMS

by

HALIM CHEDDAD

A thesis submitted for the degree of
Doctor of Philosophy of the University of London
and for the
Diploma of Imperial College

May 1987

ABSTRACT

The crew scheduling problem (CSP) forms the basic structure of many real life instances such as crew scheduling for airline industry,scheduling for buses,trains,mass transit systems,postal fleet,job shop scheduling,crew rosterings,...etc... .Solution methods for two variants of the CSP are the subject of this thesis. A survey of applications and algorithms is given in the first chapter.

In Chapters 2 and 3 we consider the CSP where a set of tasks with known starting times and durations must be performed by a set of crews,each crew having a limited work-duty time period. A "cost" is incurred in following one task by another and the objective is to minimize these "transition" costs. Two formulations of the problem have been considered. The first is a direct 0-1 programming formulation. The second formulation involves the representation of the CSP as a network flow problem in an expanded graph with additional set-partitioning type constraints. We showed via extensive computational results involving more than 250 problems with up to 150 tasks that the LP relaxation of this second formulation provides naturally integer solution(and hence a solution to the CSP) in all cases tried.Applying the graph expansion technique (GET) to two straigthforward extensions of the CSP, more than 300 randomly generated problems could be solved optimally.

Chapters 4 and 5 deal with a more general and practical version of the CSP (GCSP) in which it is assumed that the starting times of the tasks are no longer fixed but given within time-windows. Furthermore, several types of vehicle with different characteristics are assumed to cover the tasks. The GCSP was first tackled with GET which proved to be very effective for problems for which the task time-windows are narrow. However as the time-window increases , it becomes impossible to use GET because of the size of the graph which grows exponentially with the size of the problem. In our attempt to solve larger GCSP's,a tree search algorithm has been devised. Using Lagrangian relaxation ,one of the most successful integer programming techniques to obtain lower bounds ,six different integer programming formulations of the problem were considered and compared.The results obtained with this tree search procedure were satisfactory. Thus all randomly generated GCSP's of size varying between 10 and 50 tasks and with a time-window allowed to vary in the range [0,24 hours] were solved optimally.

Finally,conclusions with ideas for future research, are given in the last chapter.

ACKNOWLEDGEMENTS

Firstly, I would like to express my deep thanks to my supervisor Prof. Nicos Christofides for his constant help, encouragement and advice throughout the course of this research.

I am very grateful to the Algerian Ministry of Higher Education for giving me the opportunity to carry out this work. Their financial support is gratefully acknowledged.

I would like to thank my fellow researchers in the management science research unit for their friendliness and team-work spirit. Thanks also to A. Khelifa and A. Shaikh for their invaluable help in the typing of the thesis.

Last, but not least, I am very much indebted to my Parents and all my Brothers for their encouragement during my research years and for being such an understanding Family during these hard times.

To FERHAT and HOURIA
My Parents.

TABLE OF CONTENTS

Contents Page

CHAPTER 1

INTRODUCTION

1- Introduction .

The crew scheduling problem (CSP) and all its extensions form the basic structure of many real life problems such as crew scheduling for airline industry [2],scheduling for buses [147], trains [3,62], garbage trucks [34], mass transit systems [16,144], job shop scheduling problems [105,106], crew rosterings [33], ...etc...

In this section we present two of the main applications of the CSP . These are the airline CSP and the bus CSP . Also one of the earliest application of the CSP to a navy problem [52] is presented .

1-1 The Airline Crew Scheduling Problem (ACSP).

One of the most important and most well known problems facing airline companies is the airline crew scheduling problem (ACSP) [2].

Having established a timetable of the flights' schedule,the airline company is faced with the problem of producing another timetable for assigning crews to flights. This new timetable must be in accordance with the union regulations and it should be devised in such a way that all flights are covered and that the total crew cost is minimized. The problem is usually broken into two phases :

(i) Daily or weekly crew schedules,called "rotations" are constructed ;

(ii) The rotations are grouped together to form "rosterings" ie monthly schedules.

From now on,we will be dealing only with a variant of the first phase which is commonly called the "airline crew scheduling problem". The second phase refered to as the "bid-lines problem" has been considered by several other authors[58,75].

Defining a flight-leg as being a non-stop trip between a pair of cities(or towns), a rotation is a round trip which consists of sequences of flight-legs, the first of which and the last of which must respectively originate and terminate at the crew base. Each sequence of flight-legs is called a "duty period" , ie a period of time during which a crew may operate a plane without a rest break. The duty periods within a rotation might be separated by long rest periods,called "layovers",intended for sleeping. The formation of rotations must be in accordance with union regulations and company policy. The aim of the company is to minimize the cost of operating the crews

The union regulations guarantee the following:

(a) The duration of each duty period must be less than or equal to a given time T;

(b) Between any two consecutive duty periods there should be a minimum period of rest time ;

(c) Each crew should have a sufficient time at the crew base.
These are the main union regulations for all airline companies,and depending on the importance to the company additional rules might be considered.

1-2 Bus Crew Scheduling.

Another important application of the CSP is in the area of bus crew scheduling. In public bus scheduling the process involves the revision of timetables which generate a series of trips to be run and then the allocation of vehicles and drivers to be assigned to these jobs. The process is usually broken into two phases : first the trips or routes are generated then the fleet is scheduled. This sequential process is necessary because otherwise the problem would become computationally intractable due to its very large size. Added to this is the complication due to the union rules which govern the assignement of drivers (and buses) to trips over a working period.

A much simpler case of this bus scheduling problem which involves less complex rules and which is very closely related to our problem is the school bus scheduling problem(SBSP) [31,100,126]. Like the general case,the SBSP involves two steps:first a set of routes or trips are generated for each school such that all bus stops are visited and the students are delivered at a given time. Then,the drivers and buses must be scheduled to cover the trips.

Now let us consider the problem : the routing phase generates a set of M_k trips for each school k=1,...,K where K is the number of schools. For each trip we know the starting place(first stop) and finishing place(last stop),the total trip time duration,and the starting and finishing times of the trip. Once these parameters have been determined,each trip can be considered as one task which does not depend on the school.The problem consists of finding the minimum cost fleet required to cover all trips. The main contribution to cost in the objective function is due to vehicle operating costs which are a function of the number of vehicles(crews) used and the total time each vehicle is used(trip time and deadhead time). The deadhead time c_{ij} for a single crew serving two trips is the time required to go from the finishing place of trip i to the starting place of trip j.

In practice a few additional constraints exist :

(i) each driver cannot work more than a given number of hours over the daily planning period;

(ii) a window on the due date (starting time and finishing time) of each task might be established giving an earliest and latest possible finish time;

(iii) either all buses have sufficient capacity to process any trip or some buses would be able to process only a subset of trips;

(iv) the buses might be housed at more than one depot.

1-3 Scheduling of Navy Fuel Oil Tankers.

Consider table1-1. Each row corresponds to a pick-up point and each column correponds to a discharge point. The sequence of figures t_{ij} inside box(i,j) represent the times a tanker is to begin loading at pick-up point i in order to deliver at discharge-point j.

In addition we are given two arrays a_{ij} and d_{ij} which represent respectively the loading-traveling time from i to j and the unloading-travel time from i to j. These are represented in table 1-2 and table 1-3.

The problem consists of finding the minimum number of tankers to meet the fixed schedule established in table 1-1. This problem is a network flow problem [63] . Thus the procedure k which consists of loading fully a tanker at a pick-up point i to deliver to a discharge-point j can be viewed as a task k whose starting place is i,whose finishing place is j and whose starting time t_{ij} is given in table 1-1. The duration of the task is defined by the loading-traveling time a_{ij} (table1-2). The unloading-traveling time d_{ij} of table 1-3 represent the empty travel times. The empty travel time d_{ij} for a tanker serving two tasks k and k' is the time required to unload the tanker at i and traveling to j where i is the finishing place of task k and j is the starting place of task k'.

Table 1-1 : Starting-times (minutes).

	Delivery point C	Delivery point D
Pick-up point A	50,400	200
Pick-up point B	350	75,150

Table 1-2 : Loading Travel-Times(minutes)

To / From	C	D
A	50	100
B	150	100

Table 1-3 : Unloading Travel-Times(minutes)

To / From	A	B
C	25	20
D	40	50

2- The Basic Crew Scheduling Problem.

All combinatorial optimization problems can be divided into two main groups [105]. There is a vast majority of problems which,from a computing point of view,are hard to solve and a tiny minority which consists of all the easy problems for which efficient (ie increasing polynomially with problem size) solution techniques already exist. Loosely speaking,the problems of the first type belong to the class of NP-complete problems while the remaining ones are refered to as polynomial problems. Because of its importance in understanding the complexity of hard and easy combinatorial problems [131],the notion of NP-completeness is dealt with in greater detail in section 3 where the meanings of "hard","easy","NP-complete" and"polynomial" problems are given.

All the problems tackled in this thesis fall into the class of NP-complete problems. The common characteristic of problems in this class is that most can be viewed as "easy" problems with additional constraints. For our particular case, the easy problem on which all the problems considered are based can be defined as follows :

"A set of n tasks is given,the ith task being defined by a starting time ST_i,a starting place SL_i,a finishing time FT_i and a finishing place FL_i. A number of crews(say,K) are available to perform the above tasks. A feasible work-schedule for a given crew consists of an ordered sequence of tasks $i_1,i_2,...i_r$ so that :

$$ST_{i_k} \geq FT_{i_{k-1}} + \Delta(FL_{i_{k-1}},SL_{i_k}) \qquad k=2,...,r \qquad (1\text{-}1)$$

where $\Delta(FL_p,SL_q)$ is the travel time between locations FL_p and SL_q and is given for all location pairs.

A feasible solution to the problem consists of assigning feasible schedules to crews,as above,so that every task is performed once only by some crew.

The cost of a schedule $i_1,i_2,...i_r$ for a single crew is taken to be :

$$\sum_{k\in B_r} d_{i_k} + \sum_{k\in B_{r-1}} C_{i_k i_{k+1}} \qquad \text{where } B_r = \{1,2,....,r\} \qquad (1\text{-}2)$$

and where d_i is the cost of performing task i_k,and $c_{i_k i_{k+1}}$ is the cost of distance travelled (from FL_{i_k} to $SL_{i_{k+1}}$) and time taken (from FT_{i_k} to $ST_{i_{k+1}}$) in following the execution of task i_k by the execution of task i_{k+1}. Since all tasks are required to be performed exactly once,the sum of the first cost terms in (1-2) for all crews is a constant $\sum_{B_m} d_{i_k}$

where $B_m=\{1,2,...,m\}$. Hence complete schedules are distinguished in cost only by the sum of the second terms in (1-2) and,therefore,we will take only the second terms in (1-2) to be the cost of the schedule $i_1,i_2,...i_r$.

The objective function of the problem is,therefore,to find a set of feasible schedules, so that every task is performed exactly once,and the sum of the costs of the schedules is minimized.

In the course of the thesis,we will refer to this problem as the basic crew scheduling problem(BCSP). The BCSP can be efficiently solved using a minimum cost network flow formulation. This will be discussed in section 2-4,but before this we need to represent the problem in graph theoretic terms.

2-1 A Direct$\overset{ed}{\wedge}$ Graph Representation of the BCSP.

Let us illustrate the main idea of constructing the graph G_u representing the BCSP on the following example.

Consider the input data of a BCSP given in tables 1-4 and 1-5:

Table 1-4 : Input Data of the BCSP.

Task	Starting time (minutes)	Finishing time (minutes)	Starting location	Finishing location
1	50	150	A	B
2	80	170	A	C
3	250	380	D	A
4	350	450	C	D
5	550	650	B	A

Table 1-5 : Distances between Locations.

	A	B	C	D
A	0	100	90	130
B	100	0	150	50
C	90	150	0	100
D	130	50	100	0

Table 1-4 gives for each task (column 1) its starting time (column 2),its finishing time (column 3) , its starting location (column 4) and its finishing location (column 5).

The time durations(in minutes) required to get from any location i (i=A,B,C,D) to any other location j(j=A,B,C,D) are given in table 1-5.It is worthwhile noting that we have assumed that the time required to get from location i to location j is the same as that required to go from j to i. In real life problems (say in the case of an airline problem) this is not always the case.

(i) Task-arcs representation: If we represent each task by a directed arc (α_i,β_i) on a time axis we obtain figure1.1 . α_i(resp.β_i) represent the initial (resp. final) extremity of arc i (i=1,...,5) . The couple of figures above each arc represent the cost $\underline{c}_i$ and the time duration(in minutes) $\underline{\tau}_i$ = FT_i -ST_i of the corresponding task respectively.

In fact these arcs can be represented in such a way to get the directed graph G_u^1 of figure 1.2. These arcs will be refered to as "task-arcs". It is worthwhile noting that they have been numbered in an increasing order of their starting times. This way of ordering the task-arcs will be adopted in the rest of the thesis. Also,we will sometimes refer to the task-arcs as "required-arcs".

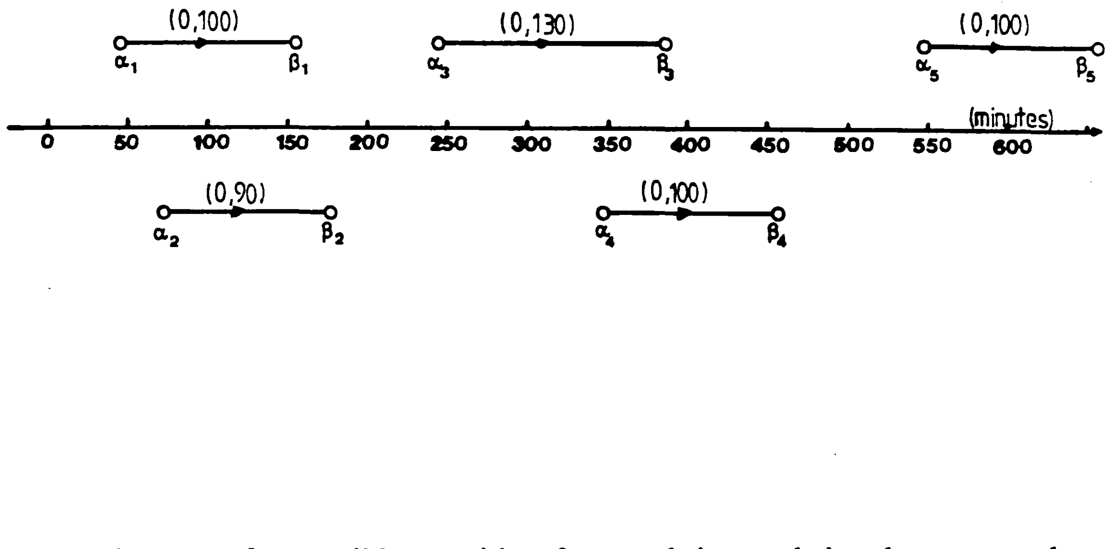
(ii) Linking-arcs representation. In graph G_u^1 we express the fact that two tasks of the BCSP can be covered by the same crew,by linking the two corresponding task-arcs. More precisely,we join the final extremity of the first task-arc (say(α_i,β_i)) to the initial extremity of the second task-arc (say (α_j,β_j)) with a directed arc going from (α_i,β_i) to (α_j,β_j) (ie linking β_i with α_j). Arc (β_i,α_j) will be called a "linking-arc".It has a cost c_{ij} and time duration $\tau_{ij} = ST_j - FT_i$

Two task-arcs (α_i,β_i) and (α_j,β_j) of G_u^1 are linked if the following condition is satisfied :

$$ST_j \geq FT_i + T(FL_i,SL_j) \qquad (1\text{-}3)$$

where ST_j (resp. FT_i) is the starting (resp. finishing) time of task j (resp. i) and $T(FL_i,SL_j)$ is the time duration for going from the finishing location FL_i of task i to the starting location SL_j of task j.

Figure 1-1 : A 5 Task-BCSP.



Taking the cost of a possible transition from task i to task j to be computed as $100+(ST_j - FT_i)$ we obtain graph G_u^2 of figure 1-3. As for G_u, the couple of figures associated with each linking-arc represent its cost and its time duration (in minutes).

Let us illustrate how the linking-arcs departing from task-arc (α_1,β_1) have been constructed :

Figure 1-2 : Network G_u^1

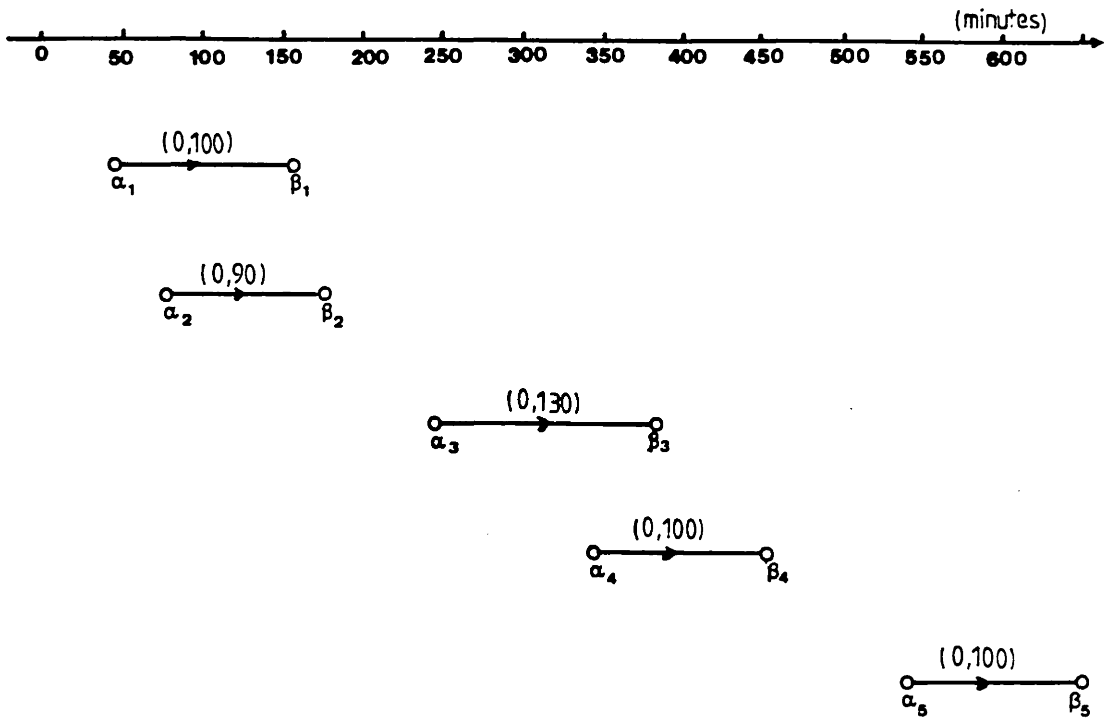
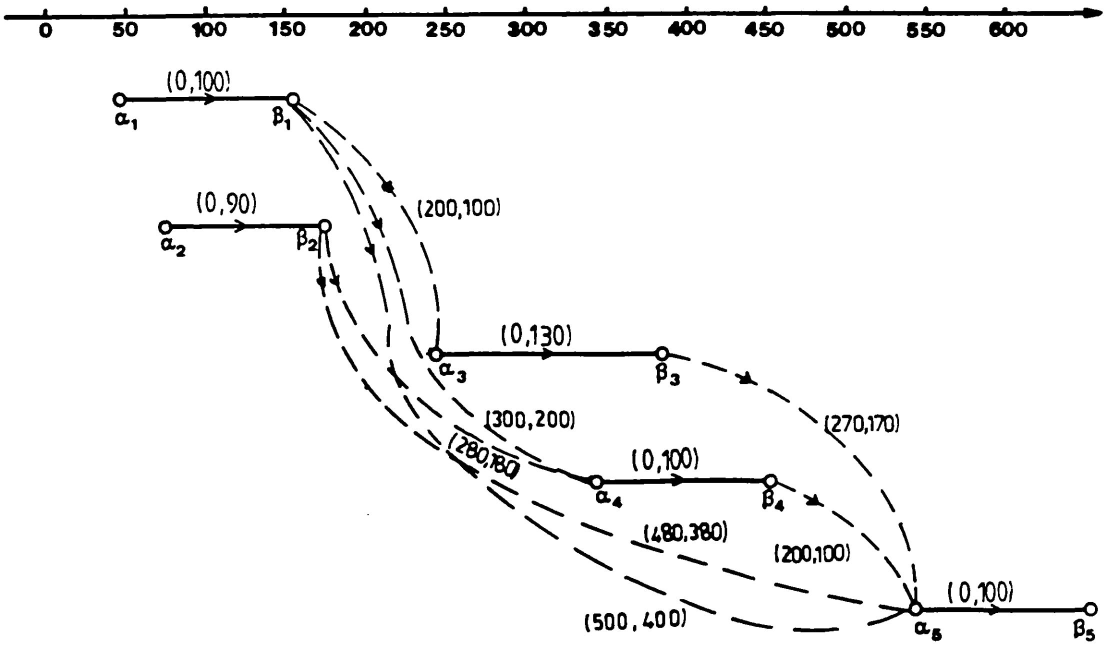


Figure 1-3 : Network G_u^2



(a) can we link task-arc (α_1,β_1) and (α_2,β_2) ?

No. Because : $ST_2=80 < FT_1+T(FL_1,SL_2)=150+100=250$

(ie condition (1-3) is violated)

(b) can we link task-arc (α_1,β_1) and (α_3,β_3) ?

Yes. Because : $ST_3=250 \geq FT_1+T(FL_1,SL_3)=150+50=200$

(ie condition (1-3) is satisfied)

(c) and similarly with task-arcs (α_4,β_4) and (α_5,β_5), they are linked with (α_1,β_1) because condition (1-3) is satisfied for each one of them .

(iii) Source-arcs and Sink-arcs. By adding a super-source β_S and a super-sink α_R,and joining β_S and α_R with respectively the starting nodes and finishing nodes of the task-arcs,we obtain the network G_u representative of the BCSP given in figure 1-4.

The arcs joining β_S to the initial extremities of the task-arcs are called source-arcs and the arcs joining the final extremities of the task-arcs to α_R are called sink-arcs. A cost $c_{S,i}$ (resp. $c_{i,R}$) and a time-duration $\tau_{S,i}$ (resp. $\tau_{i,R}$) are associated with each source-arc (β_S,α_i) (resp. sink-arc (β_i,α_R)). They are represented in G_u by the couple of figures beside each arc.

2-2 A Graph-Theoretical Definition of the BCSP.

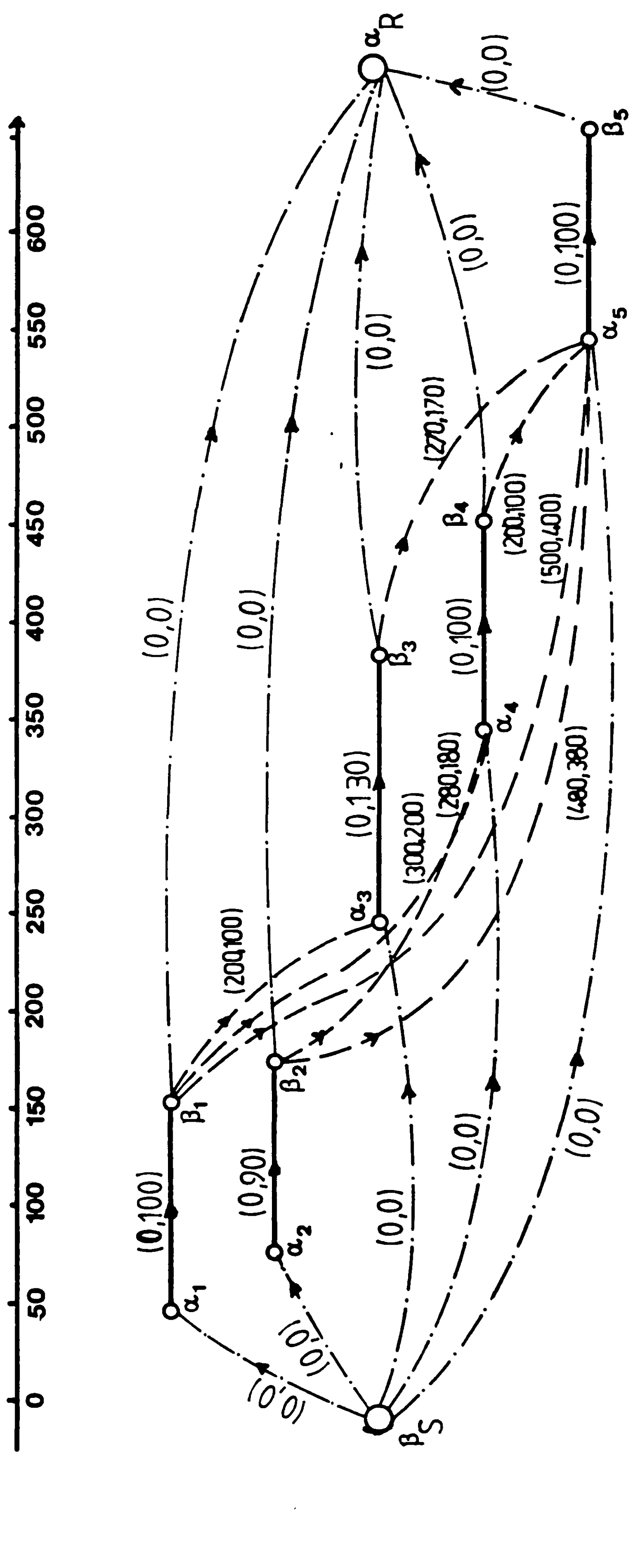
In this section a new definition of the BCSP in terms of graph G_u is given. Before this,let us consider the two following useful definitions.

Definition 1. A path P in G_u is a sequence of directed arcs

$$P=((\beta_s,\alpha_{i_1})(\alpha_{i_1},\beta_{i_1})(\beta_{i_1},\alpha_{i_2})...(\beta_{i_k},\alpha_{i_{k+1}})(\alpha_{i_{k+1}},\beta_{i_{k+1}})(\beta_{i_{k+1}},\alpha_R))$$

Definition 2. The cost (time) of a path P is the sum of the costs (times) of the arcs that form P.

Figure 1-4 : Network G_u



Example. For the network G_u of figure 1-4 , the following sequence of arcs is a path P of G.

$$P=((\beta_s,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_3)(\alpha_3,\beta_3)(\beta_3,\alpha_5)(\alpha_5,\beta_5)(\beta_5,\alpha_R))$$

the length of P is

$$L_p = 0 + \underline{\tau}_1 + \tau_{1,3} + \underline{\tau}_3 + \tau_{3,5} + \underline{\tau}_5 + 0$$

$$L_p = 0 + 100 + 100 + 130 + 170 + 100 + 0 = 600$$

the cost of P is

$$C_p = c_{s,1} + \underline{c}_1 + c_{1,3} + \underline{c}_3 + c_{3,5} + \underline{c}_5 + c_{5,R}$$

$$C_p = 0 + 0 + 200 + 0 + 270 + 0 + 0 = 470$$

Definition 3. Graph definition of the BCSP.

The BCSP is to minimize the cost of covering all the task-arcs of the associated network G_u,with a given number of paths(say,K) such that each task-arc must be covered once and only once by a path.

2-3 A Minimum Cost Network Flow Formulation of the BCSP.

BCSP is an easy problem. As shown by *Ford and Fulkerson* [63] it can efficiently be solved by using a minimum cost network flow formulation which can be described as follows :

If in network G_u we let

x_{ij} = 1 if arc (β_i,α_j) is in the optimal solution ;

= 0 otherwise ;

the BCSP becomes :

$$\text{Min} \sum_{(\beta_i,\alpha_j)\in N} c_{ij}x_{ij} \tag{1-4}$$

subject to :

$$\sum_{j\in V_i^+} x_{ij} - \sum_{p\in V_i^-} x_{pi} = \begin{cases} 0 & \text{for } i=1,\dots,n \\ K & \text{for } i=S \\ -K & \text{for } i=R \end{cases} \tag{1-5}$$

$$x_{ij} \in \{0,1\} \tag{1-6}$$

where N is the set of all linking-arcs,source-arcs and sink-arcs,

$V_i^+ = \{ j \mid (\beta_i,\alpha_j) \in N \}$

$V_i^- = \{ j \mid (\beta_j,\alpha_i) \in N \}$

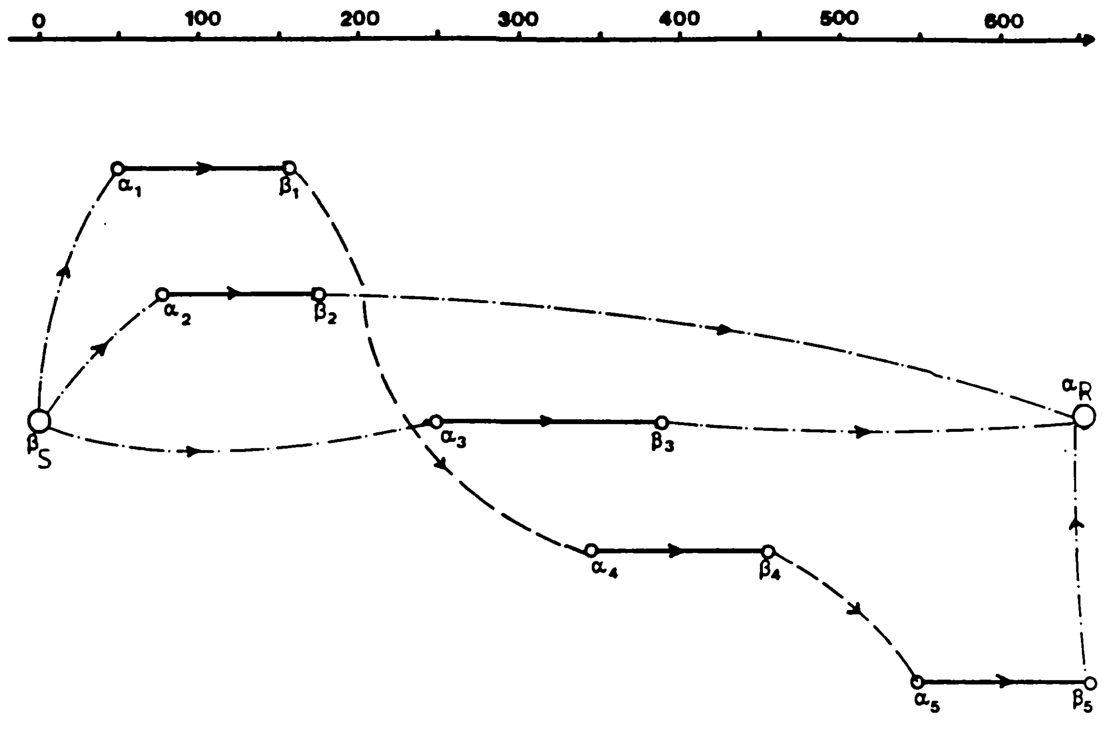
K is the number of paths(crews) with which we want to cover the task-arcs of G_u;

n is the number of tasks.

Clearly, this is a classical minimum cost network flow formulation where constraints (1-5) express the conservation of flow at each node of G_u. A variant of the BCSP is used in a portion of the RUCUS program [28] which was developed for scheduling vehicles and crews for mass transit systems.

Example. Consider the BCSP represented by network G_u of figure 1-4. Let us assume that the number in crews in the problem is 3 . Using a minimum cost network flow algorithm[63] we obtain the optimal solution of the BCSP represented in figure 1-5.

Figure 1-5 : Optimal Solution of the 5 task-BCSP.



3- An Introduction to NP-Completeness.

All problems can be divided into two main groups. The first one which consists of all undecidable problems has been known only for the last 50 years. In 1936,Turing [143] proved that some problems are so "hard" that no algorithm at all can be given for solving them ie they are "undecidable". The second group which fortunately represents the vast majority of problems consists of all the decidable problems. This group in its turn can be split up into two categories of problems.

To help the reader understand more easily the definition of each category we need to introduce the following concepts :

(i) the way programs run on actual computers is that at any time,whatever the algorithm is doing,there is only one thing it could do next. Such computers(and algorithms) are called deterministic.

(ii) Because of its usefulness in understanding the theory of complexity the concept of non-determinism was introduced. With this we assume that there exists a computer with the following power: when an algorithm is faced with a choice of several options,it has the capability to "guess" the right one. This is of course an "unreasonable" and "unrealistic" assumption.

(iii) Given a problem of size n (in our case n is the number of tasks) and a real function t(n) in n we say that an algorithm runs in time O(t(n)) if the computing time to execute the algorithm f(n) is such that

$$| f(n) | < c \, | t(n) |$$

where c is a constant and $|A|$ is the absolute value of A. For example, if t(n) is a polynomial we say that the algorithm runs in polynomial time and if t(n) is exponential we say that it runs in exponential time. One example [96,149] of an algorithm that has exponential time complexity is the well known simplex algorithm for linear problems [51]. It is worthwhile noting that if t(n) is a logarithmic function it is considered as if it is a polynomial.

With these principles in hand we are ready to define the class of decidable problems.

The first subgroup of this class consists of all problems that cannot be solved in polynomial time,not even with a non-deterministic machine. This group was first discovered in the 1960's [83]. In fact at that time,the problems that fell into this category were artificial in that they were basically constructed for this purpose. And it is only in the beginning of the 1970's that some natural problems have been shown to belong to this category [59,119]. As far as the level of difficulty in solving the problems is concerened this category of problems comes directly after the class of undecidable problems.

The second subgroup of decidable problems is made up of all the problems that can be solved in polynomial time with a non-deterministic machine. It is called the

NP-class. Because most of the practical known problems and for this matter most of the combinatorial optimization problems are non-deterministically polynomial [95],we will be dealing from now on only with this class of problems.

Now let us ask the following question : Are all the problems in class-NP "easy" or "hard" to solve ?

Clearly before answering such a question we need first of all to define "easy" and "hard". Although there is no rigorous or exact definition for either term, normally a problem is considered to be "easy" if there exists an algorithm that can solve it in polynomial time. If no such algorithm exists then the problem is "hard". To illustrate this,consider table 1-6 which compares the performance of different time complexity functions(f(n) which is given in column 1). Assuming that each computing operation takes one microsecond each entry of the table gives the execution computing time of an algorithm that runs in time O(f(n)) when it is applied to a problem of size n. The first row of the table gives the size of the problem and the first column represents the complexity function. It is clear that as soon as we pass from polynomial problems to exponential problems it becomes impossible to solve even small problems. One might argue that if we use a faster computer we can considerably reduce the execution time. Table1-7 compares the performance of an exponential algorithm (2^n) when it is run on computers with different speed. You can see that even if we can devise a computer which is one million times faster than the current ones this will produce only slight improvement in the execution time of an exponential algorithm (2^n).

All the problems that can be solved in polynomial time by a deterministic algorithm are said to belong to class P. They are "easy" and they all also belong to class NP. Clearly if a problem can be solved in polynomial time by a deterministic machine then it can automaticly be solved in polynomial time by a non-deterministic machine.

Table 1-6 : Comparison of Different Time Complexity Functions.

f(n) \ n	10	20	30	40	50	75	100	150
Log(n)	2.3 mseconds	2.9 mseconds	3.4 mseconds	3.6 mseconds	3.9 mseconds	4.3 mseconds	4.6 mseconds	5.0 mseconds
n	10 mseconds	20 mseconds	30 mseconds	40 mseconds	50 mseconds	75 mseconds	100 mseconds	150 mseconds
nLog(n)	23 mseconds	60 mseconds	102 mseconds	147 mseconds	195 mseconds	323 mseconds	460 mseconds	751 mseconds
n^2	100 mseconds	400 mseconds	900 mseconds	0.0016 seconds	0.0025 seconds	0.0056 seconds	0.01 seconds	0.02 seconds
n^3	0.001 seconds	0.008 seconds	0.027 seconds	0.064 seconds	0.125 seconds	0.420 seconds	1 seconds	3.375 seconds
n^4	0.01 seconds	0.16 seconds	0.81 seconds	2.56 seconds	6.25 seconds	31.6 seconds	100 seconds	15.6 minutes
n^5	0.1 seconds	3.2 seconds	24.3 seconds	1.70 minutes	5.2 minutes	39.5 minutes	2.78 hours	39 hours
2^n	0.001 seconds	1.05 seconds	17.9 minutes	12.7 days	35.7 years	11.9 centuries	4.02x10 centuries	4.5x10 centuries

Table 1-7 : Performance of an Exponential algorithm when run on Computers with Different Speeds. (Largest problem solved in 10 hours).

computer speed(*)	1	10	100	1000	10000	100000	1000000
Largest Problem	35	38	42	45	48	52	55

Now going back to our original question of whether theNP problems are easy or hard,we can put the question in the following simple way :

is class P = class NP ?

(a) To prove that class P = class NP we need to show that each problem of NP is in P ie for each problem of NP there exists a polynomial deterministic algorithm that can solve it ;

(b) To prove that class P ≠ class NP it is sufficient to exhibit a single NP problem which does not belong to P ie for which no deterministic algorithm exists which can solve it in polynomial time.

Up till now,inspite of the considerable amount of effort that has gone into answering this question nobody has been able to prove or disprove this equality. This is indeed a very frustrating situation especially when we know that a huge number of real life problems can be solved in polynomial time with a non-deterministic algorithm but for which no one has been able to produce a polynomial deterministic algorithm to solve one single such problem.

There is a class in NP (i.e. a subgroup of NP) in which every problem P_0 has the property that every other problem in subgroup can be polynomially reduced to it . Thus, if P_0 can be solved with a polynomial time algorithm then so can every problem in subgroup and if any problem in subgroup is intractable then P_0 also must be intractable . These problems in a sense are the "hardest" in NP . They are called NP complete problems . The importance of this class is that to prove that P=NP it is sufficient to solve a single NP-complete problem in polynomial time using a deterministic algorithm.

Given a non-deterministic polynomial problem,to determine if it is NP-complete all we need to prove is that there exists a known NP-complete problem which is polynomially reducible to it. That is there exists a polynomial algorithm that reduces one problem into the other and vice versa. Cook[46] proved that if such reduction between the two problems is possible then any polynomial time algorithm for the second problem can be converted into a corresponding polynomial time algorithm for the first problem and vice-versa.

The first problem in the list of NP-complete problems which run into thousands is the satisfiability problem proved by Cook[47].Subsequently Karp [94] presented a collection of results proving that many well known combinatorial problems,including the travelling salesman problem, are NP-complete. All the other problems in that list were,over the last 15 years,proved to be NP-complete using the principle of polynomial reducibilty.

4- The Crew Scheduling Problem (CSP).

A feasible work-schedule ,in the BCSP, for a given crew was defined as consisting of an ordered sequence of tasks $i_1,i_2,...,i_r$ so that :

$$ST_{i_k} \geq FT_{i_{k-1}} + \Delta(FL_{i_{k-1}}, SL_{i_k}) \qquad k=2,\dots,r$$

where $\Delta(FL_p,SL_q)$ is the travel time betwen the finishing place of task p and the starting place of task q, ST_{i_k} is the starting time of task i_k and $FT_{i_{k-1}}$ is the finishing time of task i_{k-1}.

In the crew scheduling problem we will be considering , the schedule above is required to be of duration no more than T hours, called the "work duty period" ,ie

$$FT_{i_r} - ST_{i_1} \leq T \qquad (1\text{-}7)$$

This constraint,commonly encountered in practice,corresponds to fuel restrictions(on vehicles),maintenance considerations,union requirements,...etc...

Whereas the BCSP can be efficiently solved using a polynomial algorithm the CSP is NP-complete [15,16,32].

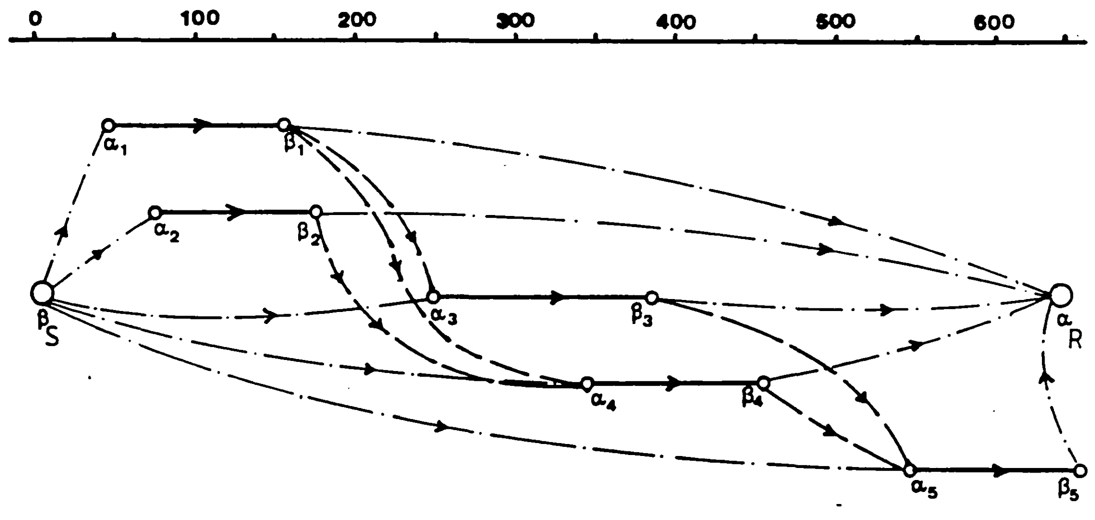
All the formulations of the CSP and the formulations of all the CSP's extensions,we have considered,are based on either expansions or modifications of the network G representative of the CSP. Consequently a description of the procedure for constructing G is necessary . This will be dealt with in this section.

Consider network G_u, of figure 1-4,representative of the 5 task-BCSP of table1-1. If we assume that an additional time constraint which restricts the work duty period to be less than 400 minutes has been imposed on the problem ,then it will become meaningless and unnecessary to link in G_u the pair of task-arcs that will produce paths of length greater than 400 minutes. Thus,linking-arc (β_1,α_5) has to be discarded since the path that will result by considering it ie

$$P = \{(\beta_5,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_5)(\alpha_5,\beta_5)(\beta_5,\alpha_R)\}$$

has length 600 minutes and this violates the limit on the work duty period. For the same reason linking-arc (β_2,α_5) also has to be eliminated. The resulting graph G is representated in figure 1-6.

Figure 1-6 : Network G representing a 5 task-CSP.



The procedure for constructing network G representative of a CSP is the same as that , of section 2-2 , for representing graph G_u of the BCSP . The only difference is that 2 conditions , instead of one , have to be satisfied before any 2 task-arcs can be joined . These are as follows :

Two task-arcs i and j (i precedes j) can be covered by a same crew if :

(i) $ST_j > FT_i + \Delta(FL_i, SL_j)$

This condition has already been considered earlier.

(ii) $FT_j - ST_i \leq T$

This is the additional condition. It means that the time required to fly both tasks i and j must be less than or equal to the work duty period T. FT_j is the finishing time of task j and ST_i is the starting time of task i.

The graph $G=(X=A \cup B, U=R \cup N_1 \cup N_2)$ now consists of the set of vertices $X=A \cup B$ where

$A=\{\alpha_i \mid i=1,...,m\} \cup \{\alpha_R\}$

and

$B=\{\beta_i \mid i=1,...,m\} \cup \{\beta_S\}$

and the set of arcs $U=R \cup N_1 \cup N_2$ where

R is the set of task-arcs

N_1 is the set of linking-arcs

N_2 is the set of source and sink-arcs.

The CSP is now the problem of covering (once only) every arc in R by paths from β_S to α_R of time-length less than or equal to T and having a minimum total cost.

5- CSP's Extensions.

At the end of chapter 3 ,two straightforward extensions of the problem will be considered. These are:

5-1 The Multiple Depot CSP (DCSP).

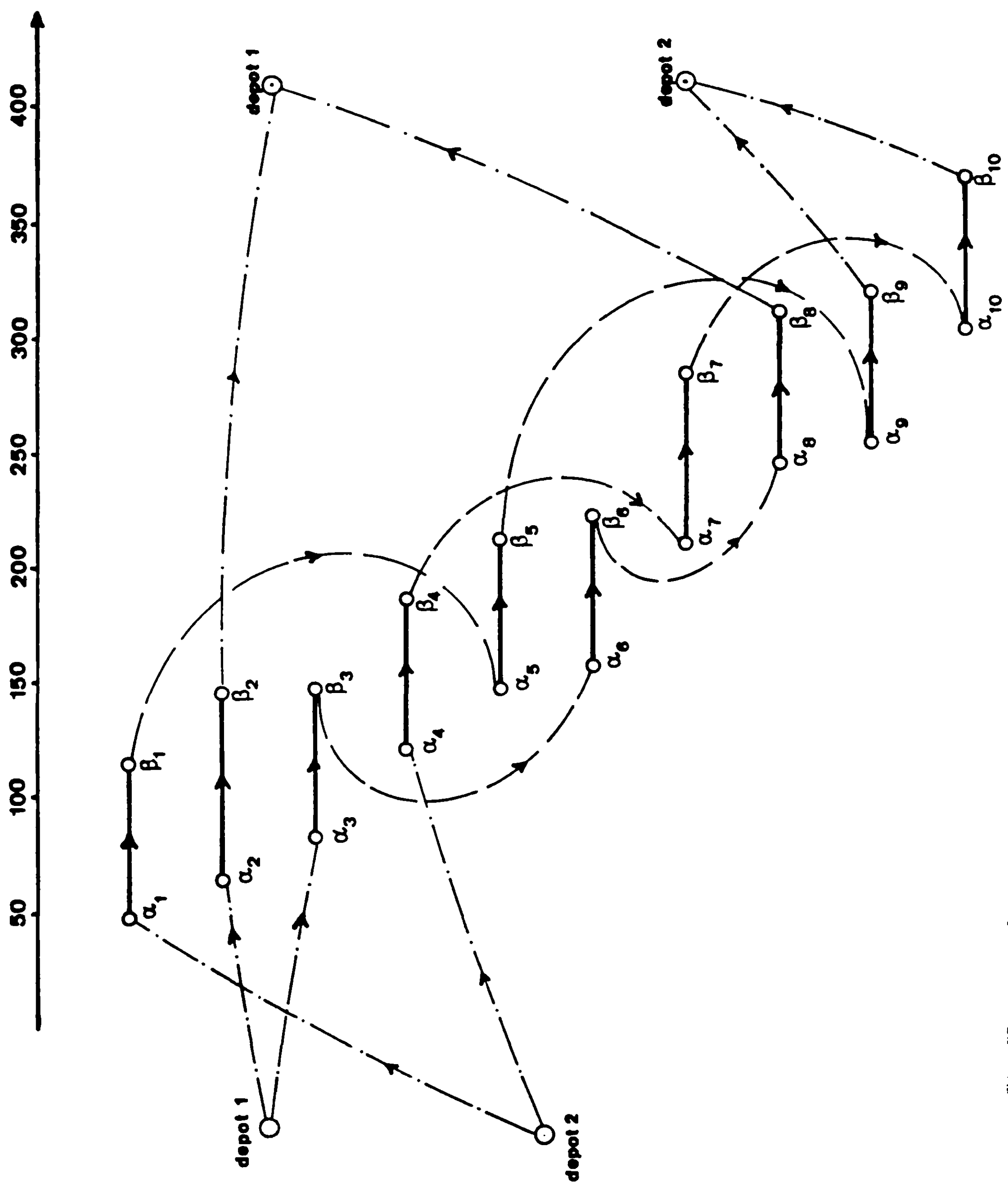
When considering the CSP we implicitly assumed that there was a single crew base out of which all the tasks were serviced. In some real problems,we might have several crew bases with known capacities ie each crew base can house no more than a given number of vehicles(used by the crews). Also each crew should return at the end of the trip to the crew base from which it has first departed. This problem that we will call multiple depot CSP (DCSP) [35] is harder than the CSP.

Consider the DCSP defined in table 1-8 and whose work duty period is assumed to be 270 minutes. Assuming we have 2 depots,a feasible solution involving 4 crews will look like the one depicted in graph G of figure 1-7. The costs of the linking-arcs are taken to be equal to their durations.

Table 1-8 : A 10 Task-DCSP.

Task	Starting time (minutes)	Finishing time (minutes)
1	50	110
2	65	140
3	80	140
4	125	185
5	150	210
6	160	220
7	215	275
8	250	310
9	260	320
10	305	365

Figure 1-7 : A Feasible Solution for the 10 Task-2 Depot DCSP.



5-2 The Airline Crew Scheduling Problem with Rest Periods.

This problem which is a pure extension of the CSP is encountered mainly in the airline industry.

The main assumptions on which the airline CSP (ACSP) is based are :

(i) the company's planning period is one day of 24 hours ;

(ii) by the end of each day all crews must have returned to the home base.

From these two assumptions we deduced the assumption that layovers were not necessary and also that the only union regulation required is the limit on the maximum number of hours a crew can fly in one duty period.

As we said previously these two assumptions are fairly reasonable for small to medium size companies. However for most of the large companies that operate 24 hours and which cover long intercontinental flights,these assumptions cannot be made;especially when we are seeking an exact solution to the problem. Consequently,for the large airline companies we need to assume :

(iii) the company's planning period is one week(instead of one day);

(iv) All crews must come back to the home base by the end of the week;

Under these conditions,the union rules become:

(a) each crew cannot fly more than a given number of hours in one duty period;

(b) between any two consecutive duty periods there must be a layover of at least T_1 hours and at most T_2 hours ($T_1 < T_2$);

(c) Each rotation cannot last more than a given number of days.

While rule (b) guarantees a sufficient period of rest time between any two consecutive duty periods,rule (c) restricts the time spent outside the crew base.

This problem is an extension of the ACSP. It will be refered to as ACSP with rest periods and is dealt with in chapter 3.

6- The General Crew Scheduling Problem (GCSP).

In the definition of the CSP we assumed implicitly that the starting times and finishing times of each task were specified. Hence if we were given any pair of tasks we could have easily said whether they can be covered by a same crew or not.

However,the problem becomes more complicated when the starting and finishing times of some or all tasks are given as time intervals(the duration of the task remaining the same regardless of its starting time). Thus,for example,in an airline context where tasks represent flight-legs,the flight Algiers-London starting at 12.30 am and finishing at 3.00 pm would be assumed to start,say ,in the time interval 12.15am - 12.45am and terminates between 2.45pm and 3.15pm. In this case it is not always possible to say whether 2 tasks can be covered by the same crew or not. To illustrate this consider two tasks i and j with i starting within the time-interval [8.00-8.30am] and finishing within [12.00-12.30am] and with j starting within [12.20am-1.00pm] and terminating within [3.00pm-3.40pm]. The duration of task i is assumed to be 4 hours. Also assume that the finishing location of task i is the starting location of task j. If in the final schedule we decided that task i starts at 8.00am then it would be possible to link it with task j since i will end at 12.00am and j cannot start before 12.20am. Whereas if task i starts at 8.30am and task j starts at 12.20am then it will be impossible to link the two tasks. Consequently the construction of the acyclic graph G representative of a CSP is no longer valid.

Considering a crew and the corresponding vehicle as a single package,the above problem gets more complicated if we allow the company to have several types of vehicles with different characteristics(these are mainly range restrictions on the vehicles). If each task can be covered by only one specific type of vehicle then the problem will result in solving M different CSP's (with time-windows) where M is the number of types of vehicles. However, complications arise when we assume that

Figure 1-8 : Network G_T representing a 7 Task- GCSP.

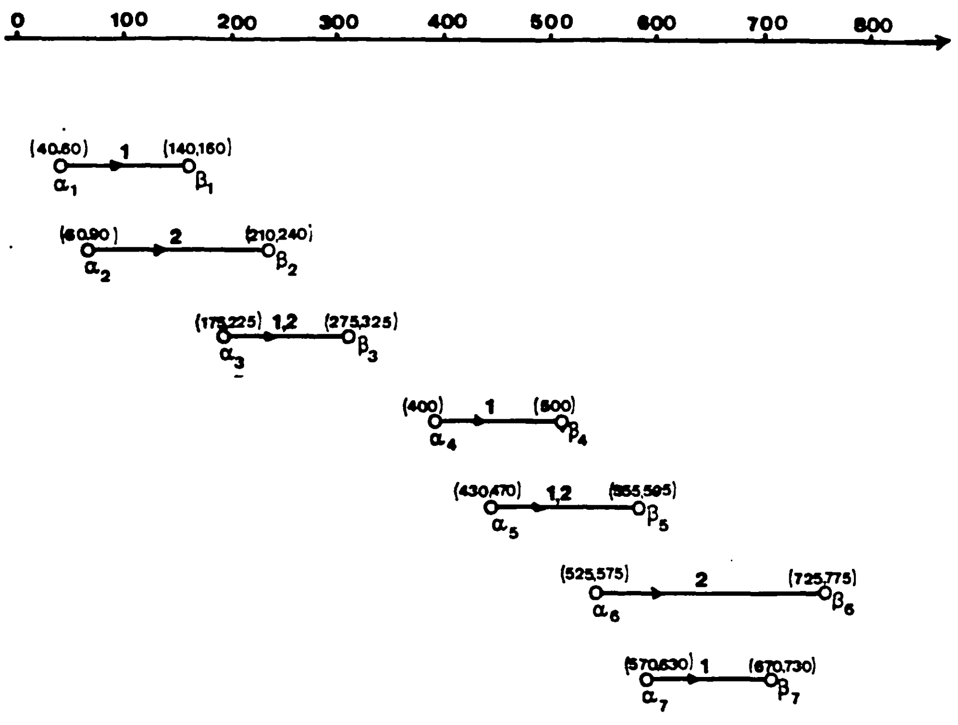


Table 1-9 : A 7 Task-GCSP .

Task	Window Starting time (minutes)	Duration time (minutes)	Vehicle Type
1	40-60	100	1
2	60-90	150	2
3	175-225	100	1;2
4	400-400	100	1
5	430-470	150	1;2
6	525-575	200	2
7	570-630	100	1

each task has, in fact, several types of vehicles by which it can be serviced. For example, in the airline context the flight Algiers-Paris can be covered by a B-737,a B727 or an A320 whereas the flight Algiers-London can be serviced only by a B737.

We will refer to this problem as the general crew scheduling problem(GCSP). It consists of a CSP in which the tasks have flexible(not fixed) due dates (ie starting times and finishing times) and in which several types of vehicles of differnt characteristics are available to service all tasks.

To fix the ideas consider the GCSP represented in table 1-9. We assume we have 2 types of vehicles for 4 crews. Also the work duty period is assumed to be 450 minutes. A graph G_T representing only the task-arcs is depicted in figure 1-8 . The figures above the task-arcs represent the vehicles'types and those at the beginning and terminal vertices of a task arc are the time intervals for the possible start-time and finish time of the arcs. Clearly the linking-arcs cannot be represented. Thus,if we consider tasks 1 and 4 it will be impossible to link them if task1 starts at 40 minutes,but if task 1 starts any time after 50 minutes then a path that includes both tasks will be feasible. For this reason, the graph's representation presented in section 2-1 cannot be applied to this problem.

7- Solving the CSP.

The airline industry is the area in which a lot of effort has been made to solve the crew scheduling problem. The importance of solving exactly the ACSP or at least to improve the current best known solution is better understood when we know that crew costs represent 10% to 15% of the company's total cost. As mentioned in [75],an improvement of 1% in the current solution of the ACSP in Alitalia ,corresponds to a profit of more than half million dollars per year.

For the last twenty five years,many authors have tried to tackle the problem using one of the two approaches :

(a) **Mathematical techniques** : although this approach provides the exact solution of the ACSP it is very limited in that only small to medium size problems,of up to 150 flights per week [113,114] can be solved. For a small company this is a reasonable size problem but for larger companies which can have up to 1000 flights per week [5] it is very small.(see Arabeyre et Al [2] for a very good account of the ACSP).
(b) **heuristic algorithms** : the advantage of this approach over the first one is that it can handle real life problems for small and large companies. However the solution produced is approximate and in some cases they can be even worse than manually produced ones.

Progress is being made,nowadays,in improving the efficiency of the heuristics [4,5,6,88] and in devising mathematical techniques to solve larger ACSP's[2,81].

7-1 Exact Methods for Solving the ACSP.

Several mathematical programming formulations have been suggested for the ACSP. The oldest and most dominant one is the set partitioning /set covering formulation [138] which is as follows :

$$\text{Min} \sum_{j \in \Omega} c_j x_j \qquad (1\text{-}8)$$

subject to :

$$\sum_{j \in \Omega} a_{ij} x_j = 1 \quad \text{for each flight-leg } i \qquad (1\text{-}9)$$

$$x_j \in \{0,1\} \qquad (1\text{-}10)$$

where Ω is the set of all rotations

$a_{ij} = 1$ if flight-leg i is covered by rotation j ;

$= 0$ otherwise ;

and c_j is a cost associated with rotation j.

Constraints (1-9) express the fact that each flight-leg must be covered by one and only one rotation. If we replace (1-9) by :

$$\sum_{\Omega} a_{ij}x_j \geq 1 \qquad \text{for each flight -leg } i \qquad (1\text{-}11)$$

the problem obtained is a set covering problem (SCP) and constraints (1-11) will express the fact that each flight-leg should be covered by at least one rotation. In other terms,a flight-leg can be covered by more than one rotation. This is a real life situation refered to as "deadheading" in which the crew travels as passengers in order to position themselves to start a new flight-leg or to return to the home base.

The procedure for solving the above SPP or SCP consists of the following steps:

(i) **Generating all the feasible rotations**: First, all the flight-legs are joined together to form flying duty periods. The duty periods and the layovers are combined together to form a rotation. Hence, a rotation will be a sequence of the form(duty period,layover,duty period,...,layover,duty period). Both duty periods and layovers are formed according to union regulations.

(ii) **Reducing the number of rotations** : In the previous step,the number of feasible rotations generated for a real life ACSP will run into thousands if not millions. Hence the reduction step is necessary so as to be able to solve a reasonable size problem.This reduction can be done by either :

(a) using dominance and logical comparison ; or

(b) put a limit on the maximum length of a rotation or eliminate all rotations with layovers at an undesirable station .

(iii) **Solving the SPP (or SCP)[112].**

Christofides and Paixao [44] proposed a very effective technique to solve large scale SCP's. This technique is based on the state-space relaxation method [43]. It is reported in [44] that SCP's of up to 400 rows and 4000 columns have been solved to optimality. Also,it is worthwhile consulting the SPP survey of Balas and Padberg [14]. As far as the ACSP is concerned,the solution techniques used up till now consist of the cutting-plane methods[68],the implicit enumeration procedures[70,111] and group theoretic approaches[134,140,141]. In the cutting-plane method the most used packages are based on Gomory's algorithm[80] and Martin's accelerated euclidean algorithm [116]. This latter one proved to be quite satisfactory and some ACSP's of up to 900 flight-legs (formulated as SPP) could be solved to optimality[75].

It is clear that the size of the matrix of the SPP formulation of the ACSP depends mainly on the planning period. Two factors play an important role in the choice of the planning period. These are the period of time in which the flight-legs are repeated and the period of time over which the union regulations operate.

For most airline companies(mainly the small and medium sized ones),the flights repeat daily. Hence a planning period of 24 hours is sufficient to tackle the problem. However,the largest companies which cover intercontinental flights that can last up to 12 hours need to perform weekly schedules. But even for some of these companies,because of the large size of the problem,the ACSP is first formulated on a daily basis. For this purpose,the company assumes complete daily periodicity and determine the minimum cost set of rotations covering each flight. Repeating each rotation on each day will enable it to cover all flights.

7-2 Heuristic Algorithms for the CSP.

In this section we present some heuristics for solving approximately the CSP. The first two heuristics are widely used because of their efficiency and also because they are easy to code. These will be described in detail and illustrated on examples.

7-2.1 The Concurrent Scheduler.[35].

This is a simple intuitive approach that has proved quite successful in practice for solving a variety of constrained scheduling problems. In our context ,the algorithm can be described as follows :

Step 1 : Number the tasks in an increasing order of their starting time. Assign task1 to crew1.

Step 2 : For k=2 to the number of tasks: If it is feasible to assign task k to an existing crew then assign it to the crew that involves the minimum linking-arc cost. Else, create a new crew and assign task k to it.

Depending on whether the CSP or one of its extensions are involved,the appropriate feasibility check would be applied in step 2.

This algorithm is in fact a member of the class of greedy heuristics which have become popular in recent years because of their simplicity[110]. Also ,it is reported in [33] ,that this algorithm is widely used in practice.

Example .

Now let us apply the concurrent scheduler to the 10 task-CSP represented in table 1-8. We assume the work duty period to be equal to 200 minutes . The cost c_{ij} of a linking-arc (β_i,α_j) is taken to be equal to its time duration and all other costs are 0.

Step 1 : - The tasks are already ordered ;

- $C_1 = \{1\}$

Step 2 : * k=2

- Task 2 overlaps with task 1 ; Hence $C_2 = \{2\}$

* k=3

- Task 3 overlaps with both task 1 and task 2 ; Hence $C_3 = \{3\}$

* k=4

- Task 4 overlaps with tasks 2 and 3 ;

- Task4 can be linked with task 1 ; Hence $C_4 = \{4\}$

* k=5

- Task 5 overlaps with task 4 ;
- Task5 can be linked only with one of tasks1,2 or 3 ;
- Choosing the minimum cost link we get $C_3=\{3,5\}$

* k=6

- Task 6 overlaps with both tasks 4 and 5;Hence it can belong to neither C_1 nor C_3;
- Task 6 can be linked only with task 2 ; Hence $C_2 = \{6\}$

* k=7

- Task 7 overlaps with task 6; Hence it cannot belong to C_2;
- Task 7 cannot be linked with task 1 because of the restriction on the work duty period; Hence it cannot belong to C_1;
- Task 7 can be linked with both tasks 3 and 5; Hence $C_3 = \{3,5,7\}$

* k=8

- Task 8 overlaps with task 7; Hence it cannot belong to C_3;
- Task 8 can be linked with neither task 1 nor task 2 because of the restriction on the work duty period; Hence it can belong neither to C_1 nor to C_2;
- Hence $C_4 = \{8\}$

* k=9

- Task 9 overlaps with both tasks 7 and 8; Hence it can belong to neither C_3 nor C_4;
- Task 9 can be linked with neither task 1 nor task 2 because of the restriction on the work duty period; Hence it can belong neither to C_1 nor to C_2;
- Hence $C_5 = \{9\}$

* k=10

- Task 10 overlaps with both tasks 8 and 9; Hence it can belong to neither C_4 nor C_5;

- Task 10 can be linked with none of tasks 1 , 2 or 3 because of the restriction on the work duty period; Hence it can belong to none of C_1,C_2 or C_3;
- Hence $C_6 = \{10\}$

The heuristic solution to the 10 task-CSP is :

Tasks 1 and 4 are assigned to crew 1 ;

Tasks 2 and 6 are assigned to crew 2 ;

Tasks 3,5 and 7 are assigned to crew 3 ;

Task 8 is assigned to crew 4 ;

Task 9 is assigned to crew 5 ;

Task 10 is assigned to crew 6 ;

The total cost is 345.

7-2.2 Two Step Approaches.[35].

This heuristic concerns only the multiple depot CSP (DCSP). This problem can be viewed as crew scheduling/clustering problem in the sense that the output is a set of crew schedules clustered by depot. This interpretation suggests two classes of approaches :

(i) the first one consists of clustering the tasks and then schedule the crews over each cluster;

(ii) the second one consists of scheduling the crews and then cluster the tasks. Both approaches are widely used in practice[33].

It is worthwhile mentioning at this point that costs are attached for bringing a crew from the depot to the tasks' starting place and from the tasks' finishing place to the depot.

As far as the first type heuristic is concerned,a weight that measures the proximity of the task to the corresponding depot is associated to each task/cluster pair. Given these weights,an assignment of tasks to clusters(depots) that minimizes the total weight can be found by solving a simple transportation problem. The CSP is then solved over each cluster using the concurrent scheduler. The second type heuristic consists of first solving the CSP over the entire task set. Then a transportation problem that assigns entire crew schedules to depots is set-up in the same manner as above.

These two approaches are in fact similar to the vehicle routing approaches[22,76,99,120] (see also [42] or [109] for a definition of the vehicle routing problem) in that in both the routing and scheduling problems we either group the required tasks into clusters,one cluster representing each depot,and then solve a single depot routing or scheduling problem or we form routes or schedules first and then assign these routes or schedules to the appropriate depot.

Now, let us apply the second approach,which consists of clustering first the tasks,to the DCSP represented by the graph of figure 1-7. The work duty period is assumed to be 200 minutes,the costs associated with the linking-arcs are equal to their time duration,a zero cost is incurred to each task-arc. The costs from the depots to the tasks' starting places and from the tasks' finishing places to the depots are given in table1-10.

The first step in this second heuristic consists of first solving the CSP. This can be done using the concurrent scheduler. The heuristic solution as found in section 7-2.1 is:

$$C_1=\{1,4\}\ ;\ C_2=\{2\}\ ;\ C_3=\{3,6\}\ ;\ C_4=\{7,10\}\ ;\ C_5=\{8\}\ ;\ C_6=\{9\}$$

where C_j (j=1,6) is the crew schedule and the numbers 1,2,...,10 are the tasks. Now let us give a weight w_{ji} to each crew-schedule C_j, with

$$w_{ji} = P(d_i,\alpha_j) + P(\beta_j,d_i) \qquad i=1,2\ ;\ j=1,..,6$$

where α_j(resp. β_j) is the starting place (resp. finishing place) of the first (resp. last) task of crew schedule j ; d_i is depot i ; P(m,k) is the cost from m to k (given by table 1-10).

The second step of the algorithm consists of solving a transportation problem that will minimize the costs of assigning the crew schedules to the depot. This can be performed by solving the minimum cost network flow problem of figure 1-9.

Table 1-10 : Costs of the Source and Sink-Arcs

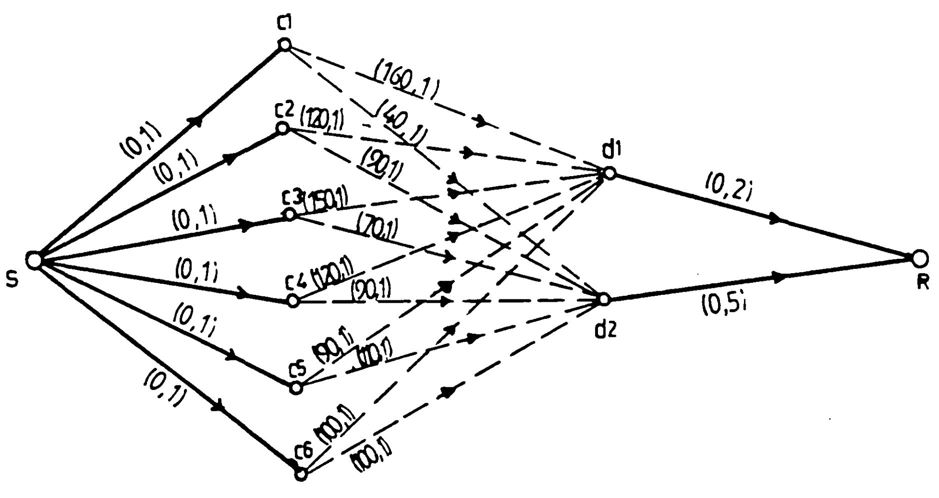
Task	Place	Depot1	Depot2
1	α_1	80	20
	β_1	40	70
2	α_2	40	70
	β_2	80	20
3	α_3	60	30
	β_3	30	80
4	α_4	60	30
	β_4	80	20
5	α_5	40	70
	β_5	60	30
6	α_6	80	20
	β_6	90	40
7	α_7	80	20
	β_7	30	80
8	α_8	30	80
	β_8	60	30
9	α_9	40	70
	β_9	60	30
10	α_{10}	90	40
	β_{10}	40	70

S and R are respectively a dummy source and sink . The C_i's represent the crew schedule and the d_i's represent the 2 depots. The two figures (in brackets) associated with each arc give respectively the cost and capacity of the arc. We have assumed that depot 1 has a capacity of 2 vehicles and depot 2 has a capacity of 5 vehicles. The input flow is equal to the number of crew schedules ie 6. The optimal solution becomes :

crew schedules 5 and 6 are assigned to depot 1 ,

crew schedules 1,2,3 and 4 are assigned to depot 2 .

Figure 1- 9 : Network Associated with Table 1-10.



7-2.3 An Interchange Heuristic[137].

This procedure can be viewed as an adaptation of the 2-opt algorithm for the TSP [107]. Assuming that a starting solution is already available (this can be done using the concurrent scheduler), the heuristic effects interchanges between the components of this schedule to improve costs. An interchange affects only 2 crew schedules,say 1 and 2. It joins the first half of crew schedule 1 with the second half of crew schedule 2 and the first half of crew schedule 2 with the second half of crew schedule 1. In this case the costs of the two new crew schedules must be compared to the costs of the two old crew schedules.

This algorithm has proved to be quite efficient for the DCSP and related problems. More details about this heuristic can be found in [137].

7-2.4 A Set Covering Based Heuristic.

This approach [142] consists of first enumerating all possible feasible rotations for a single crew. Now,let :

a_{ij} = 1 if task i is in rotation j ;
= 0 otherwise ;

and let

x_j =1 if rotation j is to be used ;
=0 otherwise .

Also let c_j be the total cost of rotation j. Then the master scheduling problem is simply the set covering problem(SCP) [112] :

$$\text{Min} \sum_j c_j x_j \tag{1-12}$$

subject to :

$$\sum_j a_{ij} c_j \geq 1 \qquad \text{all tasks i} \tag{1-13}$$

$$x_j \in \{0,1\} \tag{1-14}$$

There are 2 problems associated with solving directly the SCP. The first is that all feasible rotations could not possibly be enumerated for a large problem in a reasonable amount of time. The second is that there is no efficient exact algorithm for solving such large SCP[129].

Toregas[142] suggested the following heuristic:

Step 1 : Let S be an empty set ;

Step 2 : Generate a set C of unique (not previously generated) task-schedules, and add C to S ;

Step 3 : Solve the associated set covering problem with columns S ;

Step 4 : Retain the schedule in the optimal solution from step 3 and delete the unused schedules from S ;

Step 5 : If the time limit is exceeded,then STOP,else GO TO Step 2.

Clearly,the main step in this algorithm is step 3 which consists of solving the related SCP. Toregas claimed that the set covering problem can be solved(most of the time) by relaxing the integrality constraint and solving the associated linear program. If the solution X* is non-integral cutting-planes can be added to the linear program constraint set,and the linear program is then solved again. Few cuts are usually required to produce integral solutions,although some counter-examples have been found. An alternative way of solving the SCP is to apply any of the existing SCP's heuristics[45,87,129].

7-2.5 Orloff's heuristic [125] .

This algorithm , like the interchange heuristic , can be viewed as an adaptation of the 3-opt algorithm for the TSP [107,108] . Orloff argued that , in fact , the problem can be viewed as a special case of the travelling salesman problem[41] . Consequently Lin's heuristic can be applied directly . But before this , a feasible schedule has to be built and then Lin's heuristic [107] , which is an improvement routine on the existing solution , is applied .

The analogy between this problem and the TSP can be defined as follows : a solution to the problem consists of a set of M paths which cover all N tasks . But a solution can equivalently be considered to be a set of M cycles , centered at the depot , such that each cycle C_i (i=1,...,M) is of the form $C_i = d \rightarrow P_i \rightarrow d$ where P_i is one of the M paths that form the solution . Then there is a one-to-one relationship between feasible schedule solutions and feasible travelling salesman tours on the set N with depot d added . If two tasks i and j can not be assigned to a common crew schedule , then $c_{ij} = \infty$. Otherwise c_{ij} will be the deadhead cost of following one task by the other . C_{id} and C_{di} are large .

One of the main steps in this algorithm is to determine a good starting feasible solution . This can be performed by using one of the two previous heuristics . Alternatively we can use the "swapping routine " of Orloff [126] .

In this routine a sequence of matching problems [39] is solved to build schedules by optimally matching segments together . The matching problem starts with the set of N tasks and generates a set of minimal cost task pairs and singletons , by matching each task i to another task j , or matching a task i to itself . These task pairs and singletons are schedule segments . The costs of matching these new tasks (schedule segments) are then calculated and a new matching problem is solved which generates a set of minimal cost task pairs , i.e. , task quadruples , triples , pairs and singletons . The costs of matching these new tasks together are then calculated , and a new set of minimal cost tasks are generated . This procedure continues until no new matchings are obtained .

If matching two schedule segments is infeasible then the cost is set to infinity . A matching may be infeasible because of one of the three reasons :

(i) The time length of the resulting crew schedule exceeds the work duty period .

(ii) The earliest finish time of one schedule segment may be greater than the latest start time for the other segment .

(iii) None of the vehicles' types that can cover one schedule segment can cover the other schedule segment .

Orloff reported in his paper that this algorithm has proved to be very efficient in practice . Unfortunately, he did not report any computational results .

CHAPTER 2

A DIRECT FORMULATION OF THE CSP

1- INTRODUCTION

The solution of the BCSP consists of a set of K (K is the number of crews of the BCSP) disjoint paths which cover all task-arcs of network G_u(see chapter 1). Two paths of G_u are said to be disjoint if they have no arc in common . In the network G_u of figure 1-4 the two following paths are disjoint.

$P_1 = (\beta_S,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_3)(\alpha_3,\beta_3)(\beta_3,\alpha_5)(\alpha_5,\beta_5)(\beta_5,\alpha_R)$

$P_2 = (\beta_S,\alpha_2)(\alpha_2,\beta_2)(\beta_2,\alpha_4)(\alpha_4,\beta_4)(\beta_4,\alpha_R)$

The solution of the BCSP is not always feasible for the CSP. If , for example , we consider the solution of the BCSP given in figure 1-5 and if we assume that the work duty period is 400 minutes , then this solution will not be feasible for the CSP . Thus, path P= $(\beta_S,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_4)(\alpha_4,\beta_4)(\beta_4,\alpha_5)(\alpha_5,\beta_5)(\beta_5,\alpha_R)$ which is in the solution has length 600 minutes and violates the rule that limits the number of working hours in a duty period (in this case 400 minutes). Consequently , additional constraints which express this restriction should be considered in the formulation of the problem . These constraints will be refered to as time constraints since they express a limit on the working time .

The formulation of the CSP is presented in section 2 . In fact this formulation is not based directly on network G but on another network G^a which is an extended version of G . This will also be discussed in section 2. Section 3 deals with the solution technique . At first we wanted to use a combination of two integer programming techniques , namely the cutting-planes (C-P) method [93] and the branch and bound procedure [21,67].The idea was to start solving the linear relaxation of the CSP . If the solution is optimal for the CSP we stop , otherwise we add additional constraints (cutting-planes) that will force the solution to integrality . In case the optimal solution has not been reached after a predetermined number of cuts the C-P algorithm is embedded into a branch and bound procedure . The cutting-planes were a combination of logical [92] and Gomory's cuts [80] .

Out of 101 randomly generated CSP's , 81 were solved optimally directly by an LP package . For the rest we needed to use only the C-P algorithm to reach to the optimal . There was no need for branch and bound . These results will be presented in section 4. Unfortunately , only problems of up to 30 tasks could be considered . This was due to the limitations imposed by the existing LP packages which could not handle such large size problems. In our attempt to solve larger problems , the Lagrangian relaxation [60] was considered . This is one of the most successful integer programming techniques . Its application to the CSP is dealt with in section 5 and the computational results are presented in section 6 .

2- THE PROBLEM FORMULATION .

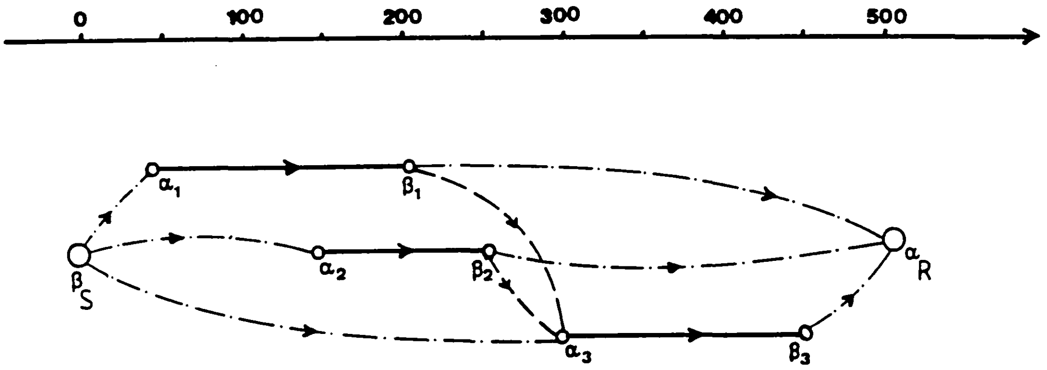
2-1 Construction of Network G^a.Algorithm A1.

As we said in the previous section , the CSP formulation presented in this chapter is based on a network G^a which is an extension of network G . The procedure for constructing such a network will be first explained on the following example :

2-1.1 Example .

Consider in figure 2.1 network G representative of a 3 task-CSP.

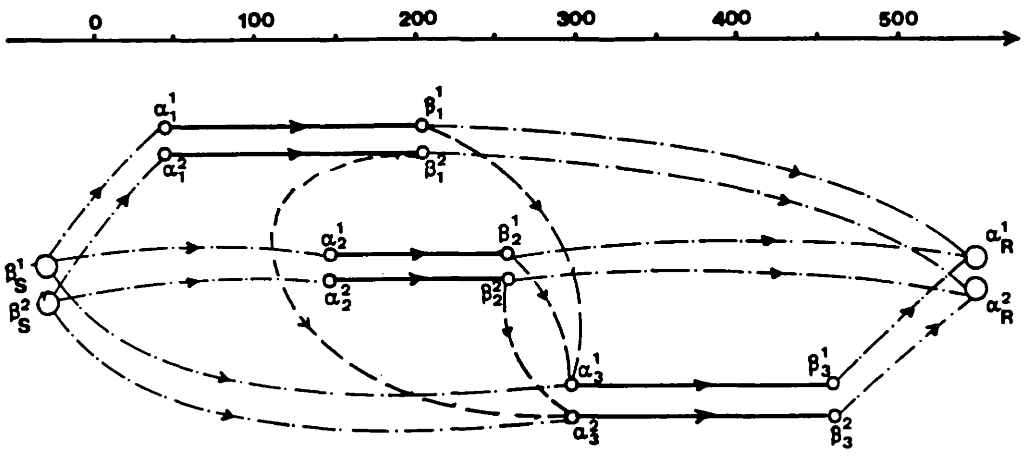
Figure 2-1 : Network G.



The working duty period is assumed to be 400 minutes and the number of paths with which we want to cover all task-arcs of G is 2 .

Now , consider task-arc (α_1,β_1) . It can be covered either by path1 or path 2. Hence , we can duplicate it into two arcs : (α_1^1,β_1^1) corresponds to the 1st path and (α_1^2,β_1^2) corresponds to the 2nd path . But in the optimal solution only one arc will be covered . Similarly, arcs (α_2,β_2) and (α_3,β_3) can be duplicated in the same manner. Assuming a super source β_S^1 (resp. β_S^2) and a super sink α_R^1 (resp. α_R^2) for path 1 (resp. 2), and joining β_S^i with α_j^i (j=1,2,3 ; i=1,2) . Also joining β_j^i (j=1,2,3) with α_R^i (i=1,2) we obtain network G^a of figure 2.2 .

Figure 2-2 : Network G^a



2-1.2 General Procedure for Constructing G^a. Algorithm A1 .

Step 1 : To each task-arc (α_i,β_i) of G there correspond K arcs (α_i^1,β_i^1) (α_i^2,β_i^2),..., (α_i^K,β_i^K) in G^a where K is the number of paths with which we want to cover G (and G^a). These arcs of G^a will be called "**task-arcs**" and have the same cost $\bar{c}_i^k$ and duration time $\bar{\tau}_i^k$ as arc (α_i,β_i) of G.

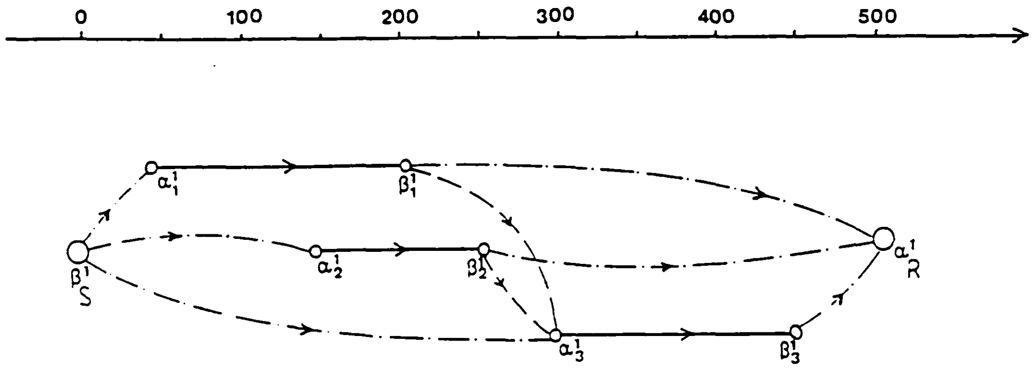
Step 2 : For each p=1,...,K, link all the task-arcs (α_1^p,β_1^p),..., (α_n^p,β_n^p) (see section 1-4, chapter 1) ; n is the number of task-arcs of G. The cost c_{ij}^p and duration time τ_{ij}^p of arc (β_i^p,α_j^p) (p=1,...,K),called "**linking-arc**', are the same as the cost and duration time of arc (β_i,α_j) of G.

Step 3 : Create K super-sources β_S^1,..., β_S^K and K super-sinks α_R^1,..., α_R^K and join each super-source β_S^p (resp. super-sink α_R^p) (p=1,...,K) with all α_i^p (resp. β_i^p) (i=1,...,n) . All the arcs (β_S^p,α_i^p) (called "**source-arcs**') and all the arcs (β_i^p,α_R^p)

(called "**sink-arcs**') have costs and duration times 0.

In the previous example (2.1.1) , G^a can be considered as the union of 2 disjoint graphs G^a_1 and G^a_2 identical to G and corresponding to paths 1 and 2 respectively . Thus G^a_1 is given in figure 2.3 and G^a_2 in figure 2.4 .

Figure 2-3 : Network G^a_1



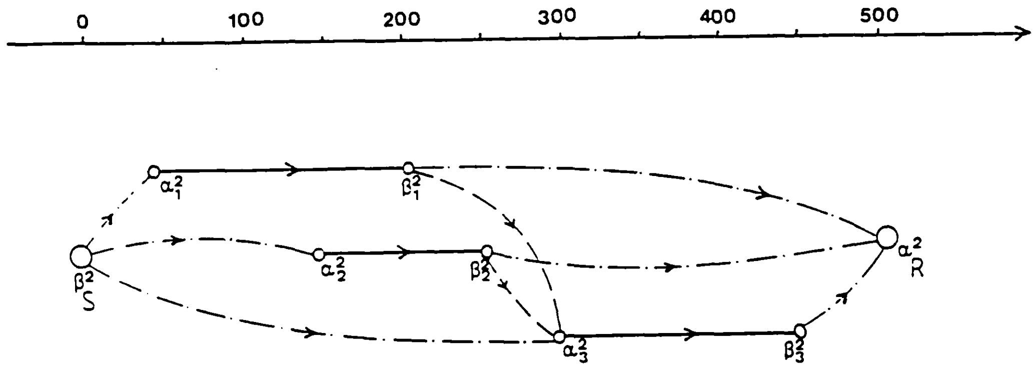


Figure 2-4 : Network G^a_2

In general G^a can be defined as the union of K disjoint graphs G^a_1,..., G^a_K all identical to G. There are as many graphs as there are paths k (k=1,...,K).

Applying algorithm A1 to network G , we obtain network :

$$G^a = (X^a = X^a_1 \cup X^a_2 , U^a = U^a_1 \cup U^a_2)$$

where X^a is the set of vertices , U^a is the set of arcs and where :

$X^a_1 = \{ \alpha^p_i \mid i=1,...,n \text{ and } p=1,...,K\} \cup \{ \alpha^p_R / p=1,...,K\}$

$X^a_2 = \{ \beta^p_i \mid i=1,...,n \text{ and } p=1,...,K\} \cup \{ \beta^p_S / p=1,...,K\}$

$U_1^a = \{ (\alpha_i^p, \beta_i^p) \mid i=1,\dots,n \text{ and } p=1,\dots,K\}$

$U_2^a = \{\text{set of all linking-arcs } (\beta_S^p, \alpha_i^p), p=1,\dots,K\} \cup$

$\{(\beta_S^p, \alpha_i^p) \mid i=1,\dots,n \text{ and } p=1,\dots,K\} \cup$

$\{(\beta_i^p, \alpha_R^p) \mid i=1,\dots,n \text{ and } p=1,\dots,K\}$

where n is the number of task-arcs of G ;

K is the number of paths with which we want to cover G ;

α_R^p and β_S^p (p=1,...,K) are the sinks and the sources ,respectively,of G^a;

The cost and duration time of each arc of G^a are described in algorithm A1 .

2-2 The Problem Formulation .

The number of paths with which we want to cover G (or G^a) is K and the number of task-arcs of G is n .

Let $V_{i,k}^+ = \{j/(\beta_i^k, \alpha_j^k) \in U_2^a\}$ and $V_{i,k}^- = \{j/(\beta_j^k, \alpha_i^k) \in U_2^a\}$

Let $x_{ijk} = 1$ if arc $(\beta_i^k, \alpha_j^k) \in U^a$ is in the optimal solution ;

$= 0$ otherwise .

The CSP can now be formulated as :

$$\min \sum_{(\beta_i^k, \alpha_j^k) \in U_2^a} c_{ij}^k x_{ijk} \tag{2-1}$$

subject to :

$$\sum_{j \in V_{i,k}^+} x_{ijk} - \sum_{j \in V_{i,k}^-} x_{jik} = \begin{cases} 0 \ , & i=1,\dots,n \\ 1 \ , & i=S \\ -1 \ , & i=R \end{cases} \qquad k=1,\dots,K \tag{2-2}$$

$$\sum_{k=1}^{K}\sum_{j\in V_{i,k}^{+}} x_{ijk} = 1 \qquad i=1,\dots,n \qquad (2\text{-}3)$$

$$\sum_{(\beta_i^k,\alpha_j^k)\in U_2^a} \tau_{ijk}x_{ijk} + \sum_{i=1}^{n} \bar{\tau}_i^k\Big(\sum_{j\in V_{i,k}^{+}} x_{ijk}\Big) \le T \qquad k=1,\dots,K \qquad (2\text{-}4)$$

$$x_{ijk} \in \{0,1\}$$

In the objective function , only the costs of the linking-arcs will be considered . There is no need for including the costs of the source and sink-arcs because each one of them has cost and duration time 0 .The choice of such an objective function is explained in section 2.3 .

Constraints (2-4) are time constraints . They ensure that , in a daily schedule , each crew cannot operate more than the working duty period fixed by union requirements . In network G^a (and G) ,they mean that the length of each one of the K paths with which we want to cover G must be less than or equal to a given length T.

The partitioning constraints (2-3) guarantee that each task must be performed once only by just one crew.

The flow constraints (2-2) express the flow conservation at each node of the graph G^a. Thus , the above formulation consists of a minimum cost network flow problem defined by (2-2) and (2-3) with the additional constraints given by (2-4).

The conservation of flow at each node of G^a could also be expressed in the following way :

$$\sum_{j\in V_{i,k}^{+}} x_{ijk} = x_{iik} \qquad \text{for } i=1,\dots,n \text{ and } k=1,\dots,K \qquad (2\text{-}5)$$

$$\sum_{j \in V^-_{i,k}} x_{ijk} = x_{iik} \qquad \text{for } i=1,\dots,n \text{ and } k=1,\dots,K \qquad (2\text{-}6)$$

$$\sum_{(\beta_s,\alpha_i) \in U^a_2} x_{sik} = 1 \qquad \text{for } k=1,\dots,K \qquad (2\text{-}7)$$

$$\sum_{(\beta_s,\alpha_R) \in U^a_2} x_{iRk} = 1 \qquad \text{for } k=1,\dots,K \qquad (2\text{-}8)$$

where $$x_{iik} = \begin{cases} 1 & \text{if arc } (\alpha^k_i, \beta^k_i) \in U^a_1 \text{ is in the optimal solution} \\ 0 & \text{otherwise} \end{cases} \qquad (2\text{-}9)$$

It is clear that constraints (2-2) are equivalent to constraints (2-5) , (2-6) ,(2-7) and (2-8) . However , for the sake of simplicity we have prefered constraints (2-2) to express the conservation of flow at each node of G^a .

2-3 Choice of the Objective Function .

In all CSP' s considered from now on we assume that the given number of crews (K) , with which we want to operate all the tasks subject to union regulations , is the minimum possible . It is clear that if K is small enough then the CSP , as defined in section 2-2 can be infeasible . For example in the airline problem , the solution that will result would be the least vehicle solution that minimizes deadhead mileage.This is a reasonable assumption if the plane capital costs are large compared to the costs attached with deadheadings. However , if the costs attached with deadheadings are of the same order as the plane capital costs , then one way of tackling the problem will be to fix the number of planes ,K, equal to different values $K_1,K_2,\dots,K_m$ (all greater than the minimum number of planes required) and for each value of K , minimize the deadhead mileage . The solution of each problem P_i (i=1,...,m) corresponding to K_i,is the sum of the minimum cost deadhead mileage and the capital costs of the K_i planes .The overall solution to the ACSP that minimizes both the capital costs and deadhead mileage is the minimum cost solution of the m ACSP's.

Now , the natural question that comes to one's mind is : How can we determine the minimum number of crews so as to cover all tasks ? This problem is an NP-complete problem [15] . Several exact and heuristic algorithms have been suggested for this problem [117,123,146].

In our case we are using the greedy heuristic algorithm that is given in section 2-4 . This algorithm has proved to be very effective (as it can be seen from the results presented in section 4).

2-4 A Greedy Algorithm for Finding the Minimum Number of Crews Required to Cover all Tasks of the CSP. Algorithm A2 .

2-4.1 Definition .Let I be a set of tasks of the CSP and assume that all the tasks of I are to be covered by a single crew P . A task $j \in I$ can be covered by a crew p if and only if for any $i \in I$, crew P can cover both tasks i and j (ie , assuming i precedes j , $ST_j > FT_i + \Delta(FL_i, SL_j)$ and $FT_j - ST_i \leq T$.

2-4.2 Algorithm A2 .

Let I be the list of tasks that will be covered by the current crew and p be the minimum number of crews required to cover all tasks of the CSP . Also let F be the set of all tasks that have not been assigned to crews yet and let n_f be the cardinality of F. The algorithm is as follows :

Step 0 : $I=\emptyset$, F= set of all tasks of the CSP ,p=1 , $n_f = |F|$, j=0 ; GOTO STEP 1;

Step 1 : j=j+1 ; If the jth task (i_j,say) of F cannot be covered by crew p, GOTO STEP 2 . Else , set $I = I \cup \{ i_j \}$ and GOTO STEP 2 ;

Step 2 : If $j = n_f$ (ie all tasks of F have been scanned) GOTO STEP 3 ; Else GOTO STEP 1 ;

Step 3 : F=F-I (ie remove all tasks of I from F) . If F= $\emptyset$ STOP ; Else p=p+1 , $n_f = |F|$, I=$\emptyset$, j=0 ; GOTO STEP 1.

For the example of table 1-6 the value obtained by algorithm A2 is 3 . (The work duty period has been assumed to be 400 minutes).

2-4.3 Complexity of Algorithm A2 .

In step 1 , C(j-1) comparisons are required for each j ($j=1,\ldots,n_f$) where C is a constant . Hence for each crew p , all the steps require $Qn_f(n_f-1)/2$ operations at most (Q is a constant).

Assuming that in the worst case we need n crews to cover all n tasks of the CSP, the algorithm will require (in that case) $Q'n^3$ operations (Q' is a constant). Thus , we can say that algorithm A2 runs in time $O(n^3)$, where n is the number of tasks of the CSP.

3- The Solution Technique : A Cutting-Planes Algorithm .

3-1. Introduction.

The technique used to solve the CSP is a combination of logical and Gomory's cutting-planes (C-P).

In 1958 , *Gomory [79]* published the first paper on cutting-planes theory.This is one of the solution techniques of integer programming (I-P)[20,68,139].It is described in detail in [68] .*Gomory [80]* proved theoretically that his C-P algorithm converges (ie the optimal solution of the problem is found) after a finite number of iterations.It has been noticed by several authors [132] however that for most of the medium or large size IP problems the algorithm did not converge fast enough.The explanation of this often poor performance of the C-P algorithm was given by many authors [132].What happens in fact is that the convergence of the algorithm to the integer optimal solution is very rapid during the first few cuts . Subsequently , this increase slows down and the cutting-planes become ineffective.. making the convergence to the optimal solution very slow. Figure 2-5 shows the convergence of the Gomory's algorithm applied to a 10 task-CSP representing a linear problem of 49 constraints and 200 variables . As it can be seen , the algorithm reaches 95% of the optimal solution after the addition of 5 cuts only.However at the 13th cut the optimal solution has not been reached yet. Actually the algorithm reached 98% of the optimum after the addition of 30 cuts.

Figure 2-5 : Convergence of the Gomory C-P Algorithm Applied to a 10 Task-CSP.

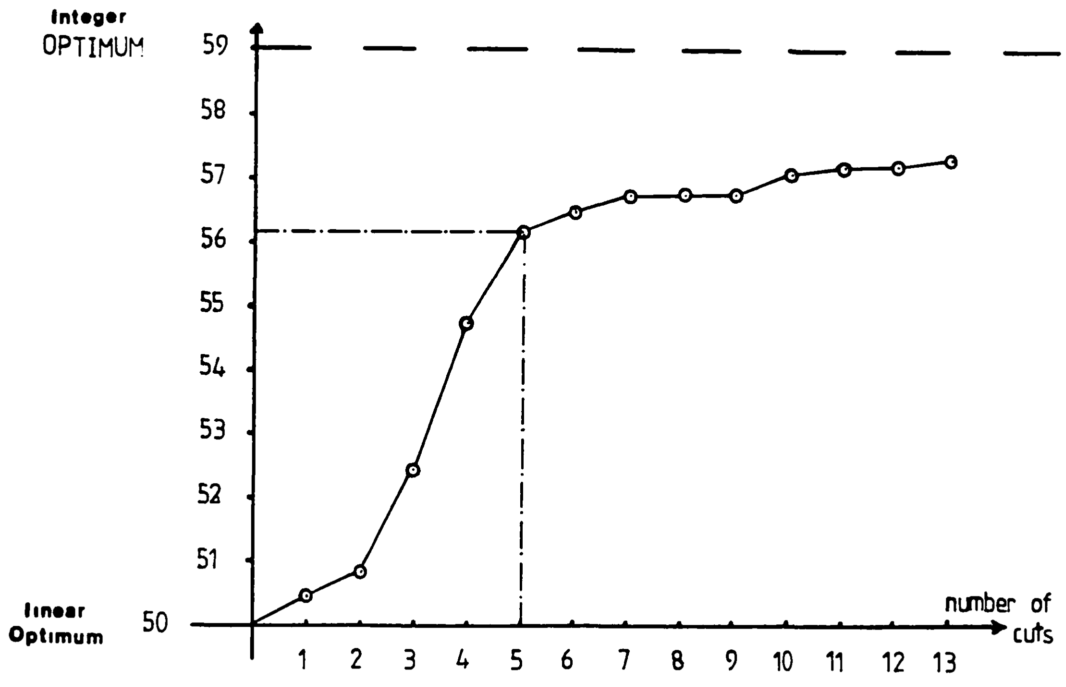


Table 2-1 : A 10 Task-CSP

Task	Starting time (minutes)	Finishing time (minutes)
1	22	64
2	85	169
3	151	255
4	193	230
5	266	324
6	322	364
7	352	398
8	385	469
9	412	492
10	493	530

Other types of C-P algorithms were suggested by various authors [8,9,77,78,148,150]but none of them could produce satisfactory results . Their common problem was their slow convergence when approaching the optimal solution. This series of unseccessful results did not mean that the C-P algorithm should be rejected .

In 1973 , *Rubin [132]* suggested a clever way of using the C-P method.He argued that since the C-P algorithm is very efficient during the first few cuts then at the precise point where the cutting-planes become redundant this algorithm should be stopped and embedded into a branch and bound procedure [7,20,50,101] in order to accelerate the convergence to the optimum.For example if we consider figure 2.5 which shows the convergence of the Gomory's C-P algorithm when applied to the 10 task-CSP of table2-1 we can see that after the addition of cut 10 , the convergence to the optimum becomes very slow . Hence Gomory's C-P algorithm can be stopped at this point and embedded in a branch and bound procedure .It has been noticed that when the cutting-planes are combined with branch and bound they perform quite well [12,49].

One year later , *Owen [124]* proposed a new type of cutting-planes called logical or disjunctive cuts [10,13] . These are linear equalities(or inequalities) that express logical conditions derived from the problem . This simple idea can be illustrated as follows : consider the two following constraints

$$2x_1 - x_2 + 5x_3 \geq 3 \tag{2-10}$$

$$3x_1 + 4x_2 - 2x_3 \geq 2 \tag{2-11}$$

the following constraint is the strongest common weakening of (2-10) and (2-11)

$$\max(2,3)x_1 + \max(-1,4)x_2 + \max(5,-2)x_3 \geq \min(3,2)$$

ie $$3x_1 + 4x_2 + 5x_3 \geq 2 \tag{2-12}$$

Assuming that (2-10) and (2-11) are cutting-planes for a certain problem and assuming that we do not know which one is valid we can be sure that (2-12) is a valid cut .It is called a disjunctive or logical cut .

Unlike all the previous cuts the logical cuts depend entirely on the structure of the problem considered and the way in which they are derived may vary from one problem to another . A very good account of the theory of disjunctive cuts has been given by *Jeroslow [92]* . Also as far as the practical side of these cutting-planes is concerned we should mention the excellent work of *Balas and Padberg [14]* on the set partitioning problem.In that paper , among the several I-P techniques that have been suggested the application of logical cuts has proved to be very effective.

The derivation of the logical cuts for the CSP is explained in section 3-2.

Inspired by *Rubin's* idea we first wanted to combine the cutting-plane algorithm (ie logical and *Gomory's*) with the branch and bound procedure . Fortunately , we did not need to go that far . Thus, 80% of the 101 problems (of size up to 30 tasks) randomly generated were solved just by an LP package. For the rest we only needed to apply the combination of the two types of C-P in order to get convergence to the optimum. There was no need for branch and bound for these 101 problems .

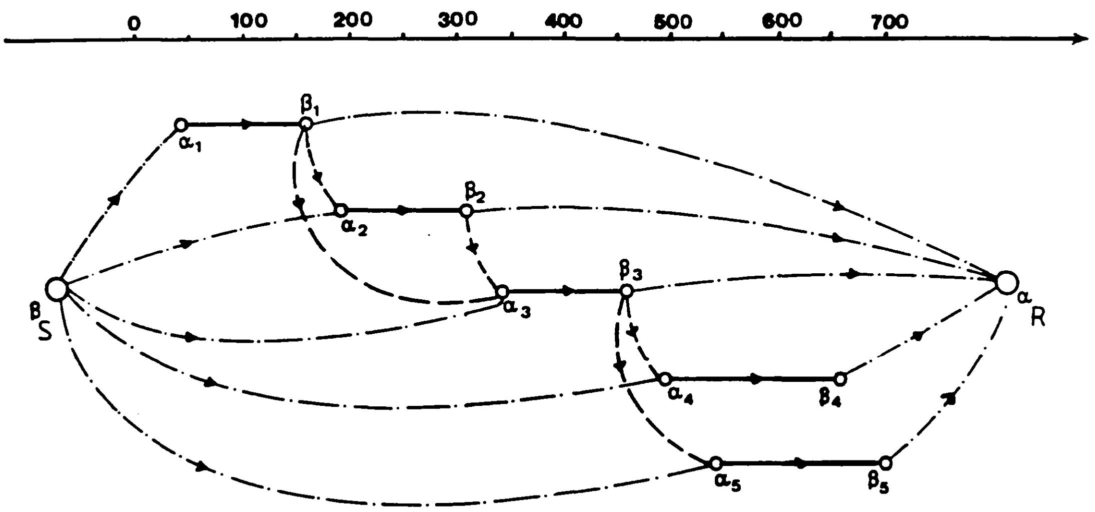
3-2. Derivation of the Logical Cuts .Algorithm A3.

As far as the theory of Gomory's cutting-planes is concerned the interested reader is refered to [68,139].In this section we will be dealing with the derivation of the logical cutting-planes .The idea of deriving the logical cuts for the CSP is very simple and will be first explained on the following example.

3-2.1 Example .

Consider the 5 task-CSP represented by network G of figure 2.6.Let us assume that the work duty period is 400 minutes and that we want to cover G with 3 paths .

Figure 2-6 : Network G



It is clear that the task-arcs (α_4,β_4) and (α_5,β_5) cannot be covered by the same path. Also neither of them can belong to a path starting with task-arc (α_1,β_1) .This is simply because the length of such a path would be greater than the work duty period . Consequently (α_1,β_1), (α_4,β_4) and (α_5,β_5) must be covered by 3 different paths .

Hence we can , from the start , assign task-arc (α_1,β_1) to path 1, task-arc (α_4,β_4) to path 2 and task-arc (α_5,β_5) to path 3. Expressing this logical condition in mathematical terms leads us to the following logical cuts :

$$x_{1R1} + x_{121} + x_{131} = 1 \qquad (2\text{-}13)$$

$$x_{4R2} = 1 \qquad (2\text{-}14)$$

$$x_{5R3} = 1 \qquad (2\text{-}15)$$

where x_{ijk} is as defined in section 2-2.

Now , let us consider task-arc (α_2,β_2) : None of task-arcs (α_4,β_4) and (α_5,β_5) can be covered by a path that covers (α_2,β_2) . Hence we can say that task-arc (α_2,β_2) will neither be in path 2 nor in path 3 . The logical cuts corresponding to these logical conditions are :

$$x_{2R2} + x_{232} = 0 \qquad (2\text{-}16)$$

$$x_{2R3} + x_{233} = 0 \qquad (2\text{-}17)$$

Now we give the general procedure for deriving the logical cuts .

3-2.2 Definition .

Given a network G , representative of a CSP , and its set B of task-arcs we define an independent set $I \subset B$ as being a set of task-arcs no two of which can be covered by a same feasible path (ie a path with length less than or equal to the work duty period) . Also we define a maximum independent set (MIS) as being an independent set with maximal cardinality.

In example 3-1 task-arcs (α_2,β_2) and (α_4,β_4) of G form an independent set and task-arcs (α_1,β_1), (α_4,β_4) and (α_5,β_5) form a MIS .

3-2.3 Algorithm A3 .

The input to the algorithm is a network G representing a CSP .

Step 1 : Determine a maximum independent set $I \subset B$.(B is the set of all task-arcs of network G) . Let $I = \{ (\alpha_{i_1},\beta_{i_1}),\ldots, (\alpha_{i_h},\beta_{i_h})\}$

Step 2 : Assign each arc $(\alpha_{i_k},\beta_{i_k})$ (k=1,...,h) of I to a different path P_k (k=1,...,h) ie add the following logical cuts to the problem formulation

$$\sum_{(\beta_{i_k},\alpha_j)\in U_2^a} x_{i_k j k} = 1 \qquad k=1,\ldots,h$$

where x_{ijk} and U_2^a are as defined in section 2-2.

Step 3 : Let $I' = B - I = \{ (\alpha_{i_{h+1}},\beta_{i_{h+1}}) ,\ldots, (\alpha_{i_n},\beta_{i_n}) \}$. For each arc $(\alpha_{i_m},\beta_{i_m})$ m=h+1,...n, determine the list L_m of all task-arcs of I that cannot be covered by a path that covers $(\alpha_{i_m},\beta_{i_m})$.

Step 4 : For each $(\alpha_{i_m},\beta_{i_m})$ for which $L_m \neq \emptyset$,let $P_{t_1},\ldots,P_{t_m}$ be the set of paths associated with the task-arcs of L_m. Add the following logical cut :

$$\sum_{k=1}^{m} \sum_{(\beta_{i_m},j)\in U_2^a} x_{i_m,j,t_k} = 0$$

This means that task-arc $(\alpha_{i_m},\beta_{i_m})$ will be covered by none of the paths $P_{t_1},\ldots,P_{t_m}$.

The algorithm is stopped when all arcs $(\alpha_{i_m},\beta_{i_m})$ for which $L_m \neq \emptyset$ have been considered .

3-2.4 Example .

Consider network G of figure 2.6 .

Step 1 : maximum independent set $I = \{ (\alpha_1,\beta_1), (\alpha_4,\beta_4),(\alpha_5,\beta_5) \}$;

$B= \{ (\alpha_1,\beta_1), (\alpha_2,\beta_2), (\alpha_3,\beta_3),(\alpha_4,\beta_4),(\alpha_5,\beta_5) \}$

Step 2: for arc (α_1,β_1) : $x_{121} + x_{131} + x_{1R1} = 1$

for arc (α_4,β_4) : $x_{4R2} = 1$

for arc (α_5,β_5) : $x_{5R3} = 1$

where α_R is the sink.

Step 3 : $I' = \{ (\alpha_2,\beta_2), (\alpha_3,\beta_3) \}$

* Consider task-arc (α_2,β_2) : $L_2 = \emptyset$

can (α_2,β_2) be in a same path with (α_1,β_1) ? Yes ; $L_2 = \emptyset$

can (α_2,β_2) be in a same path with (α_4,β_4) ? No ; $L_2 =\{ (\alpha_4,\beta_4) \}$

can (α_2,β_2) be in a same path with (α_5,β_5) ? No ; $L_2 =\{ (\alpha_5,\beta_5), (\alpha_4,\beta_4)\}$

* Consider task-arc (α_3,β_3) : $L_3 = \emptyset$

can (α_3,β_3) be in a same path with (α_1,β_1) ? Yes ; $L_3 = \emptyset$

can (α_3,β_3) be in a same path with (α_4,β_4) ? Yes ; $L_3 = \emptyset$

can (α_3,β_3) be in a same path with (α_5,β_5) ? Yes ; $L_3 = \emptyset$

Step 4 : * Consider task-arc (α_2,β_2) : $L_2 \neq \emptyset$

Add the following cut to the problem :

$$x_{232} + x_{233} + x_{2R2} + x_{2R3} = 0$$

* Consider task-arc (α_3,β_3) : $L_3 = \emptyset$. STOP .

3-3 A Heuristic Algorithm for Finding a MIS in Network G. Algorithm A4.

The input to the algorithm is a network G representing an n task-K crew CSP .

Step 0 : Let M be a maximal independent set of task-arcs of G . M=$\{ (\alpha_1,\beta_1) \}$; i=1

GOTO STEP 1 ;

Step 1 : i=i+1.

If there is no linking-arc joining task-arc (α_i,β_i) with one of the task-arcs of M then GOTO STEP 2 ;Else task-arc (α_i,β_i) cannot belong to M. GOTO STEP 3 ;

Step 2 : Add task-arc (α_i,β_i) to M ; M=M U $\{ (\alpha_i,\beta_i) \}$; GOTO STEP 3 ;

Step 3 : If i=n all task-arcs of G have been scanned , STOP ; Else GOTO STEP 1.

In step 1 of Algorithm A4 ,the statement "if there is no linking-arc joining task-arc (α_i,β_i) with one of the task-arcs of M " means "if no crew can cover task i with one of the tasks corresponding to M".

It is easy to show that algorithm A4 runs in time $O(n^2)$.

3-4 The CSP Cutting-Planes Algorithm : Algorithm A5 .

Now that we have described how we generate the logical cuts and that we have explained the concept of Gomory's cutting-planes we are ready to describe the algorithm based on the two types of cuts and that we will call , for the sake of simplicity , CSP cutting-plane algorithm.

Algorithm A6 : Let us call (P) the CSP formulated as in section 2-2. The algorithm consists of the following steps :

Step 0 : Solve the linear relaxation of (P) . If the solution is not integer GOTO STEP 1 ; Else STOP ,the current solution is optimal .

Step 1 : Using Algorithm A3,generate the logical cuts and add them to problem (P) .The resulting problem is called (P') . GOTO STEP 2 ;

Step 2 : Using Gomory's cutting-planes solve problem (P'). STOP.

4- Computational Results .

To test the algorithm presented so far in this chapter more than 100 CSP's of size ranging from 5 to 30 tasks were randomly generated . The results are summarized in tables 2-2 , 2-3 and 2-4 .

In table 2-2 we present the computational results of the 81 CSP's solved with linear programming. Table 2-3 gives the performance of the CSP algorithm A5 applied to the 20 remaining CSP's. In both tables 2-2 and 2-3 we give the performance of the heuristic algorithm A2 . In table 2-4 we compare the performance of an algorithm based only on Gomory's cuts with that of algorithm A5.Before considering the results of each algorithm , let us describe the data generation process.

Table 2-2 (a) : Computational Results for the 81 CSP's Solved with Linear Programming.

Problem	Number of Tasks	Heuristic Algorithm				LP Results		
		MNC1+	MNC2++	Error	Time*	Optimal Solution Value	Size of LP [rowsxcol]	Time*
1	5	3	3	0	0.06	18	29x 81	0.4
2	5	2	2	0	0.06	27	21x 66	0.2
3	5	2	2	0	0.07	32	21x 54	0.2
4	5	2	2	0	0.06	25	21x 50	0.2
5	6	3	3	0	0.08	27	33x 108	0.6
6	6	3	3	0	0.07	24	33x 93	0.6
7	6	2	2	0	0.06	36	24x 70	0.2
8	6	2	2	0	0.06	45	24x 72	0.2
9	7	3	3	0	0.07	36	37x 108	0.6
10	7	3	3	0	0.08	32	37x 114	0.7
11	7	2	2	0	0.09	40	27x 119	0.4
12	7	3	3	0	0.08	32	37x 141	0.7
13	8	4	4	0	0.09	36	52x 216	1.2
14	8	2	2	0	0.09	54	30x 102	0.5
15	8	4	4	0	0.08	32	52x 184	1.1
16	8	2	2	0	0.09	50	30x 106	0.5
17	9	3	3	0	0.10	48	45x 147	0.9
18	9	2	2	0	0.10	63	33x 102	0.6
19	9	4	4	0	0.10	42	57x 200	1.5
20	9	4	4	0	0.10	40	57x 192	1.4

* Seconds of CYBER 855 (Fortran Compiler)
+ MNC1 = Minimum Number of Crews (exact)
++ MNC2 = Minimum Number of Crews (heuristic).

Table 2-2 (b) : Computational Results for the 81 CSP's Solved with Linear Programming

Problem	Number of Tasks	Heuristic Algorithm MNC1+	MNC2++	Error	Time*	Optimal Solution Value	LP Results Size of LP [rowsxcol]	Time*
21	10	5	5	0	0.11	38	75x 310	2.1
23	10	5	5	0	0.11	60	75x 265	2.1
24	10	4	4	0	0.10	48	62x 216	1.3
25	10	3	3	0	0.12	56	49x 201	1.1
26	11	3	3	0	0.13	72	53x 177	1.3
27	11	5	7	2	0.13	36	109x 476	4.6
28	11	5	5	0	0.13	48	81x 380	2.3
29	11	5	5	0	0.11	54	81x 345	2.3
30	12	3	3	0	0.13	63	57x 246	1.5
31	12	4	4	0	0.13	72	72x 252	1.9
32	12	4	6	2	0.14	54	102x 444	3.8
33	12	3	3	0	0.13	70	57x 234	1.5
34	13	3	4	1	0.13	81	77x 260	2.2
35	13	5	5	0	0.15	68	93x 385	3.4
36	13	4	4	0	0.13	72	77x 304	2.3
37	13	3	3	0	0.13	90	61x 243	1.2
38	14	5	5	0	0.15	89	99x 375	4.0
39	14	6	6	0	0.14	72	116x 450	5.0
40	14	5	5	0	0.15	78	99x 435	4.1
41	14	5	5	0	0.16	66	99x 520	4.1

* Seconds of CYBER 855 (fortran Compiler)
+ MNC1 = Minimum Number of Crews (exact)
++ MNC1 = Minimum Number of Crews (heuristic).

Table 2-2 (c) : Computational Results for the 81 CSP's Solved with Linear Programming.

Problem	Number of Tasks	Heuristic Algorithm MNC1+	MNC2++	Error	Time*	Optimal Solution Value	LP Results Size of LP [rowsxcol]	Time*
44	15	6	6	0	0.17	71	123x 582	5.3
45	15	4	4	0	0.17	88	87x 380	3.0
48	16	6	6	0	0.18	108	130x 654	5.4
52	17	7	8	1	0.18	81	177x 848	8.9
53	17	6	7	1	0.18	90	157x 693	8.4
54	17	7	7	0	0.19	108	157x 770	8.5
55	17	5	5	0	0.19	95	117x 425	5.1
56	18	7	7	0	0.19	99	165x 735	8.7
58	18	5	7	2	0.19	78	165x 686	8.7
59	18	6	6	0	0.18	90	144x 832	7.1
60	19	5	5	0	0.21	112	129x 585	4.8
61	19	6	6	0	0.20	117	151x 792	7.8
62	19	5	5	0	0.20	93	129x 585	4.8
63	19	5	5	0	0.19	106	129x 765	4.9
66	20	7	7	0	0.22	104	181x 749	4.5
68	20	7	7	0	0.21	104	181x 959	9.7
69	21	7	7	0	0.21	102	189x 812	10.7
70	21	6	7	1	0.22	126	189x 749	10.9
71	21	6	6	0	0.21	140	165x 720	8.7
72	21	6	6	0	0.21	129	165x 840	8.8

* Seconds of CYBER 855 (Fortran Compiler)
+ MNC1 = Minimum Number of Crews (exact)
++ MNC1 = Minimum Number of Crews (heuristic).

Table 2-2 (d) : Computational Results for the 81 CSP's Solved with Linear Programming

Problem	Number of Tasks	Heuristic Algorithm MNC1+	MNC2++	Error	Time*	Optimal Solution Value	LP Results Size of LP [rowsxcol]	Time*
73	22	7	7	0	0.22	120	197x1022	12.3
74	22	7	7	0	0.22	135	197x 770	12.2
75	22	7	7	0	0.24	129	197x1015	12.8
76	22	7	7	0	0.24	135	197x 861	12.5
77	23	7	9	2	0.24	126	257x1053	19.0
79	23	8	8	0	0.25	135	231x1272	18.1
80	23	8	8	0	0.24	116	231x 968	18.1
81	24	7	7	0	0.25	136	213x1113	15.9
82	24	9	9	0	0.24	135	267x1305	20.7
83	24	8	9	1	0.25	120	267x1125	20.2
84	24	7	8	1	0.26	144	240x1120	18.6
90	26	8	9	1	0.27	153	287x1494	21.1
91	26	8	8	0	0.28	149	258x1152	19.1
92	26	8	8	0	0.27	181	258x1232	19.6
93	26	8	8	0	0.26	162	258x1296	19.6
94	27	9	9	0	0.29	120	297x1431	20.8
95	27	9	9	0	0.27	128	297x1836	22.0
96	28	8	8	0	0.29	160	276x1232	20.4
97	28	10	12	2	0.29	129	400x1924	57.0
98	29	8	8	0	0.29	168	285x1472	20.6
99	29	10	10	0	0.29	152	349x1390	29.2

* Seconds of CYBER 855 (Fortran Compiler)
+ MNC1 = Minimum Number of Crews (exact)
++ MNC2 = Minimum Number of Crews (heuristic).

Table 2-3 : Computational Results for the 20 CSP's Solved with the CSP Cutting-Plane Algorithm .

Problem	Number of Tasks	Heuristic Algorithm MNC1+	MNC2++	Error	Time*	CSP Cutting-Plane Algorithm Optimal Solution Value	Linear Solution Value	Gap [%]	Size of LP [rowsxcol]	Number of Preassigned Tasks	Number of Gomory Cuts	Lower bound Value	Time*
22	10	4	4	0	0.11	59	48.4	18	62x 224	7	1	59	1.8
42	15	6	8	2	0.16	56	51.0	9	159x 792	10	3	56	9.8
43	15	6	6	0	0.16	81	68.8	15	123x 618	9	8	81	7.9
46	15	4	4	0	0.16	96	90.2	6	87x 619	6	1	96	3.2
47	16	4	4	0	0.17	99	80.2	19	92x 432	10	1	99	4.3
49	16	4	4	0	0.18	90	79.2	12	92x 416	7	6	90	4.8
50	16	5	5	0	0.17	113	105.1	7	111x 510	8	2	113	5.5
51	16	4	5	1	0.18	88	68.6	22	111x 615	8	7	88	8.3
57	18	7	7	0	0.19	88	85.4	3	165x 833	9	1	88	9.1
64	20	6	6	0	0.20	112	112.0	0	158x 672	9	2	112	9.6
65	20	7	8	1	0.22	96	88.3	9	204x1096	10	1	96	14.0
67	20	6	6	0	0.21	127	101.7	21	158x 708	9	15	127	16.3
78	23	8	8	0	0.23	120	114.1	5	231x1016	11	1	120	20.3
85	25	8	10	2	0.25	120	91.3	24	305x1730	13	19	120	36.2
86	25	9	9	0	0.25	128	113.9	11	277x1431	11	5	128	24.3
87	25	7	7	0	0.25	162	150.7	7	221x1099	10	12	162	21.8
88	25	9	11	2	0.26	126	108.4	14	333x1694	14	2	126	32.3
89	25	7	7	0	0.27	151	144.9	4	221x 882	15	2	151	15.2
100	30	9	9	0	0.31	168	166.4	1	327x1674	16	2	168	31.4
101	30	11	11	0	0.31	171	148.2	13	393x1694	17	16	171	56.3

* Seconds of CYBER 855 (Fortran Compiler)
+ MNC1 = Minimum Number of Crews (exact)
++ MNC2 = Minimum Number of Crews (heuristic)

Table 2-4 : Comparison of the Gomory C-P Algorithm and the CSP C-P Algorithm.

Problem	Number of Tasks	Optimal Solution Value	Linear Solution Value	Size of LP	Gomory C-P Algorithm Number of Cuts	Gomory C-P Algorithm Value of Lower Bound	Gomory C-P Algorithm Gap [%]	CSP C-P Algorithm Number of Cuts	CSP C-P Algorithm Value of Lower Bound
1	10	59	48.4	62x 224	20	56.0	5.1	1	OPT
2	15	56	51.0	159x 792	15	OPT	OPT	3	OPT
3	15	81	68.8	123x 618	20	79.2	2.2	8	OPT
4	15	96	90.2	87x 619	10	OPT	OPT	1	OPT
5	16	99	80.2	92x 432	16	OPT	OPT	1	OPT
6	16	90	79.2	92x 416	20	90.0	0.	6	OPT
7	16	113	105.1	111x 510	20	109.4	3.2	2	OPT
8	16	88	68.6	111x 615	20	84.0	4.5	7	OPT
9	18	88	85.4	165x 833	4	OPT	OPT	1	OPT
10	20	112	112.0	158x 672	19	OPT	OPT	2	OPT
11	20	96	88.3	204x1096	20	93.3	2.8	1	OPT
12	20	127	101.7	158x 708	20	123.2	3.0	15	OPT
13	23	120	114.1	231x1016	20	120.0	0.	1	OPT
14	25	120	91.3	305x1730	20	114.2	4.8	19	OPT
15	25	128	113.9	277x1431	20	128.0	0.	5	OPT
16	25	162	150.7	221x1099	7	OPT	OPT	12	OPT
17	25	126	108.4	333x1693	15	OPT	OPT	2	OPT
18	25	151	144.9	221x 882	18	OPT	OPT	2	OPT
19	30	168	166.4	327x1674	17	168.0	0.	2	OPT
20	30	171	148.2	393x1694	20	169.1	1.1	16	OPT

* "OPT" means that the lower bound is optimal.

4-1 The Data Generation Process .

For each one of the 101 problems , we needed to generate the cost coefficients of the linking-arcs , the duration times of the tasks and their starting times . The working duty period was assumed in all cases to be equal to 6 hours . The number of crews was provided by the heuristic algorithm A2 .

All the data were randomly generated according to a uniform distribution . The durations (resp. starting times) were assumed to vary within the range [45 minutes to 2 hours 30 minutes] (resp. [00.00 hours to 24.00 hours]). Each cost coefficient c_{ij} associated with linking-arc (β_i,α_j) has been generated according to the following formula :

$$c_{ij} = (1+ \alpha)\, d_{ij}$$

where d_{ij} = duration of linking-arc (β_i,α_j)

= starting time of task j - finishing time of task i

and

α = random number generated by the uniform ditribution U(0,1)

4-2 Computational Results for Algorithm A2.

The heuristic algorithm A2 , for finding the minimum number of crews required to process all tasks , has proved to be very effective . Thus out of the 101 randomly generated CSP's ,it has produced the exact minimum number of crews (MNC) in 84 cases . For the remaining problems , it has given the MNC with an error of 1 in 10 problems and an error of 2 for 7 others .Column 3 ,of tables 2.2 and 2.3 ,which gives the exact value of MNC was obtained using the algorithm described in the next chapter .

4-3 Computational Results for Algorithm A5 .

The first obvious remark that one can make by just looking at table 2-2 is that most of the CSP's considered (80%) were solved at step 0 of the algorithm ie by just using an LP package . This shows the tightness of the formulation of the problem (section 2-2) .The LP package used in our case is the XMP package of Marsten [112].

The remaining 20 problems which could not be solved by the LP package were tackled by using algorithm A5. The results are shown in table 2-3 .

Comparing the results produced by the Gomory's fractional C-P algorithm with those obtained with algorithm A5 we can clearly see from table 2.4 that without the addition of the logical cuts to the CSP we would have been forced to embed the algorithm into a tree search.In fact , by adding the logical cuts we have strengthened the CSP's formulation and subsequently only few Gomory's cuts were required to reach the optimum while for the same set of problems when we have removed the logical cuts many more Gomory's cuts were needed and in half of the cases the algorithm was not converging at all .

As far as the choice of the source row (in Gomory's algorithm) is concerned we have noticed , after trying several possibilities , that the best strategy consists of cutting first on the variables (with the highest fractional part in the optimal tableau) which correspond to the linking-arcs ; if all such variables are integer then choose the ones corresponding to the task-arcs ; in the last resort , cut on the source and sink - arcs' variables . Also because of the limitations in computer storage we have decided to set the maximum number of cuts allowed to 20 .

4-4 Drawback of the CSP Cutting-Plane Algorithm .

The cutting-plane computing program used to solve the CSP is based on the linear programming package of Marsten . It is reported in [112] that the largest linear problems that have been solved with this package had 1500 constraints and 7000 variables . In our case , a problem of n tasks and K crews will have more than nK constraints.If we assume that in graph G , representative of CSP , each task-arc is linked on average with 7 other task-arcs (which is quite a reasonable assumption) then the problem will have more than 7nK variables . Thus , for example , a 50 task-20 crew CSP will have more than 1000 constraints and 7000 variables .

Consequently , because of limitations on the size of the problem imposed by LP packages , this formulation is very restricted . Only small to medium size CSP's of up to at most 50 tasks can be solved .

5- The Lagrangian Relaxation Method .

The branch and bound (B&B) procedure is the most successful and most widely used technique in integer programming (IP) . Thus , all the commercial general IP packages are based on this technique [27,64,102,130] and for most of the packages that solve particular IP problems it is still the B&B procedure which is used .

The most important step in the B&B method is the determination of the lower bound of the problem (see [67] for a detailed description of the B&B method) . To obtain an upper bound to the IP problem (assume it is a minimization problem) heuristic algorithms are usually used . However , for the lower bound several techniques [26,37,43,68,70] have been proposed to obtain good lower bounds to the problem . Among these techniques one of the most efficient and most successful is the Lagrangian relaxation .

5-1 A CSP Formulation Suitable for the Lagrangian Relaxation .

The CSP is formulated as an unconstrained CSP plus additional constraints which express the fact that all optimal paths of network G (ie the network representative of the CSP) must be of length T or smaller , where T is the work duty period .

The formulation is as follows :

If in network G we let

$x_{ij} = 1$ if arc (β_i,α_j) is in the optimal solution ;

$= 0$ otherwise ;

then , the CSP becomes :

$$\min \sum_{(\beta_i,\alpha_j) \in N} c_{ij} x_{ij} \qquad (2\text{-}18)$$

subject to :

$$\sum_{j \in V_i^+} x_{ij} - \sum_{p \in V_i^-} x_{pi} = \begin{cases} 0 & \text{for } i=1,\dots,n \\ K & \text{for } i=S \\ -K & \text{for } i=R \end{cases} \qquad (2\text{-}19)$$

x_{ij} must form K paths of length less than or equal to T . (2-20)

$$x_{ij} \in \{ 0,1 \} \qquad (2\text{-}21)$$

where c_{ij},N,V_i^+,V_i^-,K and n are as defined in section 2.3 (chapter 1).

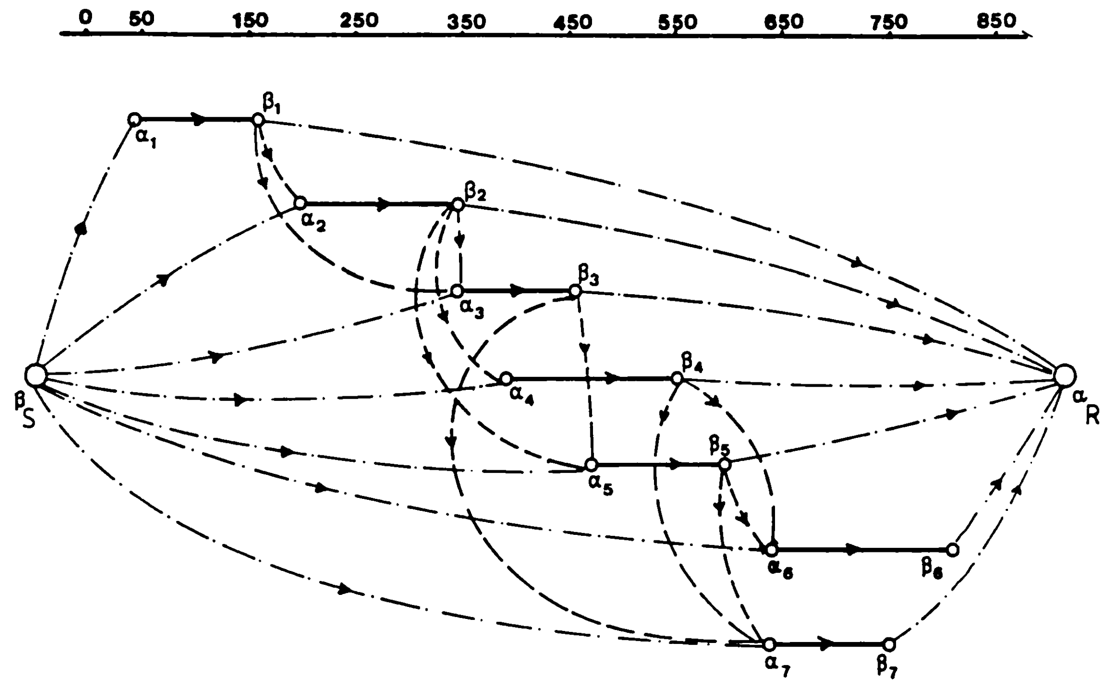
5-2 Expressing the Time Constraints as Linear Constraints .Algorithm A6.

Clearly , the problem which consists of (2-18),(2-19) and (2-21) is a minimum cost network flow problem .In this section we will consider the best way of expressing constraints (2-20) that we will call "time-constraints'.

5-2.1 Example .

Consider the 7 task-3 crews CSP represented by network G of figure 2.7. We assume the work duty period to be 400 minutes .

Figure 2-7 : Network G



Dropping the time constraints from the CSP and solving the remaining problem as a minimum cost network flow problem we obtain the optimal solution x_0 which consists of the following paths :

$P_1= (\beta_s,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_2)(\alpha_2,\beta_2)(\beta_2,\alpha_3)(\alpha_3,\beta_3)(\beta_3,\alpha_5)\ (\alpha_5,\beta_5)\ (\beta_5,\alpha_6)(\alpha_6,\beta_6)(\beta_6,\alpha_R)$

$P_2= (\beta_s,\alpha_4)(\alpha_4,\beta_4)(\beta_4,\alpha_R)$

$P_3= (\beta_s,\alpha_7)(\alpha_7,\beta_7)(\beta_7,\alpha_R)$

The cost of x_0 is 340 .

Clearly this solution is not feasible for the CSP since the length of path P_1 (ie 750minutes) exceeds the work duty period (ie 400 minutes).Consequently we need to add time constraints to the relaxed problem so as to eliminate x_0 . In fact , all we need to add is just one constraint that will prevent the formation of path P_1 in the optimal solution . Paths P_2 and P_3 do not violate the time restriction .

At first we wanted to express the time constraint as follows :

$$100x_{12}+50x_{12}+140x_{23}+10x_{23}+100x_{35}+30x_{35}+120x_{56}+50x_{56}+150x_{6R} \leq 400$$

ie $$150x_{12}+150x_{23}+130x_{35}+170x_{56}+150x_{6R} \leq 400 \qquad (2\text{-}22)$$

This is the way we expressed the time constraints in section 2.5.Unfortunately it is not correct to express the time constraints in this way because by doing so we eliminate the following feasible solution x_1 of the CSP :

x_1 consists of the following paths :

$$P_1' = (\beta_s,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_2)(\alpha_2,\beta_2)(\beta_2,\alpha_3)(\alpha_3,\beta_3)(\beta_3,\alpha_R)$$

$$P_2' = (\beta_s,\alpha_5)\ (\alpha_5,\beta_5)\ (\beta_5,\alpha_6)(\alpha_6,\beta_6)(\beta_6,\alpha_R)$$

$$P_3' = (\beta_s,\alpha_4)(\alpha_4,\beta_4)(\beta_s,\alpha_7)(\alpha_7,\beta_7)(\beta_7,\alpha_R)$$

Clearly , x_1 is a feasible solution for the CSP of figure 2.7 because each one of P_1',P_2' and P_3' has length less than or equal to 400 minutes and they cover all task-arcs of G. Now to see that x_1 is cut off by constraint (2-22) we just need to check that (2-22) is violated for x_1 .

Indeed in x_1 we have :

$$x_{12}=1\ ;\ x_{23}=1\ ;\ x_{35}=0\ ;\ x_{56}=1\ ;\ x_{6R}=1$$

Hence the left-hand side of (2-22) becomes

$$150 \times 1\ +\ 150 \times 1\ +\ 130 \times 0\ +\ 170 \times 1\ +\ 150 \times 1\ = 620 > 400$$

Therefore, we cannot express the time constraints as in constraint (2-**22**) .

Now, let us consider path P_1 . Knowing that each $x_{ij} = 0$ or 1 , P_1 is well defined by the following constraint :

$$x_{12}+x_{23}+x_{35}+x_{56} = 4 \qquad (2\text{-}23)$$

If we want to eliminate the formation of P_1 in the optimal solution we need to add the following constraint to the relaxed problem of the CSP :

$$x_{12}+x_{23}+x_{35}+x_{56} \leq h \qquad (2\text{-}24)$$

where h is a constant to be determined . If we set h=1 we will eliminate a feasible solution that has task-arcs (α_1,β_1) and (α_2,β_2) in one path and task-arcs (α_5,β_5) and (α_6,β_6) in another path .Hence h=1 is not possible .Similarly if we set h=2 we will eliminate solution x_1 defined above .Indeed , in x_1 we have the left-handside of (2-24) equal to $x_{12}+x_{23}+x_{35}+x_{56} = 1+1+0+1 = 3$

Finally , the time constraint that prevents the formation of P_1 in the optimal solution is

$$x_{12}+x_{23}+x_{35}+x_{56} \leq 3 \qquad (2\text{-}25)$$

In fact one can do better . By remarking that task-arcs (α_1,β_1), (α_2,β_2) and (α_3,β_3) form a maximal path P_4, in that any task-arc added to P_4 will violate the time restriction , we can write the time constraint as

$$x_{12}+x_{23}+x_{35} \leq 2 \qquad (2\text{-}26)$$

and by remarking that task-arcs (α_5,β_5) and (α_6,β_6) form a maximal path we can write the time constraint :

$$x_{35}+x_{56} \leq 1 \qquad (2\text{-}27)$$

Hence constraints (2-26) and (2-27) express the time constraints that prevent the formation of path P_1 in the optimal solution of the CSP of figure 2.7 .Constraints (2-26) and (2-27) are together stronger than constraint (2-25). Thus path P_5 is eliminated by constraint (2-26) but is not eliminated by constraint (2-25), where P_5 is

$P_5 = (\beta_S,\alpha_1)(\alpha_1,\beta_1)(\beta_1,\alpha_2)(\alpha_2,\beta_2)(\beta_2,\alpha_3)(\alpha_3,\beta_3)(\beta_3,\alpha_5)(\alpha_5,\beta_5)(\beta_5,\alpha_R)$

5-2.2 Algorithm A6 .

Given a path P of length greater than the work duty period , the following algorithm expresses the time constraints relatively to P.

We said previously that any path P= $(\beta_S,\alpha_{h_1})(\alpha_{h_1},\beta_{h_1}),\ldots,(\alpha_{h_k},\beta_{h_k})(\beta_{h_k},\alpha_R)$ of G is well defined by its set of linking-arcs A= { $(\beta_{h_1},\alpha_{h_2}),\ldots,(\beta_{h_{k-1}},\alpha_{h_k})$ }
Hence if we are given A we can easily form P . In the algorithm when we say " a path formed by the arcs of A " we mean the path whose linking-arcs are the arcs of A .

Step 0 : Let $h_1,h_2,\ldots,h_k$ be the linking-arcs of P ; set i=1 and A ={ h_1 }
GOTO STEP 1 ;

Step 1 : i=i+1 ; A= A $\cup$ {h_i} ;If the path formed by the arcs of A has length greater than T, GOTO STEP 2 ; Else GOTO STEP 3

Step 2 : Add the following time constraint to the CSP

$$\sum_{(\beta_i,\alpha_j)\in A} x_{ij} \le |A|-1 \qquad (|A| = \text{cardinality of } A) \qquad (2\text{-}28)$$

Set A = { h_i } ; GOTO STEP 3

Step 3 : If i = k STOP ; Else GOTO STEP 1 .

Algorithm A6 runs in time O(n) where n is the number of linking-arcs of path P.

5-3 A Brief Description of Lagrangian Relaxation .

We saw in the previous section that given a path P whose length exceeds the work duty period the corresponding time constraint is of the form :

$$\sum_{(\beta_i,\alpha_j)\in A(p)} x_{ij} \le h$$

where h is a suitable constant and A(P) is the set of linking-arcs of P.

The extra constraints which express the restriction on the maximum length of optimal paths (of G) cannot all be added to the CSP at the same time because their number increases exponentially with the size of the CSP (ie number of tasks) . Consequently , they are added when necessary and relaxed in a Lagrangian way .

At this point it is worthwhile explaining the main idea of the Lagrangian relaxation technique . For more details the interested reader is refered to [61,71,74,135].

Let us assume that a set of r time constraints have been determined . Instead of solving the CSP directly the Lagrangian relaxation consists of solving the following easier problem ($\mathcal{P}_u$):

$$\min \sum_{(\beta_i,\alpha_j)\in N} c_{ij}\, x_{ij} + \sum_{k=1}^{r} u_k\Big(\sum_{(\beta_i,\alpha_j)\in A(p_k)} x_{ij} - h\Big) \qquad (2\text{-}29)$$

subject to : (2-19) and (2-21)

where P_k is the path corresponding to the kth ($k=1,\ldots,r$) time constraint and $A(P_k)$ is its set of linking-arcs .

Clearly , given $u_1,u_2,\ldots,u_k \geq 0$ this problem is a minimum cost network flow problem which can easily be solved . Also one can easily prove that the optimum of ($\mathcal{P}_u$) (say Z_u) is a lower bound to the optimum of the CSP (say Z) ie $Z_u \leq Z$. Depending on the values of the Lagrange multipliers $u_1,u_2,\ldots,u_k$ the value of this lower bound Z_u is different . Since the main aim of the Lagrangian relaxation is to determine a good lower bound to the CSP to be embedded in a branch and bound scheme the problem becomes one of solving :

$$\max_u z_u = \max_u \min_x \left[\sum_{(\beta_i,\alpha_j)\in N} c_{ij}\, x_{ij} + \sum_{k=1}^{r} u_k\Big(\sum_{(\beta_i,\alpha_j)\in A(p_k)} x_{ij} - h\Big) \right] \qquad (2\text{-}30)$$

subject to (2-19) and (2-21)

There are several methods for solving this problem [1,18,19,69,121].However, the most widely used is subgradient optimization [48,86]. This is due to the fact that it is very simple to program and that it has performed well for most IP problems solved by Lagrangian relaxation [11,73,136].

Starting with an initial value $U^0 = (u_1^0,\dots,u_r^0)$,the subgradient optimization method consists of generating a series of Lagrange multiplier vectors $\{U^{k+1}\}$ defined by

$$u^{k+1} = u^k + t_k\Big(\sum_{(\beta_i,\alpha_j)\in A(p_m)} x_{ij}^k - h\Big) \qquad m=1,\dots,r \tag{2-59}$$

where (x_{ij}^k) is the optimal solution of problem (P_{U^k}) and t_k is given by :

$$t_k = \frac{\lambda(Z^* - Z_{U^k})}{\sum_{m=1}^{r} \|\sum x_{ij} - h\|^2} \tag{2-60}$$

in which Z^* is a feasible solution to CSP.It can be found by a heuristic ;

Z_{U^k} is the optimum of (P_{U^k})

$\|A\|^2$ is the Euclidean norm of A ;

and λ is a parameter which is initially set to 2 and is halved if after a given number of iterations there is no increase in the optimal value of (P_{U^k}).

5-4 Generating the Time Constraints .Algorithm A7.

In the previous section we have made the assumption that a set of r time constraints (which is a small subset of the large set of all time constraints) has been determined . The following is an algorithm which describes the dynamic way in which the time constraints are generated .

Algorithm A7.

Step 0 : Set $C = \emptyset$ and $i = 0$. GOTO STEP 1 .

Step 1 : Solve the CSP without considering the time constraints (this is a minimum cost network flow problem) . If all paths of the solution have length T (ie the work duty period) or less STOP : the solution is optimal for the CSP ; Else add all the paths with length greater than T to C . GOTO STEP 2.

Step 2 : i =i +1 ; If i > 5 STOP ; Else express all the paths in C as time constraints (using algorithm A6) , add them to the current problem and relax them in a Lagrangian way .GOTO STEP 3 .

Step 3 : Perform 20 subgradient iterations on the relaxed problem . At each iteration add all the paths ,that violate the work duty period, to C . GOTO STEP 2.

6- Computational Results for the Lagrangian Relaxation .

A total of 60 CSP's (see table 2.5) were randomly generated according to the data generation process described in section 4 .

As far as the parameters of the subgradient optimization are concerned the initial values of the Lagrange multipliers were set to zero ; after few tests on some problems we have noticed that the most suitable value for the step size was 4 . Also , the value of the upper bound was set initially to infinity and whenever a feasible solution was found in the course of algorithm A7 the upper bound was updated .

All problems were considered in 3 groups as follows :

Group1 . Twenty CSP's were randomly chosen from the 81 problems generated in section 4 for which the optimal solution was found after using just an LP package . As it can be seen from table 2-5 , except for 5 problems , the quality of the lower bound is quite good . Furthermore , when compared with the results of table 2-2 we can see that less computing time than for the linear relaxation was necessary to obtain these bounds. One should not forget that none of the problems considered were solved to optimality by the Lagrangian relaxation and to achieve this we need to embed the technique in a tree search which means that more computing time will be required .

Group 2 . The 20 CSP's of section 4 for which the addition of the cutting-planes was necessary to obtain the optimum were then tested . Clearly , the quality of the lower bound obtained with the Lagrangian relaxation (table2-5) for this set of problems was in general poor.Thus for some problems the bound was more than 20% away from the optimum. Hence,if we wanted for this specific problem to embed this bound in a tree search procedure we would have to explore thousands of nodes before we could find the optimal solution .

Group 3 : In an attempt to solve larger CSP's , twenty problems of size varying between 35 and 55 tasks were then randomly generated . In 70% of the cases the Lagrangian relaxation could produce lower bounds in the range of 96%-100% which represent good quality bounds to be embedded into a tree search procedure . However, for the 6 other problems the quality of the bound was not that good . Thus for the 35 tasks problem it dropped to 78% which is basically a very poor bound. For this group of problems the optimal solution was found by the algorithm described in chapter 3 .

Table 2-5 : Computational Results for Lagrangian Relaxation

Group 1

Problem	Number of Tasks	Value of Optimal Solution	Value of Lower Bound	Gap [%]	Time*
1	5	18	17.15	4.7	0.193
2	11	72	68.98	4.2	0.329
3	12	70	67.97	2.9	0.386
4	13	68	66.71	1.9	0.391
5	14	89	87.13	2.1	0.460
6	14	72	71.21	1.1	0.458
7	14	78	78 **	0.	0.378
8	15	71	68.09	4.1	0.501
9	16	108	105.95	1.9	0.526
10	17	95	92.63	2.5	0.614
11	18	90	87.84	2.4	0.628
12	19	117	115.25	1.5	0.767
13	21	126	124.99	0.8	0.823
14	21	140	140 **	0.	0.985
15	22	135	128.93	4.5	0.718
16	23	126	126 **	0.	1.053
17	24	136	129.20	5.0	0.940
18	26	153	148.41	3.0	1.299
19	28	160	159.84	0.1	1.736
20	29	168	163.63	2.6	1.975

* Seconds of CYBER 855 (Fortran Compiler)
** Non-Optimal Solution

Table 2-5 : Computational Results for Lagrangian Relaxation

Group 2

Problem	Number of Tasks	Value of Optimal Solution	Value of Lower Bound	Gap [%]	Time*
1	10	59	48.31	18.1	0.350
2	15	56	50.37	10.1	0.481
3	15	81	67.95	16.1	0.507
4	15	96	89.43	6.8	0.434
5	16	99	79.38	19.8	0.629
6	16	90	77.20	14.2	0.626
7	16	113	103.62	8.3	0.559
8	16	88	67.68	23.0	0.538
9	18	88	85.10	3.3	0.644
10	20	112	108.53	3.1	0.769
11	20	96	86.20	10.2	0.817
12	20	127	98.10	22.8	0.985
13	23	120	108.93	9.2	1.264
14	25	120	91.09	24.1	1.255
15	25	128	112.21	12.3	1.308
16	25	162	144.33	10.9	1.370
17	25	126	107.71	14.5	0.939
18	25	151	138.79	8.1	1.408
19	30	168	158.57	5.6	1.942
20	30	171	145.39	14.9	1.952

* Seconds of CYBER 855 (Fortran Compiler)

Table 2-5 : Computational Results for Lagrangian Relaxation

Group 3

Problem	Number of Tasks	Value of Optimal Solution	Value of Lower Bound	Gap [%]	Time*
1	35	207	201.20	2.8	2.262
2	35	200	196.20	1.9	2.783
3	35	216	214.71	0.6	2.646
4	35	225	175.50	22.0	2.097
5	40	252	232.59	7.7	3.645
6	40	208	195.32	6.1	3.673
7	40	234	227.45	2.8	3.885
8	40	243	240.08	1.2	3.179
9	45	279	263.37	5.6	4.611
10	45	248	240.81	2.9	3.560
11	45	256	252.16	1.5	4.518
12	45	279	272.30	2.4	4.274
13	50	297	294.32	0.9	5.032
14	50	315	309.65	1.7	4.748
15	50	264	264 **	0.	5.230
16	50	301	255.35	15.2	5.901
17	55	304	304 **	0.	5.975
18	55	351	348.89	0.6	6.270
19	55	296	271.43	8.3	6.730
20	55	322	314.92	2.2	6.127

* Seconds of CYBER 855 (Fortran Compiler)

** Non-Integer Solution

7- Conclusion .

The formulation of the CSP presented in section 2-2 has proved to be so tight that 80% of the 101 CSP's randomly generated were exactly solved by the use of just an LP package . Also the CSP cutting-plane algorithm which is based on logical and Gomory's cutting-planes has proved to be quite useful in solving the remaining problems .

However , because of the very large size of the formulation and because of the limitations imposed by LP packages (on which the cutting-plane program is based) only CSP's of up to 30 tasks have been solved . In order to solve larger problems Lagrangian relaxation was investigated on another formulation . In its original form this formulation consists only of a BCSP .The time constraints are dynamically generated in the course of the Lagrangian relaxation algorithm. As they are generated , they are added to the problem and subsequently relaxed in a Lagrangian fashion .

In addition to being able to tackle CSP's of size up to 55 tasks , the Lagrangian relaxation technique has proved in some cases to be an efficient tool for obtaining good lower bounds . Thus we can say that in 60% of the cases tried the quality of the bound was quite satisfactory . However for some cases it can be as bad as 20% away from the optimum . Thus in 40% of the CSP's tested we found the Lagrangian relaxation to be poor .

In summary , the CSP cutting-planes algorithm based on an efficient formulation of the CSP has proved to be much better than the Lagrangian relaxation . It would have been worth further investigation had it not been for the fact that the procedure of the next chapter proved to be a vastly superior method for all CSP test problems .

CHAPTER 3

A FORMULATION OF THE CSP BASED ON GRAPH THEORETICAL CONCEPTS

1- INTRODUCTION .

We saw in the previous chapter why the direct formulation of the CSP was inadequate for large problems (more than 50 tasks) . In an attempt to overcome this drawback a new formulation of the CSP will be given in this chapter . Like the previous one it is a network flow based formulation . The network $\bar{G}$ on which this formulation is based is an expansion of the network G of a CSP (see section 3).
In $\bar{G}$, all the paths have length equal to the working duty period or less . Consequently , the time constraints will not be considered in the formulation which consists of flow constraints and additional partitioning type constraints .

In section 2 , an example will be given which explains the main idea of converting network G representative of a CSP into network $\tilde{G}$ and the possible reduction of $\tilde{G}$ into a smaller network $\bar{G}$. The general algorithm of expanding any network G , representing a CSP , into a network $\tilde{G}$ and the reduction of $\tilde{G}$ into $\bar{G}$ will be described in detail in section 3 . Section 4 deals with the formulation of the problem. The problem is solved by using just a linear programming package . In section 5 we study the worst case analysis of the size of network $\tilde{G}$ on which the formulation is based . Very good results for 270 randomly generated CSP's are presented in section 6 . Two straightforward extensions of the problem are considered in section 7 . The good results obtained for the 270 CSP's led us to ask ourselves about the possible integrality of the extreme points of the convex polytope of the CSP .This is discussed in section 8 .

2- Example Illustrating the Main Idea of the Formulation .

Consider network G , of figure 3-1 ,which represents a 4 task-CSP . For each task i (i=1,...,4) of the CSP , let :

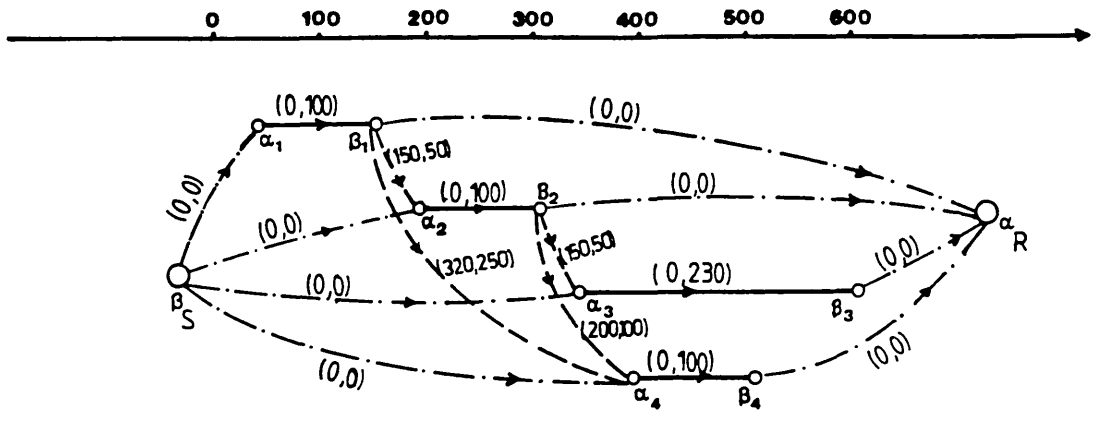
P(i) = set of all paths of G that have arc (α_i,β_i) as the first task-arc .

For example , corresponding to the second task of the CSP represented by G we have:

$$P(2) = \{(\beta_S, \alpha_2)(\alpha_2,\beta_2)(\beta_2, \alpha_3)(\alpha_3,\beta_3)(\beta_3, \alpha_R)\} \cup \{(\beta_S, \alpha_2)(\alpha_2,\beta_2)(\beta_2, \alpha_R)\}$$
$$\cup \{ (\beta_S, \alpha_2)(\alpha_2,\beta_2)(\beta_2, \alpha_4)(\alpha_4,\beta_4)(\beta_4, \alpha_R) \}$$

Now , if we consider task-arc (α_2,β_2), it is clear that it can belong either to a path of P(1) or to a path of P(2) . Hence we can represent task 2 by 2 directed task-arcs (α_2^1,β_2^1) and (α_2^2,β_2^2) having the same starting and finishing times as

Figure 3-1 : Network G Representing a 4 Task-CSP.



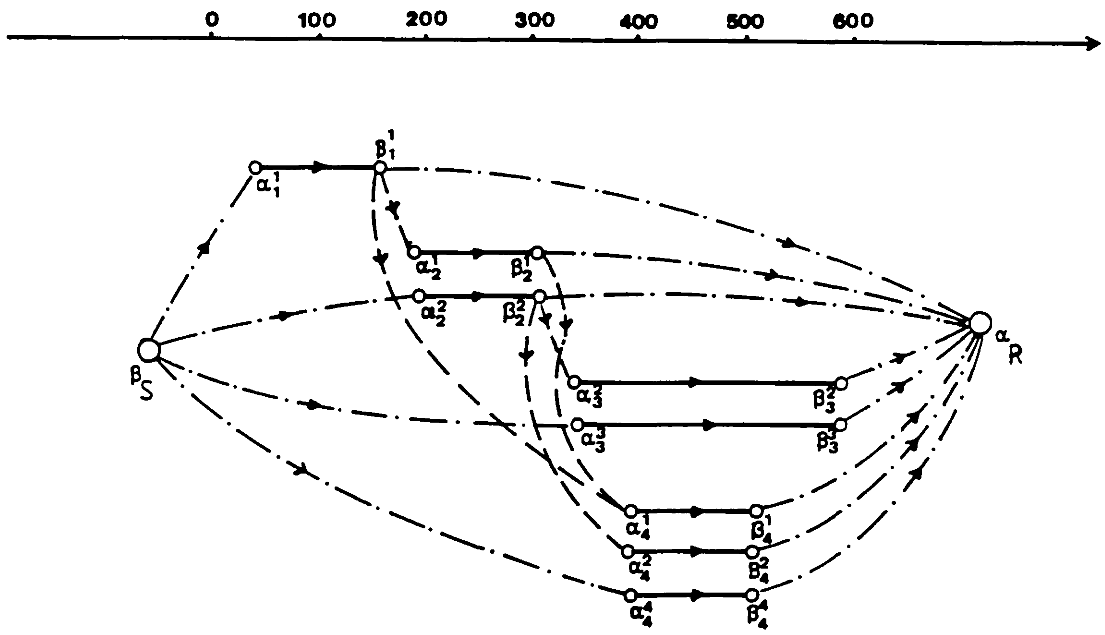
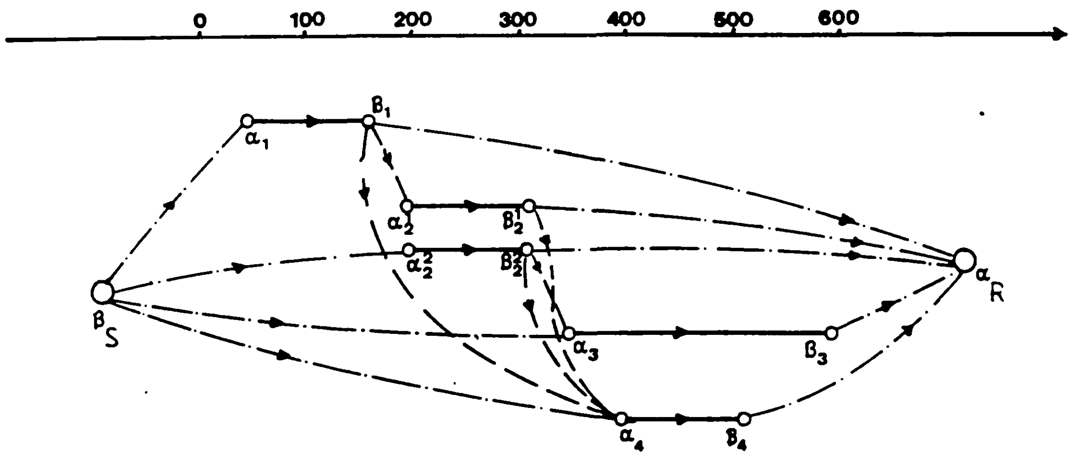


Figure 3-2 : Network $\widetilde{G}$

(α_2,β_2). If , in the optimal solution arc (α_2^1,β_2^1) is chosen this will mean that task 2 will be covered by the same crew that starts from task 1 . Similarly , if arc (α_2^2,β_2^2) is picked up then this means that one of the crews will start its route by covering task 2 . In the rest of this section , we will express this fact by saying that arc (α_2^1,β_2^1) (resp. (α_2^2,β_2^2)) represents task 2 in P(1) (resp. P(2)) . Repeating the same procedure of representing by task-arcs each task in the possible P(i) ' s , and joining all the task-arcs corresponding to the same P(i) ,(i=1,...,4) , we get network $\tilde{G}$ of figure 3-2. Furthermore this new network $\tilde{G}$ can be reduced in size . Thus , it can easily be seen that task-arcs (α_4^1,β_4^1), (α_4^2,β_4^2) and (α_4^4,β_4^4) can be coalesced . This is because they all have the same end , ie they all have no succeeding task-arc . For the same reason , task-arcs (α_3^2,β_3^2) and (α_3^3,β_3^3) can also be coalesced . But (α_2^1,β_2^1) and (α_2^2,β_2^2) cannot be coalesced because they do not have the same end : arc (α_2^2,β_2^2) is linked with (α_4^2,β_4^2) and (α_3^2,β_3^2) , whereas (α_2^1,β_2^1) has no link at all with the arcs that represent task 3 (ie (α_3^2,β_3^2) and (α_3^3,β_3^3)).

The network $\bar{G}$ obtained by coalescing (α_4^1,β_4^1), (α_4^2,β_4^2) and (α_4^4,β_4^4) together and by coalescing (α_3^2,β_3^2) and (α_3^3,β_3^3) together is given in figure 3-3.

Figure 3-3 : Network $\bar{G}$.



Finally , before going on to the next section , it is worthwhile noting that :

(i) Networks G,$\tilde{G}$ and $\bar{G}$ are all alike in that they have the same structure , ie they all have task-arcs, linking-arcs and source (sink)-arcs ;

(ii) To each task i of the CSP , there corresponds a unique task-arc (α_i,β_i) in G and a set of task-arcs (α_i^1,β_i^1), (α_i^2,β_i^2),..., (α_i^p,β_i^p) in $\tilde{G}$ (or $\bar{G}$) which represents task i in the possible P(k)'s ,k=1,...,p ;

(iii) For each task i of the CSP , the source β_S of network $\tilde{G}$ (or $\bar{G}$) is linked with the task-arc (β_S, α_i^i) that represents i in P(i). This is because all the other task-arcs that represent i in the other P(j)'s have to belong to paths starting with previous task arcs . Consequently , there is no need to consider the source arcs that link them with the source . However , the sink α_R is linked with all the task-arcs of $\tilde{G}$(and $\bar{G}$) ;

(iv) It is clear that in each set of task-arcs of $\tilde{G}$ (or $\bar{G}$) one and only one arc must be covered ;

(v) The length of any path of $\bar{G}$ is less than or equal to the work duty period since only the task-arcs corresponding to the same P(i) have been linked together.

3- The Graph Expansion Algorithm A8.

Let us assume that the tasks are ordered and renumbered in ascending order of their starting time τ_i^s.

Let P (k) be the family of all paths of network G from the source β_S to the sink α_R whose first task-arc corresponds to task k . Consider a task-arc (α_i,β_i) in G . Clearly , arc (α_i,β_i) must belong to one of the paths P_k ,k=1,...,i so that

$$FT_i - ST_k \leq T \qquad \text{and} \qquad FT_k + \Delta(FL_k, SL_i) \leq ST_i \tag{3-1}$$

Let us denote the set of those k which satisfy the above condition for a given i by Q_i .

Step 1 : Task-arcs

Each task i corresponds to a set of $|Q_i|$ arcs (α_i^p,β_i^p) , $p \in \tilde{Q}_i$. The pth copy of this arc,ie (α_i^p,β_i^p) , is a representation of (α_i,β_i) in the family P(p) . The cost and time duration of arcs (α_i^p,β_i^p) , $p \in \tilde{Q}_i$ are the same as those for arc (α_i,β_i) in G , and their totality (for i=1,...,m) is referred to as the task-set of $\tilde{G}$.

Step 2 : Linking,source and sink-arcs .

(i) From a 'super-source' vertex β_S add the source-arcs (β_S,α_i^1) for i=1,...,m (corresponding to arcs (β_S,α_i) in G) ;

(ii) To a 'super-sink' vertex α_R add the sink-arcs (β_i^p,α_R) for i=1,...,m, and $p \in \tilde{Q}_i$, (corresponding to arcs (β_i,α_R) in G) ;

(iii) Add linking-arcs (β_k^p,α_i^R) $k \varepsilon \tilde{Q}_i$; for i=1,...,m, $p \varepsilon \tilde{Q}_i$;(corresponding to the linking-arcs (β_k,α_i) of G).

The cost and time duration of all these arcs are the same as for their corresponding arcs in G .

Step 3 : Reduction Process .

Graph $\tilde{G}$ above can be reduced in size somewhat by some very simple equivalence conditions , to produce a new graph that we will call $\bar{G}$. In particular , consider a task i and all the arcs (α_i^p,β_i^p) ,$p \in \tilde{Q}_i$ which represent i in $\tilde{G}$. Let $\tilde{V}_i^+$ be the set of terminal vertices of arcs which emanate from vertex $\beta_i^p$of graph $\tilde{G}$. Let $\tilde{W}_i^+$be the set of task-arcs corresponding to the linking-arcs of G whose initial vertices are in $\tilde{V}_i^+$. We assume a dummy task-arc (say (n+1)) corresponding to the case where α_R (ie the sink of $\tilde{G}$) belongs to $\tilde{V}_{i_p}^+$.If $\tilde{W}_{i_p}^+$is the same set for all $p \in S \subset Q_i$ then any path $p \in$ S which "covers" task i will have the same 'future' (ie will follow the same subpath to α_R) regardless of its 'past' (ie the subpath followed to reached task i). Thus , the arcs (α_i^p,β_i^p) for $p \in$ S can be coalesced into a single arc (α_i^s,β_i^s) so that every arc emanating from β_i^p in $\tilde{G}$ now emanates from β_i^s and every arc terminating at α_i^p in $\tilde{G}$ now terminates at α_i^s. G is the graph with all such arcs coalesced and $\hat{\tilde{Q}}_i$,$\bar{V}_{i_p}^+$the equivalents of $\hat{\tilde{Q}}_i$,$\tilde{V}_{i_p}^+$.

The above algorithm which transforms G into $\bar{G}$ clearly produces a graph $\bar{G}$ in which all paths are time feasible . Indeed, the penalty paid for ensuring this path feasibility is firstly that the graph is enlarged , and secondly that we must now add "set-partitioning" type constraints to express the fact that for a given task i , only one of the arcs in the set represented by $\bar{Q}_i$ needs to be "covered" by the β_S to α_R paths in $\bar{G}$.

4- The Problem Formulation .

Let $y_{i_p j_q} = \begin{cases} 1 & \text{if non-required arc } (\beta_i^p, \alpha_j^q) \in \bar{N} \text{ of } \bar{G} \\ & \text{is in the solution} \\ 0 & \text{otherwise} \end{cases}$

The problem now becomes :

$$\text{Minimize} \sum_{(\beta_i^p, \alpha_j^q) \in \bar{N}} c_{ij}\, y_{i_p j_q} \tag{3-2}$$

subject to :

$$\sum_{j_q \in \bar{V}_{i_p}^+} y_{i_p j_q} - \sum_{j_q \in \bar{V}_{i_q}^-} y_{j_q i_p} = \begin{cases} 0 & , \ i=1,\ldots,m \ ; \ p \in \bar{Q}_i \\ K & , \ i=S \ , \ p=1 \\ -K & , \ i=R \ , \ p=1 \end{cases} \tag{3-3}$$

$$\sum_{p \in \bar{Q}_i} \sum_{j \in \bar{V}_{i_p}^-} y_{j_q i_p} = 1 \qquad i=1,\ldots,m \tag{3-4}$$

$$y_{i_p j_q} \in \{0,1\} \ , \ \forall \ (\beta_i^p, \alpha_i^q) \in \bar{N} \tag{3-5}$$

Constraints (3-3) above are the flow conservation constraints and constraints (3-4) are set-partitioning type constraints expressing the fact that each task must be performed exactly once .

5 - Worst Case Analysis of the Size of network $\tilde{G}$.

If in the CSP there are m tasks , the maximum number of task-arcs in the expanded graph G is m(m+1)/2 which leads to m(m+1) + 2 vertices . This condition occurs when all task are disjoint in time and T (the shift time) is very large . If ,as in all real problems , the shift time allows a maximum of k tasks to be performed by a single crew-schedule then the maximum number of task-arcs in G is km . Under these conditions , the maximum number of non-required arcs terminating at the initial vertex of a task-arc is also k ; so that the maximum number of linking-arcs in G is k^2m . Therefore , the formulation defined by (3-2)-(3-5) contains at most k^2m variables and (k+1)m + 2 constraints . Thus , for any real-world CSP the problem in formulation (3-2)-(3-5) is of very much smaller size than the problem in formulation (2-1)-(2-4) , and well within the capabilities of existing LP codes .

6- Computational Results .

Use of linear programming to obtain bounds from the formulation presented in the previous section led to computational experience (see table 3-1) whereby in all 270 test problems generated and solved , the LP solution was integer in all cases , and hence no tree-search was necessary .

All problems were randomly generated in 4 groups as follows :

Group 1 : CSP's varied in size from 10 to 30 tasks , and a total of 100 problems were generated . For each problem the duty period was taken to be T=6 hours , and each task had a random starting time in the range 00.00 to 24.00 hours and a duration that varied randomly from 45 minutes to 2 hours 30 minutes . The cost coefficient c_{ij} of the linking-arcs were generated according to the formula :

$$c_{ij} = (1 + \alpha)\, d_{ij}$$

where d_{ij} is the duration (in minutes) from the end of task i to the beginning of task j , and α is a randomly generated constant in the range 0 to 1 . In all problems , the number of crews M assumed was the minimum number M* feasible for the CSP . A uniform distribution was used to generate all random numbers .

Group 2 : CSP's varied in size from 40 to 80 tasks , all other details as for group 1. A total of 60 problems were generated .

Group 3 : CSP's varied in size from 35 to 150 tasks with a duty period of T= 6 hours , but with no two tasks overlapping in time . The same randomly generated task duration as in group 1 was used but with an unlimited (instead of 24 hours) planning horizon . A total of 60 problems were generated .

Group 4 : CSP's generated from the same 25 task-problem by varying the cost coefficients thus producing 50 test problems .

From table3-1 it is seen that CSP's of quite practical size (about 100 or so tasks) can be solved optimally by the algorithm based on the formulation of section 4 . As a means of comparison , it is worthwhile to note here that problems in groups 2 and 3 were too large to be solved by the formulation (and algorithm) of the previous chapter .

Remark 1 : Finding the minimum number of crews , of the CSP , required to cover all tasks .

We have already mentioned in section 2-3 (chapter2) that the problem of finding the minimum number of crews (MNC) required to cover all tasks of the CSP is an NP-complete problem .The current formulation of the CSP (section 4) can be used for this purpose . The idea is based on the fact that if a number m of crews is used which is less than MNC then solving the formulation of the CSP with any LP package will lead to an infeasibility of the problem .

The method we have adopted to determine the MNC of all the 270 CSP's of table 3-1 was as follows :

(i) Use the greedy algorithm of section 2-4 (chapter 2) to determine a heuristic value of the MNC .Let m_0 be this value ;

(ii) By setting MNC = m_0, m_0-1, m_0-2,... we solve with the LP package the corresponding CSP's . Whenever an infeasibility is found then we know that MNC=m_0 + 1 where m_0 is the number of crews of the "infeasible" CSP .

It is worthwhile to mention that in more than 80% of the cases the MNC was obtained directly by the heuristic algorithm and for all the remaining problems the difference between the MNC and its heuristic value never exceeds 2 .

Remark 2 : An attempt to generate a CSP with a non-integer LP-solution.

If we can prove that all the extreme points of the linear polytope (say P) of the CSP , as formulated in section 4 , are integer then we can say that the CSP can always be solved with an LP package . To prove this is not an easy task . This is discussed in more details in section 8 .

Now , to prove that the CSP cannot always be solved with an LP package it suffices to produce a single problem for which the optimal solution of its linear relaxation is not integer . Problems of group 4 (see table 3-1) have been generated with this purpose in mind . All these problems were generated from the same 25 task-CSP by varying the cost coefficients . This means that they all have the same polytope (say P25) but have different objective functions . The optimal solution of each one of them correspond to a different extreme point of the polytope .

In doing so , we have tried to find a non-integer extreme point of the polytope P25 . But it was in vain . Although we can say that the 50 extreme points (of P25) we have considered are all integer this does not mean that there does not exist a non-integer extreme point of P25 . Simply because the number of these extreme points runs into thousands and to be able to claim this we have to keep varying the cost coefficients till all the extreme points of P25 are tested . Clearly , this is practically impossible . In these circumstances we think that some effort should be made in trying to prove that the CSP can always be solved with an LP package . This is discussed in section 8 .

Table 3-1 : Computational Performance of the Graph Expansion Algorithm.

Problem	Number of Tasks	Number of Problems	Size of the LP Problem [rowsxcolumns] minimum	average	maximum	Size of Expanded Graph [Task-Arcs] minimum	average	maximum	Time* Expansion Time minimum	average	maximum	LP Solution Time minimum	average	maximum
Group 1	10-14	20	25x 59	30x 75	38x 94	15	18	24	0.1	0.2	0.2	0.2	0.4	0.6
	15-19	20	41x 98	49x 132	57x 151	26	31	38	0.2	0.3	0.3	0.8	1.0	1.3
	20-24	20	59x 151	62x 171	66x 190	39	40	42	0.3	0.4	0.4	1.0	1.2	1.6
	25-29	20	77x 239	83x 263	91x 298	52	57	62	0.5	0.5	0.6	2.1	2.5	2.9
	30	20	88x 263	91x 298	92x 321	58	61	62	0.5	0.6	0.6	2.1	2.9	3.2
Group 2	40	20	129x 494	132x 509	153x 632	89	92	113	0.8	0.9	0.9	4.5	6.5	8.4
	60	20	211x 732	222x 864	262x 984	151	162	202	1.0	1.7	2.3	15.6	17.4	19.2
	80	20	385x2018	386x2179	390x2239	305	306	310	4.1	4.2	5.1	29.6	34.7	37.8
Group 3	35	10	112x 319	121x 323	128x 338	77	86	93	0.6	0.6	0.6	4.8	5.3	5.4
	50	20	180x 473	185x 482	192x 487	130	135	142	0.9	1.0	1.1	8.9	10.2	9.3
	70	20	314x1810	317x1821	332x1997	239	242	257	2.4	3.1	3.4	20.7	23.4	30.1
	150	10	722x6048	726x6119	737x6280	572	576	587	10.7	11.6	12.9	200.3	210.2	218.5
Group 4	25	50	77x 239	79x 242	80x 260	52	54	55	0.5	0.5	0.5	2.1	2.2	2.2

* Seconds of CYBER 855 (Fortran Compiler)

7- Extensions of the CSP .

Two straightforward extensions of the crew shceduling problem are considered in this section . Each extension corresponds to a more realistic version of the CSP . It basically consists of the CSP plus one or two additional constraints . The graph expansion technique was modified and applied to each problem accordingly .

Like the CSP , a linear programming package was sufficient to solve all the randomly generated problems .

Without loss of generality , we will assume that the CSP is an airline crew scheduling problem . Also we will suppose that each crew is associated with a single plane and vice versa . Hence the terms 'crew' and 'plane' will be used interchangably.

7-1 The Airline Crew Scheduling Problem with Rest Periods .

7-1.1 Introduction .

In the airline crew scheduling problem (ACSP) the planning period was assumed to be one day of 24 hours . This is indeed the case for most airline companies. However for some companies (the largest ones) a weekly planning period is much more desirable . This is because of the type of flights they cover . One such type consists of the intercontinental flights which can take up to 15 hours and can start at any time of the day (8.00am,4.00pm,3.00am,...etc...) . Consequently , the union regulations require that each crew who cover such type of flights should have a rest of at least a given number of hours (say 12) between any two consecutive trips .

The new airline crew scheduling problem ACSP1 is :
"minimize the cost of constructing crew pairings such that :

(a) each flight must be covered once by one and only one crew (the number of flights N and crews K are known);

(b) the planning period is one week of 7 days ;

(c) the flying duty period T_0 ,ie the maximum period of time during which a crew operate without stop , is fixed and the same for all crews ;

(d) the crew rest period (CRP) between any two consecutive trips must be greater than a time T_1 and smaller than T_2 (of course $T_2 > T_1$) "

A crew pairing consists of a set of trips separated by crew rest periods . A trip is in turn made up of a set of flight-legs covered by a crew without interruption .

Using the graph expansion algorithm A8 of section 3 , we first expand the network G representative of an ACSP into a network $\tilde{G}$ in which all paths satisfy condition (c) above . Then , network $\tilde{G}$ is in its turn modified into network $\bar{G}$ in which all paths satisfy (a),(b) and (d).

An example explaining this modification is given in section 7-1.2 . Then the general algorithm A9 for converting network G into G is described in section 7-1.3 . Based on G , a minimum cost network flow problem formulation plus additional partitioning type constraints has been devised .One hundred seventy randomly generated problems of size varying between 5 and 50 flights have been solved using just an LP package . The results are presented in section 7-3.

7-1.2 Example .Consider the ACSP1 represented by network G of figure 3-4 . In fact this is just a subgraph of the whole graph G . In reality , G contains many more flight-arcs since the planning period is supposed to be one week . In figure 3-4 only the first 10 flight-arcs have been represented . This will be enough to explain the main idea of algorithm A9 . Although the network of figure 3-4 is just a subgraph of G , in what follows we will refer to it as G .

The flying duty period T_0 is assumed to be 600 minutes , the minimum and maximum crew rest periods (CRP) are respectively equal to T_1=700 minutes and T_2=1000 minutes .Using algorithm A8 of section 3 , we obtain network $\tilde{G}$ of figure 3-5 . For the purposes of clarity the linking-arcs and sink-arcs of $\tilde{G}$ have not been represented .

Figure 3-4 : Network G .

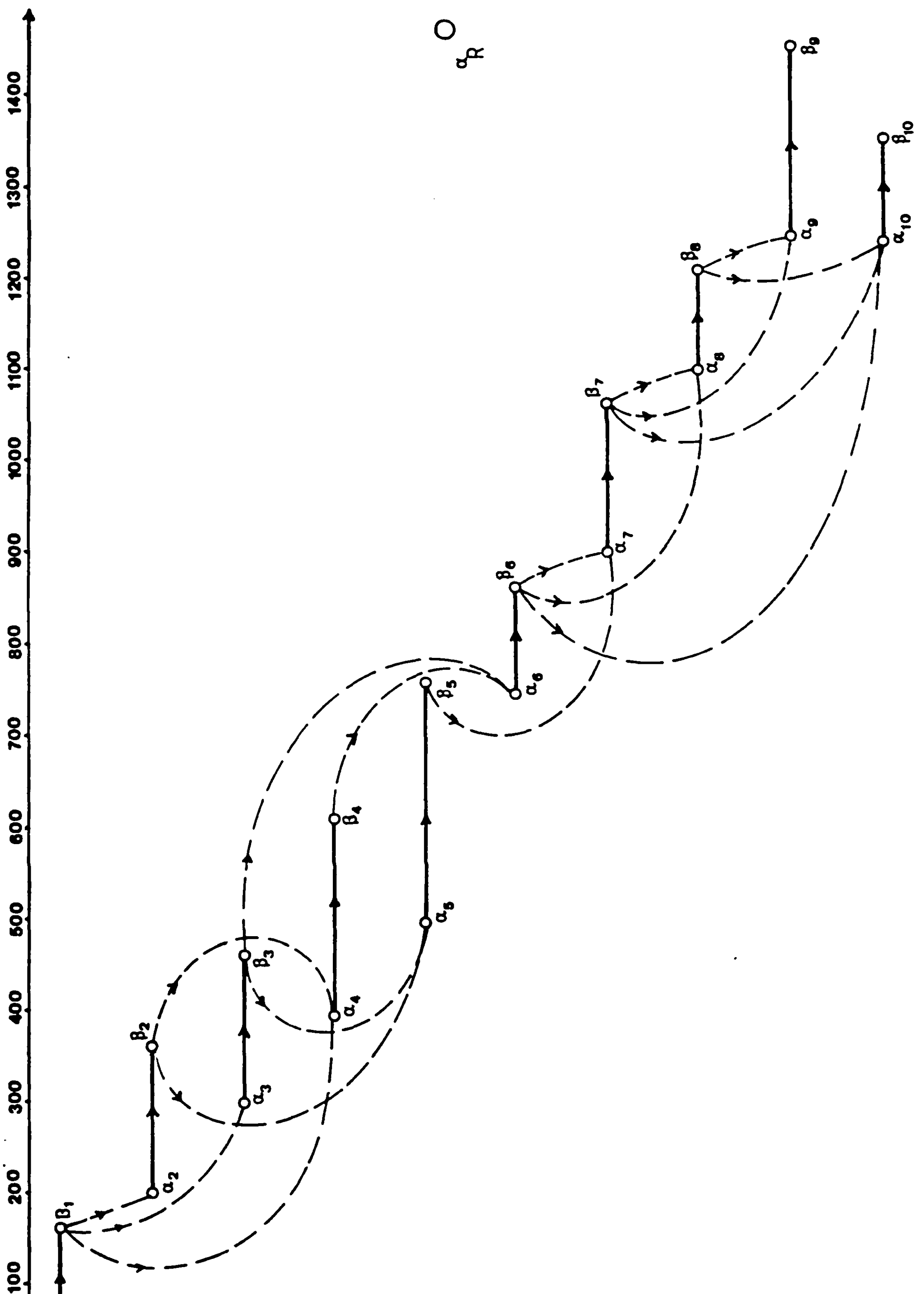


Figure 3-5 : Network $\tilde{G}$.

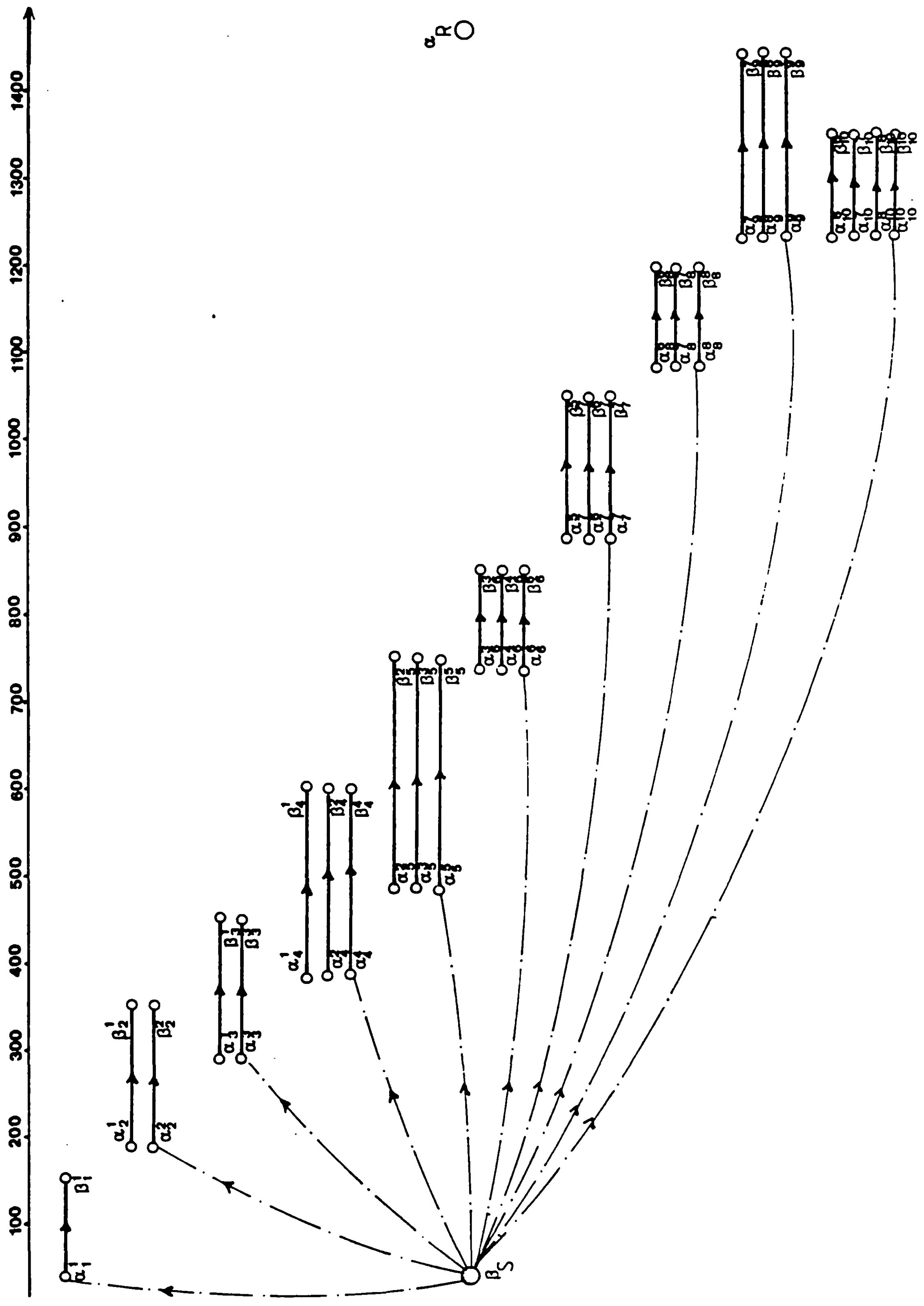
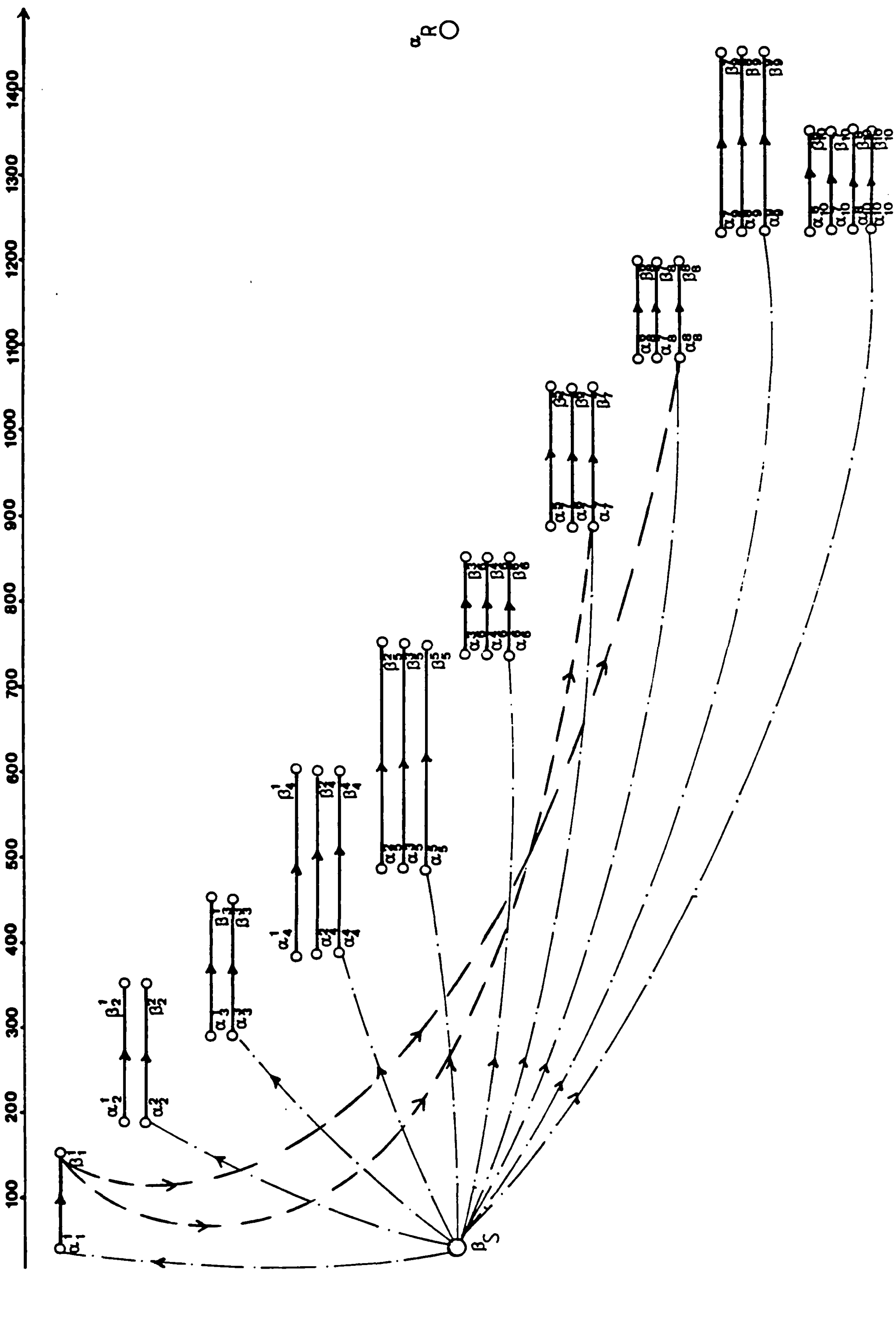


Figure 3-6 : Network $\bar{G}$.



If in $\tilde{G}$ we assume that a trip consists only of flight-arc (α_1^1,β_1^1) , a crew covering this trip will need a rest of minimum CRP of 700 minutes and maximum CRP of 1000 minutes . Hence the next trip covered by this crew will have either (α_7^7,β_7^7) or (α_8^8,β_8^8) as first flight-arc . This is because the difference d_1 (resp. d_2) between the starting time of (α_7^7,β_7^7) (resp. (α_8^8,β_8^8)) and the finishing time of (α_1^1,β_1^1) is such that $T_1 \leq d_1 \leq T_2$(resp. $T_1 \leq d_2 \leq T_2$). We express this fact by joining arc (α_1^1,β_1^1) with both (α_7^7,β_7^7) and (α_8^8,β_8^8). These arcs will be called "**rest-arcs**" and costs will be attached to them since they have the same function as the linking-arcs.

Applying the same reasoning to all flight-arcs of $\tilde{G}$ we get network $\bar{G}$ which is represented in figure 3-6 . It is exactly the same as network G except that it has rest-arcs in addition. For the sake of clarity only the rest-arcs departing from arc (α_1^1,β_1^1) have been represented.

7-1.3 Algorithm A9.

In this subsection , a detailed description of the algorithm that converts G into G is given . For each flight f of the ACSP we will use P(f) to be the set of all paths of G that have arc (i,j) (corresponding to f) as first flight-arc . G is the network representative of an ACSP .

First, algorithm A8 is applied to network G to obtain a network $\tilde{G}$ in which no path has length greater than the flying duty period . Subsequently , $\tilde{G}$ is used as input for algorithm A9 which consists of the following :

Step 0 : i = 0 ; GOTO STEP 1;

Step 1 : i = i + 1 . Consider flight i and set j = i . GOTO STEP 2 ;

Step 2 : j = j + 1. Consider flight j (j>i) and compute

d = starting time of flight j - finishing time of flight i

GOTO STEP 3 ;

Step 3 : If d<T1 or d>T2 GOTO STEP 4 ;Else link all flight-arcs of $\tilde{G}$ corresponding to flight i with the flight-arc corresponding to P(j) .GOTO STEP 4 ;

Step 4 : If all flights j's have been scanned GOTO STEP 5 ; Else GOTO STEP 2;

Step 5 : If all flights i's have been scanned STOP ; Else GOTO STEP 1 .

The output of algorithm A9 is network $\bar{G}$ and the new arcs generated in step 3 are called "rest-arcs" .

7-2 The Multiple Depot Airline Crew Scheduling Problem [35].

7-2.1 Introduction .

In the airline crew scheduling problem (ACSP) , we assumed implicitly that there was one single depot from which all the planes depart and to which they all return . Also the costs of bringing a plane from the depot to any flight's starting place and from any flight's ending place to the depot were assumed to be nil .

This section deals with a more practical version of the problem . That is to suppose that the airline company has not only one depot but several depots . Also each depot has got a known capacity ie it cannot house more than a given number of planes. This problem is called the multiple-depot ACSP (DCSP or ACSP2) and is defined as follows:

"minimize the cost of covering a certain number of flight-legs by a given number of planes (or crews) such that :

(a) each crew is assigned to only one plane ;

(b) each flight must be covered by one and only one plane ;

(c) the planning period is one day of 24 hours ;

(d) the flying duty period T_0 is fixed and the same for all crews ;

(e) the company has a given number of depots (say L) ;

(f) each plane must return , at the end of its journey , to the same depot from which it has departed ;

(g) costs are attached to the operation of bringing a plane from a depot to a flight's starting place or from a flight's finishing place to the depot ."

The network representative G_D of an ACSP2 is exactly the same as the network G representative of an ACSP except that in G_D we assume L super-sources and L super-sinks (L is the number of depots).Since each flight of ACSP2 can be covered by one and only one plane which comes from one depot , we can represent each flight by L flight-arcs (one for each depot) . Linking the flight-arcs corresponding to the same depot together in the same manner as in G we obtain L different networks , each with a source β_S^i and sink α_R^i corresponding to depot i . A unique network G_D results by joining all sources $\beta_S^1,\beta_S^2,...,\beta_S^L$ (resp. sinks $\alpha_R^1,\alpha_R^2,...,\alpha_R^L$) to a super-source β_S (resp. super-sink α_R).

An example explaining how to obtain G_D from G is given in section 7-2.2 and the general procedure is given in section 7-2.3.Applying algorithm A8 of section 3 to each one of the subnetworks of G_D we obtain a network $\bar{G}_D$ in which all the above conditions of the ACSP2 are satisfied .We conclude this whole section by giving some good computational results . Thus , for all the 170 randomly generated problems , the optimal integer solution was found in all cases by just using an LP package . The results are presented in section 7-3 .

7-2.2 Example .

Consider the 6 flight - 4 planes ACSP2 represented by network G of figure 3-7 in which two depots of capacity 3 and 2 respectively are given . For the sake of clarity the source and sink-arcs have not been represented.Also let us assume that the flying duty period is 500 minutes . Source β_S^i (i=1,2) and sink α_R^i(i=1,2) correspond to depot i (i=1,2).

This network is clearly the same as the network representative of an ACSP except that we assume 2 sources and 2 sinks .Each flight i (i=1,...,6) can be covered by a plane coming either from depot 1 or depot 2 . Hence we can represent each flight i

Figure 3-7 : Network G .

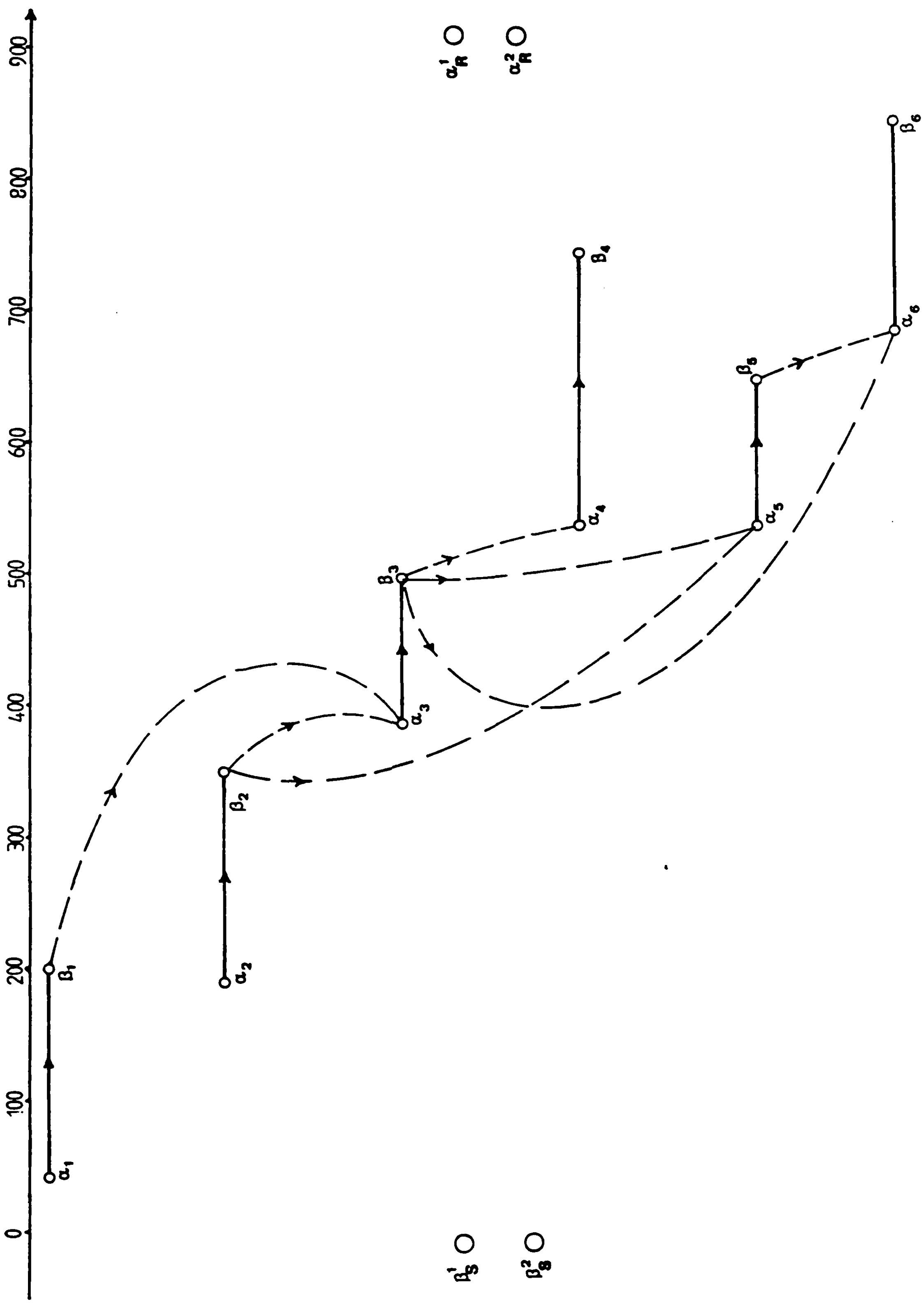
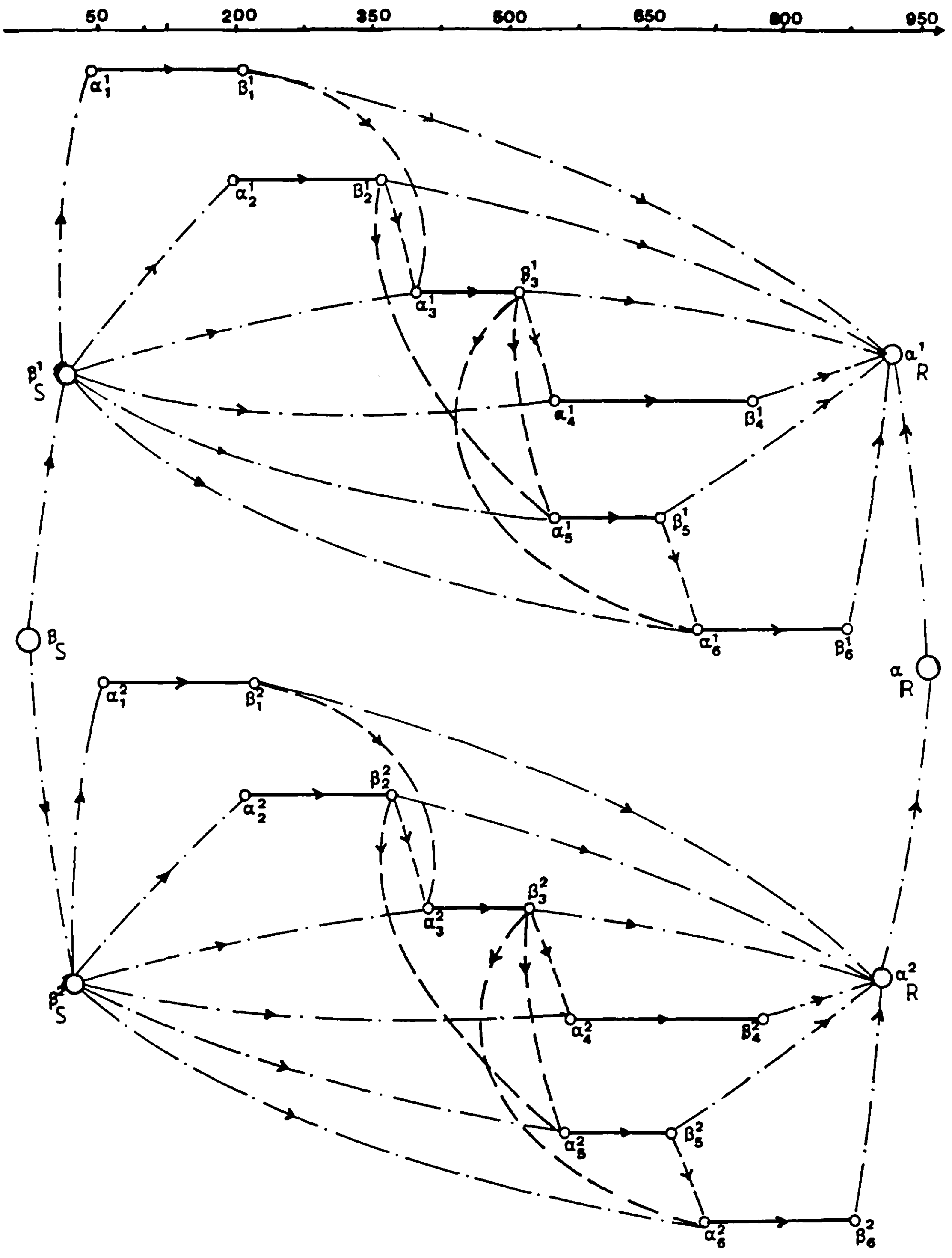


Figure 3-8 : Network G_D .



by 2 flight-arcs , one corresponding to depot 1 and the other to depot 2 . If we link together all the flight-arcs corresponding to the same depot , and then join them to the corresponding source and sink , we obtain 2 similar networks . Finally joining β_S^1 and β_S^2 (resp, α_R^1 and α_R^2) to a super-source (resp. super-sink) we obtain network G_D of figure 3-8. The following remarks concerning G_D can be made :

(a) Network G_D is made up of 2 subnetworks corresponding respectively to the 2 depots ;

(b) each subnetwork of G_D can be considered as a network representative of 6 flights ACSP. Consequently algorithm A8 of section 3 can be applied separately to each subnetwork in order to obtain 2 subnetworks G_D^1 and G_D^2 in which all paths have duration less than or equal to the flying duty period ;

(c) costs are attached to each linking-arc , source-arc and sink-arc ;

(d) arc(β_S,β_S^1) (resp. (β_S,β_S^2)) and (α_R^1, α_R) (resp.(α_R^2, α_R)) have capacity 3 (resp.2);

(e) the total input flow (from β_S) is 4 (ie the number of planes).

7-2.3 Algorithm A10.

In this section , we present a procedure for constructing the network G_D representative of a ACSP2 with n flights , K planes and L depots (d_1,...,d_L) of known capacities.As input to the algorithm we have network G representative of the ACSP which consists of the same flights as the ACSP2 and the same number of planes .

Step 1 : Construct L disjoint networks G_1,...,G_L all identical to G . Network G_i ($1 \leq i \leq L$) corresponds to depot d_i and has got source β_S^i and sink α_R^i .GOTO STEP2 ;

Step 2 : Join all sources β_S^1,..., β_S^L with a super-source β_S and all sinks α_R^1,..., α_R^L with a super-sink α_R .GOTO STEP 3;

Step 3 : Set the capacity of each arc (β_S , β_S^i) and (α_R^i, α_R) equal to that of the depot d_i ; set the capacities of all other arcs to 1 ; GOTO STEP 4 ;

Step 4 : Attach costs to the source-arcs and sink-arcs of each network G_i ($1 \leq i \leq L$).

Set the input flow equal to the number of planes of the problem .

As output to this algorithm we obtain network G_D representative of a ACSP2. Since each G_i (i=1,...,L) of G_D is identical to G if we apply algorithm A8 to each one of them we obtain a network $\bar{G}_D$ made up of L networks $\bar{G}_1,...,\bar{G}_L$ all identical to $\bar{G}$ in which all paths have duration smaller than or equal to the flying duty period . Each flight of the ACSP2 has in each $\bar{G}_i$ (i=1,...,L) several corresponding flight-arcs . Hence out of all the flight-arcs of $\bar{G}_D$ corresponding to that flight only one will be in the optimal solution .

7-3 Computational Results for the Two CSP's Extensions .

For each extension of the CSP (table 3-3) a total of 190 problems were randomly generated in two groups :

Group 1 : Problems varied in size from 10 to 50 tasks . A total of 140 problems were genrated for each one of the multiple depot CSP and the CSP with rest periods . The cost coefficients c_{ij} of the linking-arcs were generated as in section 6,ie according to the formula :

$$c_{ij} = (1 + \alpha)\, d_{ij}$$

where d_{ij} is the duration (in minutes) from the end of task i to the beginning of task j and α is a randomly generated number in the range 0 to 1 . Also , in all problems the number of crews was taken to be the minimum number M* feasible for each problem . The way of finding M* has already been explained in section 6 (remark 1) . All the other details are given in table 3-2.

Group 2 . For each CSP's extension , the problems were generated from the same 25 task-problem by varying the cost coefficients thus producing 50 test problems . By doing this we were trying to generate a problem with a non-integer LP solution . This has already been explained and discussed in section 6 .

Table 3-2 : Some Details about the Randomly Generated Process of the Problems of Table 3-3.

	CSP with rest period	Multiple Depot CSP
Task Starting Time	Vary uniformly in the range [0,1week]	Vary Uniformly in the range [0,24hours]
Maximum Duty Period (hours)	12	6
Maximum Rest Period (hours)	16	N/A
Minimum Rest Period (hours)	12	N/A
Number of Depots	N/A	2 depots for DCSP's of up to 20 tasks; 3 depots for larger DCSP's.

N/A : non-applicable.

Table 3-3 : Average Computational Results for the 2 CSP's Extensions.

Problem	Number of Tasks	Number of Problems	Size of LP+ [rowsxcolumns] CSP1*	Size of LP+ [rowsxcolumns] CSP2**	Size of G [Task-Arcs] CSP1	Size of G [Task-Arcs] CSP2	Time++ Expansion Time CSP1	Time++ Expansion Time CSP2	Time++ LP Solution Time CSP1	Time++ LP Solution Time CSP2
	10-14	20	39x 108	52x 167	26	40	0.2	0.4	0.7	1.1
	15-19	20	69x 200	87x 289	49	68	0.3	0.7	1.9	2.5
	20-24	20	96x 332	152x 450	73	126	0.5	1.1	4.1	8.1
Group 1	25-29	20	111x 399	183x 860	82	155	0.7	1.6	4.7	10.2
	30	20	104x 398	213x 899	72	180	0.7	2.2	4.7	15.6
	40	20	144x 600	342x1701	102	300	1.2	4.0	7.7	31.2
	50	20	263x 996	473x1958	210	420	4.1	10.1	19.5	66.8
Group 2	25	50	116x 420	189x 780	88	162	0.7	1.6	5.2	11.3

* CSP1 = CSP with rest period ;
** CSP2 = Multiple depot CSP ;
+ Size of LP = Size of Linear Programming Problem ;
++ Seconds of CYBER 855 (Fortran Compiler) .

8-APPENDIX : Are the Extreme Points of the CSP Polytope Integer ?

As a consequence of the results obtained in the previous section , the natural question that comes to one's mind is :
"Does the linear relaxation of the formulation of the CSP always provide the optimal integer solution of the problem ?"

In this section , we will make an attempt to answer this question .

Our problem is in fact just a particular case of the following more general problem .

Consider the integer programming problem (P)

$$\min \ cx \tag{3-6}$$

subject to

$$Ax = b \tag{3-7}$$

$$x \text{ integer} \tag{3-8}$$

$$x \geq 0 \tag{3-9}$$

and its relaxation (LP) (ie constraint (3-8) is dropped) ; where

A is an mxn matrix of real numbers ;

C is a non-negative vector of n real numbers

and b is an m-vector of real numbers ;

Let $P' = \text{conv}\ \{ x \varepsilon R^n \mid Ax=b \ ;\ x \text{ integer} \}$; P' is the convex hull of the integer points of P ;and let $P'' = \{ x \varepsilon R^n \mid Ax=b \ ;\ x \geq 0 \}$.

The question is : "under which conditions on the matrix A and the m-vector b ,do (P) and (LP) have always the same optimal solution , independently of the cost vector c ? "

The first paper that dealt with this problem is the one published in 1956 by *Hoffman and Kruskal* [89] who gave a necessary and sufficient condition for the 2 polytopes P' and P" to be equal . If these 2 polytopes are equal then , independently of the cost c considered , the two problems (P) and (LP) will have always the same optimal solution . At this point it is worthwhile to remark that the converse is not true (as we will see later). Before stating the theorem of Hoffman and Kruskal we need to give the following necessary definition :

Definition : A matrix A is totally unimodular if and only if the determinant of each square submatrix of A is equal to 0,1 or -1 .

Theorem 1 (*Hoffman and Kruskal*) : If the matrix A of problem (P) is totally unimodular then P' = P".The converse holds true .

An example of problems that have totally unimodular matrices is the class of network flow problems .It is well known that for this class of problems the integer problem and its linear relaxation have always the same optimal solution[17].

In our case , we are only interested in the first part of the theorem . However , with just definition 1 in hand it was not an easy task to determine if the matrix A of the CSP was totally unimodular . Fortunately , the work of *Camion* [36] came to our rescue . In that paper it was proved that :

Theorem 2 (*Camion*) : A matrix A (with 0,+1 and -1 entries) is totally unimodular if and only if for every square Eulerian submatrix $A_{I,J}$ we have :

$$\sum_{i,j} a_j^i \equiv 0 \quad \text{modulo } 4 \tag{3-10}$$

where

Definition: A submatrix $A_{I,J}$ of A is said to be Eulerian if and only if :

$$\forall i \in I \quad \sum_{j \in J} a_j^i \equiv 0 \mod 2 \tag{3-11}$$

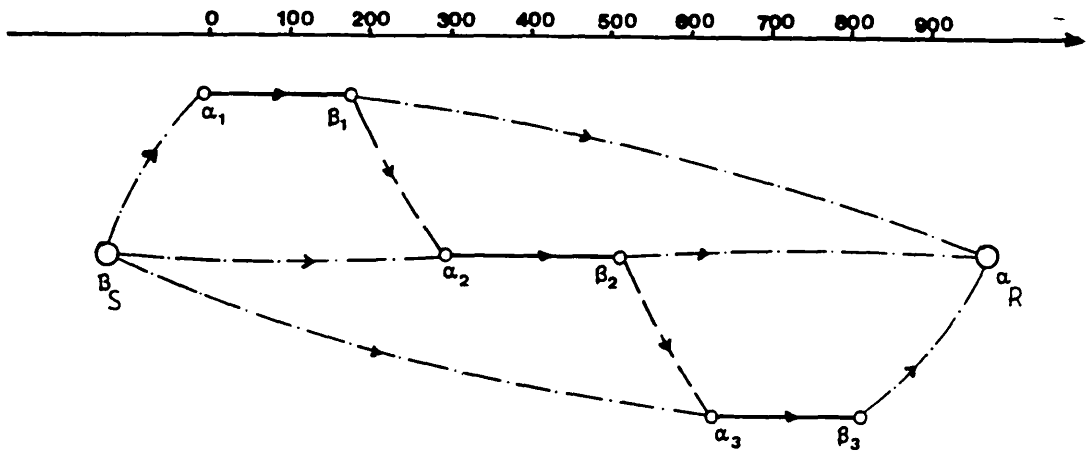
$$\forall j \in J \quad \sum_{i \in I} a_j^i \equiv 0 \mod 2 \tag{3-12}$$

in other words if the sum of elements in any column or in any row of $A_{I,J}$ is even .

After applying theorem 2 to our case (note that the matrix of the CSP has only 0,+1 and -1 entries) we found out that the matrix of the CSP is not totally unimodular. The following is a counter-example .

Example : Consider ,in figure 3-9, network G representing a 3 task-CSP and its expansion , network $\bar{G}$ in figure 3-10.

Figure 3-9 : Network G.



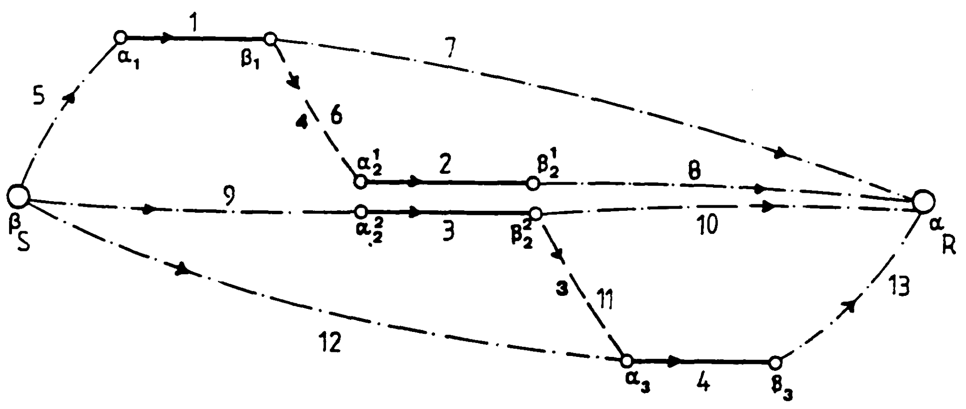


Figure 3-10: Network $\bar{G}$.

Table 3-4 : Matrix A

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	1	1	0	0	0	0	0	0	0	0	0	0
2	-1	0	0	0	0	1	1	0	0	0	0	0	0
3	1	0	0	0	-1	0	0	0	0	0	0	0	0
4	0	-1	0	0	0	0	0	1	0	0	0	0	0
5	0	1	0	0	0	-1	0	0	0	0	0	0	0
6	0	0	-1	0	0	0	0	0	0	0	1	1	0
7	0	0	1	0	0	0	0	0	0	-1	0	0	0
8	0	0	0	-1	0	0	0	0	0	0	0	0	1
9	0	0	0	1	0	0	0	0	0	0	-1	-1	0
10	0	0	0	0	1	0	0	0	1	0	0	1	0
11	0	0	0	0	0	0	-1	-1	0	-1	0	0	-1

Table 3-5 : Matrix $A_{i,J}$

	1	2	3	5	7	8	9
1	0	1	1	0	0	0	0
2	-1	0	0	0	1	0	0
3	1	0	0	-1	0	0	0
4	0	-1	0	0	0	1	0
7	0	0	1	0	0	0	-1
10	0	0	0	1	0	0	1
11	0	0	0	0	-1	-1	0

Let A be the corresponding constraints matrix . It is represented in table 3-4.

If in matrix A we consider the submatrix $A_{I,J}$ which consists of :

columns 1,2,3,5,7,8 and 9 ; and

rows 1,2,3,4,7,10 and 11

we can easily see that $A_{I,J}$ (see table 3-5) is Eulerian , since the sum of elements in any row and in any column is even . However ,

$$\sum_{i,j} a_j^i = \text{sum of all elements of } A_{I,J} = 2$$

i.e. $$\sum_{i,j} a_j^i \not\equiv 0 \mod 4$$

Hence A is not totally unimodular .

The fact that the matrix of the CSP is not totally unimodular does not answer our initial question . Thus , if P'= P" this does not mean that (P) and (LP) cannot have always the same optimal solution .For example *Berge* (1972) proved that if a matrix A is balanced , but not necessarily unimodular then

$$\mathrm{SOL(P)} = \mathrm{SOL(LP)} \tag{3-13}$$

where SOL(P)(resp. SOL(LP)) is the optimal solution of (P)(resp.(LP)). This implies, of course, that each unimodular matrix is balanced .

Two years later, in 1974,*Padberg* generalized the notion of balanced matrices to introduce the notion of perfect matrices . He showed that if a matrix A is perfect then (3-13) is satisfied . He also proved that each balanced matrix is perfect.

Unfortunately, these 2 results (of *Berge and Padberg*) cannot be applied to our case because they are restricted to 0-1 matrices only and the matrix of the CSP has 0,+1 and -1 entries .Consequently, we will limit ourselves to just giving the corresponding references [30] for *Berge* and [128] for *Padberg* .

From now on, we will consider a completely different approach from the one discussed above . Since we know that the class of network flow problems have totally unimodular matrices then all we need to do is to check if the CSP is equivalent to a network flow problem . In which case , we would have answered our question .

This idea was first suggested by *Iri* [91] who gave a necessary and sufficient condition for an integer programming problem to be convertible into a network flow problem .

For this purpose , the matrix A of the IP problem must be transformed first, by row operations and column permutations , into a matrix of the form

$$A = [I / L]$$

where I is an mxm identity matrix ; and

L is an mx(n-m) matrix .

Theorem 3 (*Iri* 1966) . Let A be the matrix of an integer programming problem (P) . The two following statements are equivalent :

(i) (P) can be converted into a network flow problem ;

(ii) L is the cut-set matrix of a graph. .

To define the cut-set of a graph we need to introduce the following notions :

A graph G is connected if there is a path from any vertex to any other vertex ie G has to be in one piece . An example of connected graph (G_1) is given in figure 3-11.Removing edge e_5 from G_1 we obtain the disconnected graph G_2 shown in figure 3-12.

Figure 3-11: A Connected Graph G_1

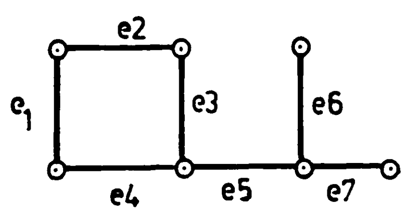
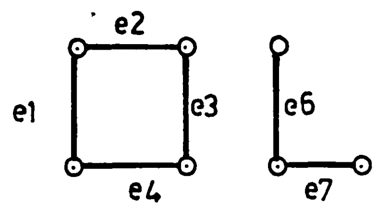


Figure 3-12: A Disconnected Graph G_2



Now,given a graph G=(X,E) where X is the set of vertices and E is the set of edges , a disconnecting set of G is a set of edges ,say D⊂E, whose removal disconnects G.Futhermore , if D does not contain any disconnecting set ie if no proper subset of D is a disconnecting set then D will be called "cutset".

In graph G_3 of figure 3-13 the set $D=(e_1,e_{10},e_9,e_4)$ is a disconnecting set. The disconnected graph G_4 obtained after the removal of E is shown in figure 3-14.

Figure 3-13: Graph G_3

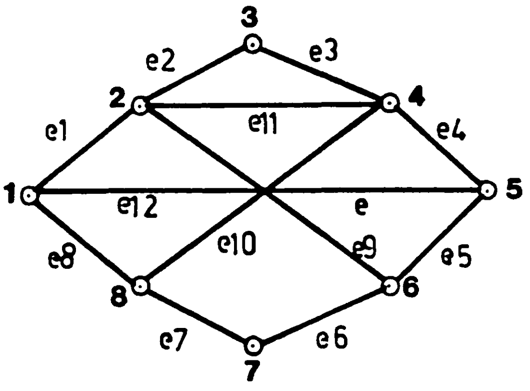
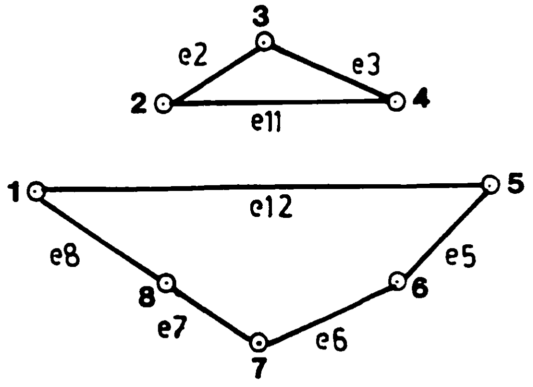


Figure 3-14: Graph G_4



Finally , the set $D'=(e_2,e_3)$ is a cutset in that it is a disconnecting set and none of its subsets (ie $\{e_2\}$ or $\{e_3\}$) disconnects the graph.For a more rigourous definition of cut-sets and spanning-trees the reader is refered to [29,39,54,82].

Now, going back to Iri's theorem we have not been able to prove or disprove that in general the matrix A of a CSP is the cutset matrix of a graph . However for some given CSP's (by "given" we mean that the input data of the problem are known) condition (ii) of Iri's theorem has been proved to be satisfied . The following is one such example :

Example :

Consider the network G of figure 3-9 and its expanded network $\bar{G}$ (figure3-10). The constraints matrix corresponding to the formulation of the CSP is presented in table 3-4.

By row operations and column permutations , matrix A becomes A' as shown in table 3-6. In fact it has been transformed in such a way to have the required form

Table 3-6 : Matrix A'

	1	2	3	4	5	6	7	8	9	11	10	12	13
1	1	0	0	0	0	0	0	0	0	0	1	0	1
2	0	1	0	0	0	0	0	0	0	0	1	-1	1
3	0	0	1	0	0	0	0	0	0	0	-1	1	-1
4	0	0	0	1	0	0	0	0	0	0	0	0	-1
5	0	0	0	0	1	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	0	0	0	0	1	-1	1
7	0	0	0	0	0	0	1	0	0	0	0	1	0
8	0	0	0	0	0	0	0	1	0	0	1	-1	1
9	0	0	0	0	0	0	0	0	1	0	-1	1	-1
11	0	0	0	0	0	0	0	0	0	1	0	1	-1
10	0	0	0	0	0	0	0	0	0	1	0	1	-1

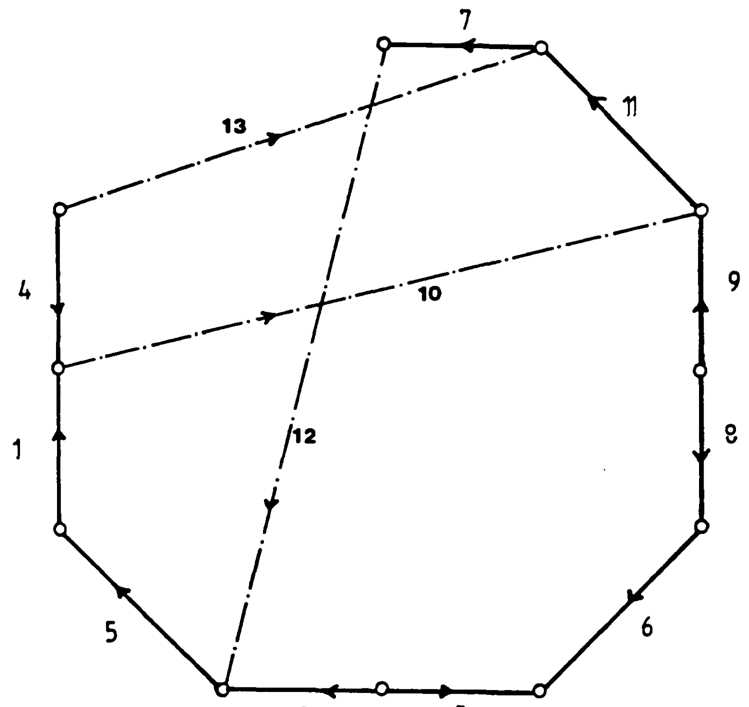


Figure 3-15 : Graph G_A

A'=[I/L].According to the definition of the cut-set matrix of a graph , it is clear that A' is the cutset matrix of graph G_A of figure 3-15. Each column of A is represented by an arc in G_A. The arcs corresponding to the columns of the identity-matrix I are in dotted lines and form a spanning-tree .This shows that the CSP represented by the graph of figure 3-4 can be exactly soved by a linear programming algorithm.

CHAPTER 4

THE GENERAL CREW SCHEDULING PROBLEM

1- Introduction .

The general CSP (GCSP) which was introduced in chapter 1 and which will be considered in this chapter and in the next one consists of a CSP plus the two following conditions :

(i) In all problems considered so far the scheduling problem has always assumed that the starting and ending times for each task were specified . Complications arise , however , if there exists a time-window in which a task must be carried out : that is to say , task i must be completed between 8.00am and 8.30am . With no time-windows , the set of tasks that can follow any particular task can be specified a priori and an acyclic network as the one representing the CSP can be constructed . With time-windows , the set of feasible tasks that can follow a given task cannot be specified beforehand and hence this acyclic network cannot be formed . For

Figure 4-1 : Network G

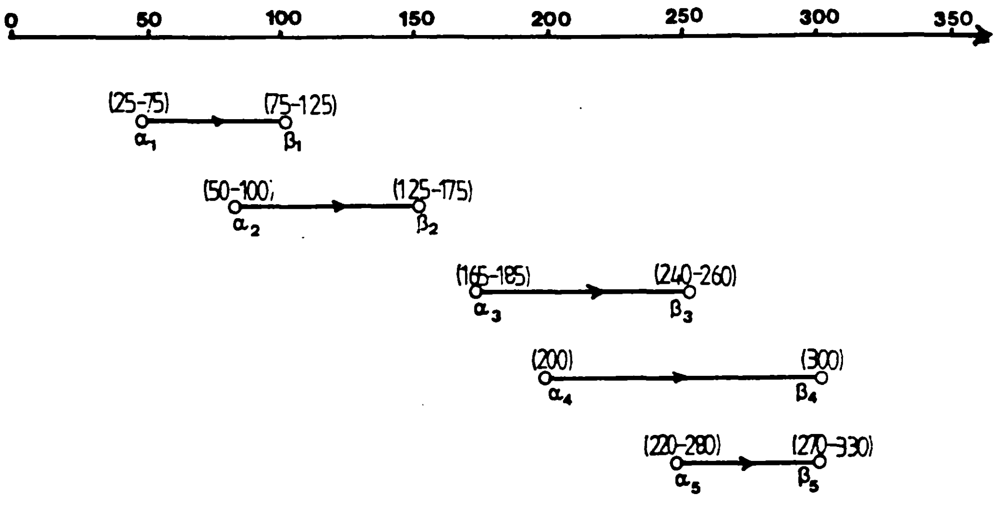


Table 4-1 : A 5 Task-GCSP .

Task	Window Starting time (minutes)	Duration time (minutes)
1	25-75	50
2	50-100	75
3	165-185	75
4	200-200	100
5	220-280	50

example consider the GCSP presented in table 4-1 . The work duty period is assumed to be equal to 200 minutes . The graph G which consists only of the task-arcs is represented in figure 4-1.

To show that an acyclic network cannot be formed consider tasks 1 and 2 . If a crew start their trip by covering task 1 at any time between ,say 25 and 30 minutes , they will be able to next cover task 2 if it starts at time τ_2 greater than 80 minutes . However if the trip of this same crew starts ,say at time 75 minutes , then it will be impossible for them to cover task 2 . Now even if we decide to link in graph G task-arcs 1 and 2 with a linking-arc we will be ignoring a number of feasible solutions that can be optimal and consequently the resulting solution will only be approximate .

(ii) Assuming that the crew and the corresponding vehicle form a single resource package the problem becomes more difficult if we allow the possibility that vehicles with different characteristics are available to service the tasks . In most cases the characteristic is vehicle capacity . For example , in the school bus scheduling problem [31] mini-buses can service the small schools and the regular buses can service the large schools and either vehicle can service the medium size schools . For each task the set of vehicles that may service it is specified .Assuming we want to cover the tasks of the GCSP of table 4-1 with 3 crews who will be using 2 vehicle types , the optimal solution to the problem is depicted in figure 4-2. The cost c_{ij} of linking 2 tasks i and j in the optimal solution (or more precisely following task i with task j) is the difference of time between the starting time of task j and the finishing time of task i . The types of vehicle that can cover any task are given in table 4-2 .

The general CSP (GCSP) consists in fact of 2 extensions of the CSP . If we ignore the existence of time-windows ie if the starting and the finishing times of each task are specified beforehand then we will be dealing with a multiple vehicle type CSP (VCSP) which is similar to the multiple depot CSP (see section 7-2 of chapter 3) . Now if in the GCSP we assume there is only one type of vehicle the resulting problem will consist of solving a CSP with the additonal requirement that the due times are given within time windows . This problem will be refered to as basic time-window CSP (TCSP) . Each one of these two problems (ie VCSP and TCSP) is clearly harder than the CSP .

Figure 4-2 : Optimal Solution of the 5 Task-GCSP.

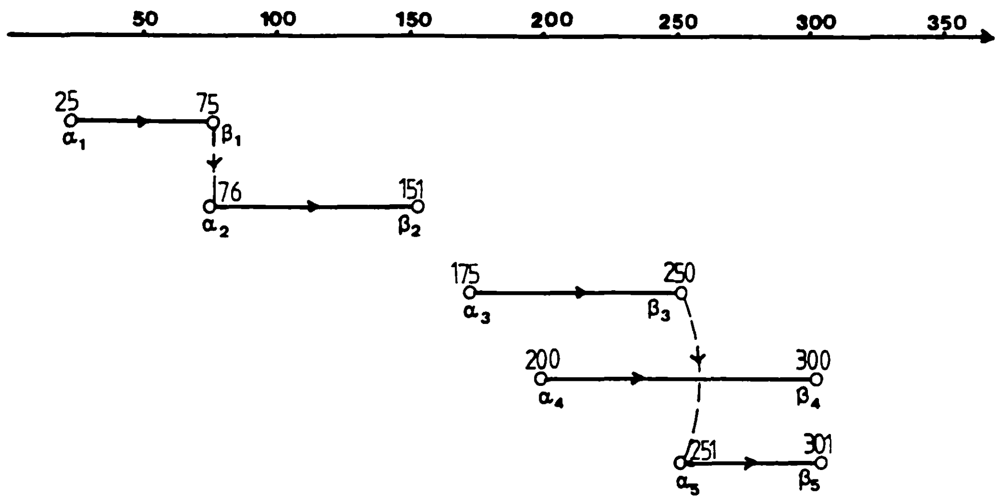


Table 4-2 : Vehicle Types' Assignment to the Tasks of the 5 Task-GCSP.

Task	1	2	3	4	5
Type of Vehicle	1	1;2	1;2	2	2

Motivated by the series of successful results obtained with the application of the graph expansion technique (GET) to the CSP and its extensions we will attempt to generalize GET to more general problems .In section 2 we will be describing the application of

GET to the TCSP . The results obtained , when the "average time window" is narrow, proved to be of the same quality as those obtained with the CSP . In this context the "average time-window " is the sum of all the task time-windows divided by the number of tasks. When the "average time-window" gets bigger the size of the expanded graph becomes so large that no existing LP package can solve the corresponding linear program . Consequently , only small TCSP's of size up to 15 tasks could be solved to optimality when the time-window can be very large ie take any value between 00.00 and 24.00 hours .In section 3 the application of GET to the VCSP is shown to be as efficient as in the case of the multiple depot CSP (see previous chapter) . All one hundred ninety randomly generated problems of size varying between 10 and 50 tasks were solved optimally using just an LP package . The expanded graph was constructed using an algorithm similar to that used for the multiple depot CSP .Section 4 deals with the application of GET to GCSP . It is particularly shown via extensive computational results that GET is very efficient provided the "average time-window" is small . Unfortunately as it increases,the size of the problem that existing LP packages can handle decreases. To overcome this drawback other approaches based on integer programming techniques will be considered in the next chapter to deal with the case where the "average time-window" gets larger .However we also consider a modified version of GET which , although it produces an expanded network of about the same size as the one obtained with the previous GET it is particularly useful when considering the GCSP with rest periods . This is discussed in section 5 .More than one hundred fifty problems of each type (ie TCSP,VCSP and GCSP) were randomly generated and solved to optimality using just an LP package . The computational results are presented in sections 6 .

2- The Basic Time Window Crew Scheduling Problem (TCSP).

2-1 Introduction .

The basic time-window crew scheduling problem (TCSP) is defined as follows: " minimize the cost of covering a certain number of tasks (n,say) by a given number (K , say) of crews in such a way that :

(a) each task must be covered once only by one crew ;

(b) the planning period is one day of 24 hours ;

(c) the work duty period is the same for all crews ;

(d) each task can have either :

(d1) a fixed and specified starting-time ;

(d2) a starting time given within a time-window ;

(e) the duration of each task is constant regardless of when it starts ."

Unlike network G representative of a CSP , network G_0 representative of a TCSP is not acyclic . However with a slight modification of G_0 , we can obtain a network G_t which has the same properties as G .

Applying the graph expansion algorithm A8 (see section 3, chapter 3) to G_t we get a network $\overline{G}_t$ in which all paths have duration less than or equal to T . An example explaining the conversion of G_0 into G_t is first given in section 2.2 . Then the general procedure of modifying G_t into $\overline{G}_t$ is explained in section 2.3 . Section 2.4 deals with the formulation of TCSP (based on network $\overline{G}_t$) which is a minimum cost network flow problem plus additional partitioning type constraints . As with the CSP , an LP package was sufficient to solve all 170 randomly generated problems of size varying between 10 and 50 tasks . The results are presented in section 6 .Before considering the next section it is worthwhile to mention that heuristic algorithms have been devised to solve the TCSP or its variants [55,115,125,137].

2-2 Example .

Consider the TCSP represented by the data of table 4-3 , in which the work duty period is assumed to be 7 hours 30 minutes . Except for task2 , the starting times of all tasks are given within a time-window . For example task 4 starts between 12.10am and 12.30am and hence finishes between 14.10am and 14.30am.

Table 4-3 : A 4 Task-TCSP .

Task	Window Starting time (hr & mn)	Duration time (minutes)
1	6.50-7.00	90
2	9.00	120
3	12.00-12.20	150
4	12.10-12.30	120

The corresponding graph G_0 , in which only the task-arcs are represented , is depicted in figure 4-3 . Unlike network G representative of a CSP it is impossible to represent the linking-arcs in G_0 . Indeed, if we assume that task 1 starts at 7.00 and task 3 at 12.00 it will be possible to link the two corresponding task-arcs in G_0 . However if task 1 starts at 6.50 and task 3 starts at 12.10 it will be impossible to link them .

Figure 4-3 : Network G_0

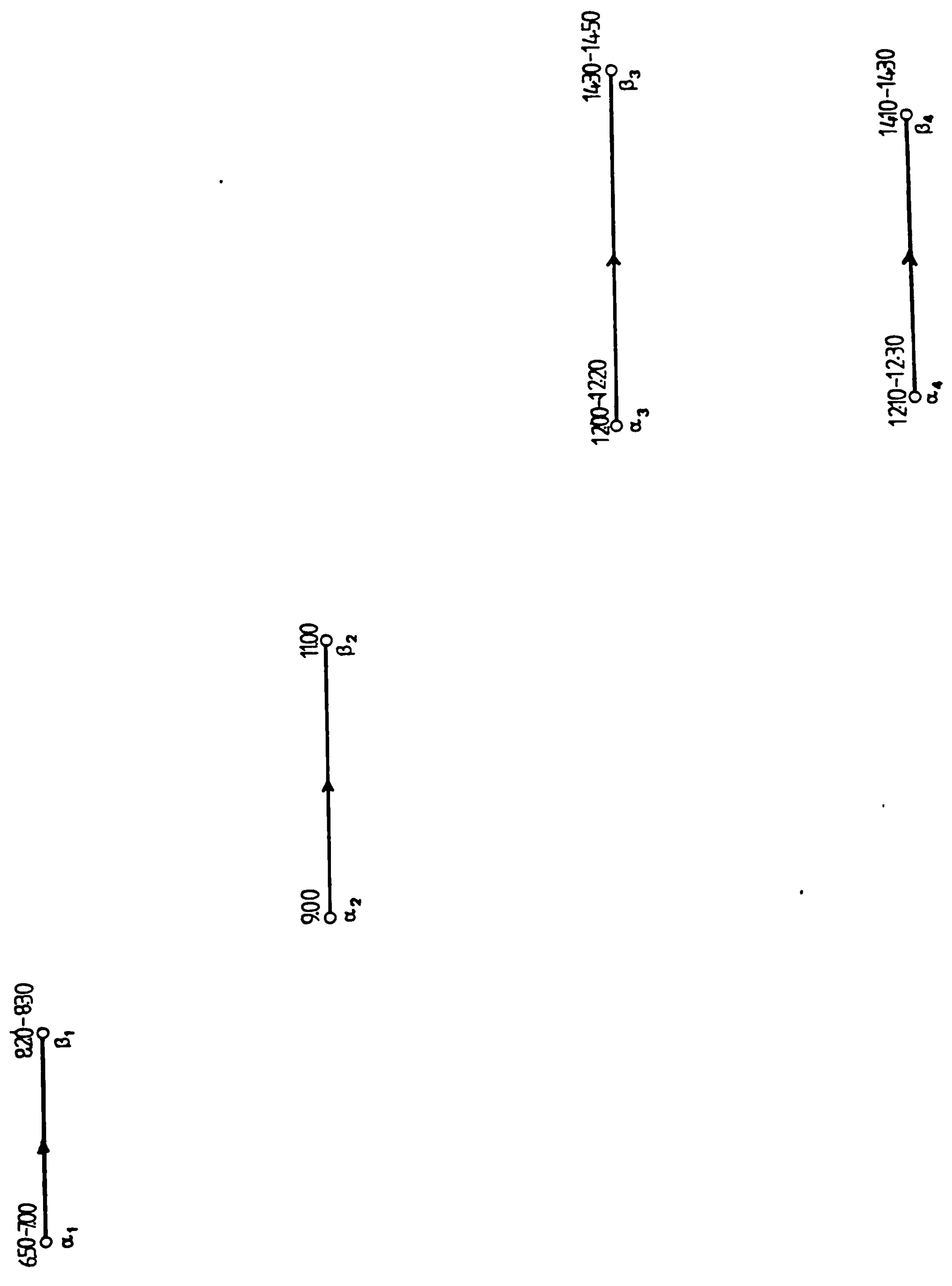
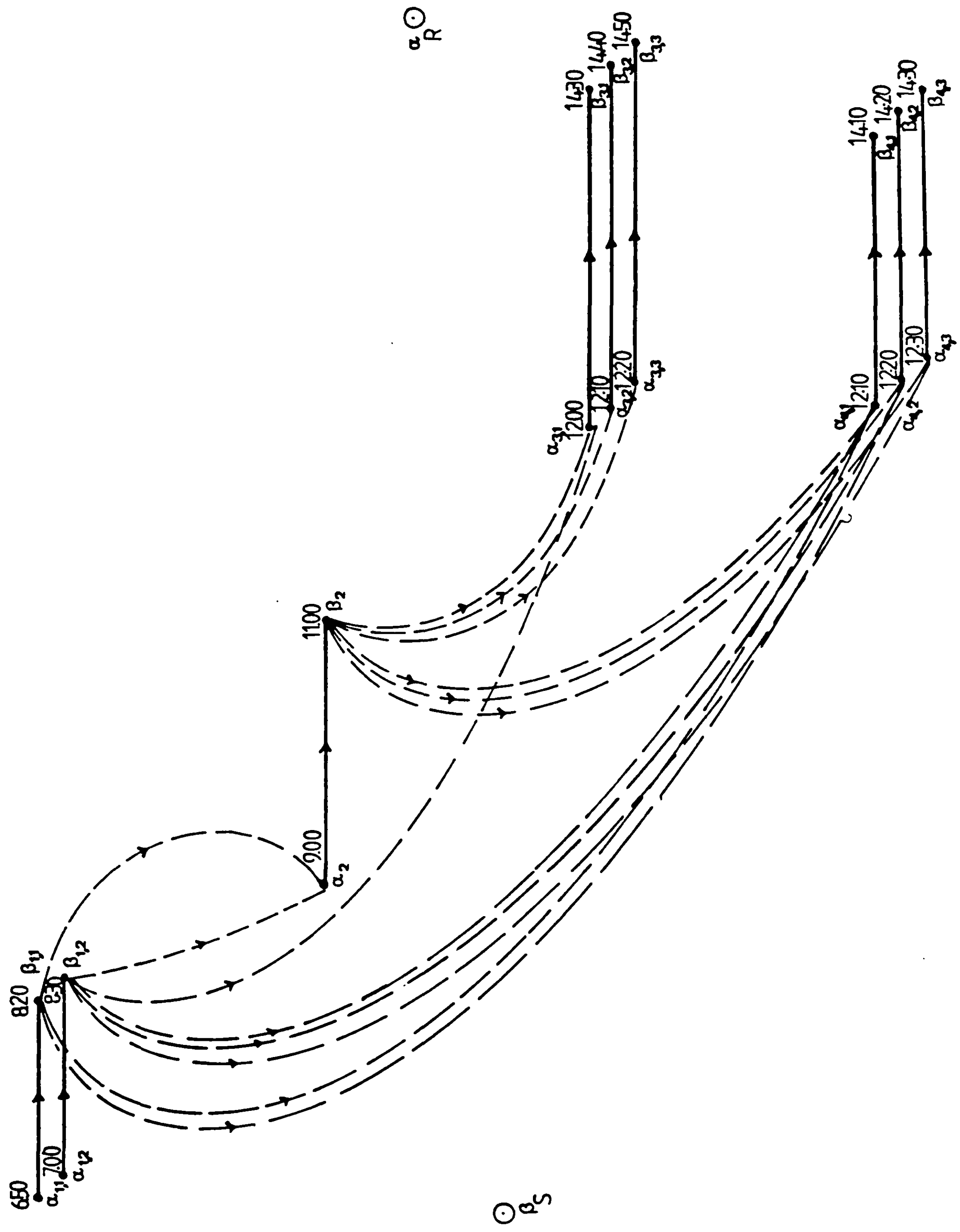


Figure 4-4 : Network G_t of the 4 Task-TCSP.



To overcome this problem we need to make the following assumption :"Instead of considering the tasks starting-times within continuous time-intervals we will consider them within discrete time-intervals. Thus , task 3 , for example , which is to start between 12.00 and 12.20 will be assumed to start either at 12.00 or 12.10 or 12.20 . Similarly task1 (resp. 4) is assumed to start either at 6.50 or 7.00 (resp. 12.10 or 12.20 or 12.30).

We express this fact by representing task 1 (resp. 3 and 4) by 2 (resp. 3) task-arcs as shown in network G_t of figure 4.4 . Like network G representative of a CSP , network G_t consists of :

(a) **task-arcs** : to each task i of the TCSP corresponds a set of task-arcs ;

(b) **linking-arcs** : these are the arcs that link the task-arcs . The conditions for joining two task-arcs i and j (or more precisely to join the final extremity β_i of task-arc i with the initial extremity α_j of task-arc j) are exactly the same as for G .Also since in G_t a task is represented by a set of task-arcs we have to consider in G the extra condition that 2 task-arcs correponding to the same task cannot be linked ;

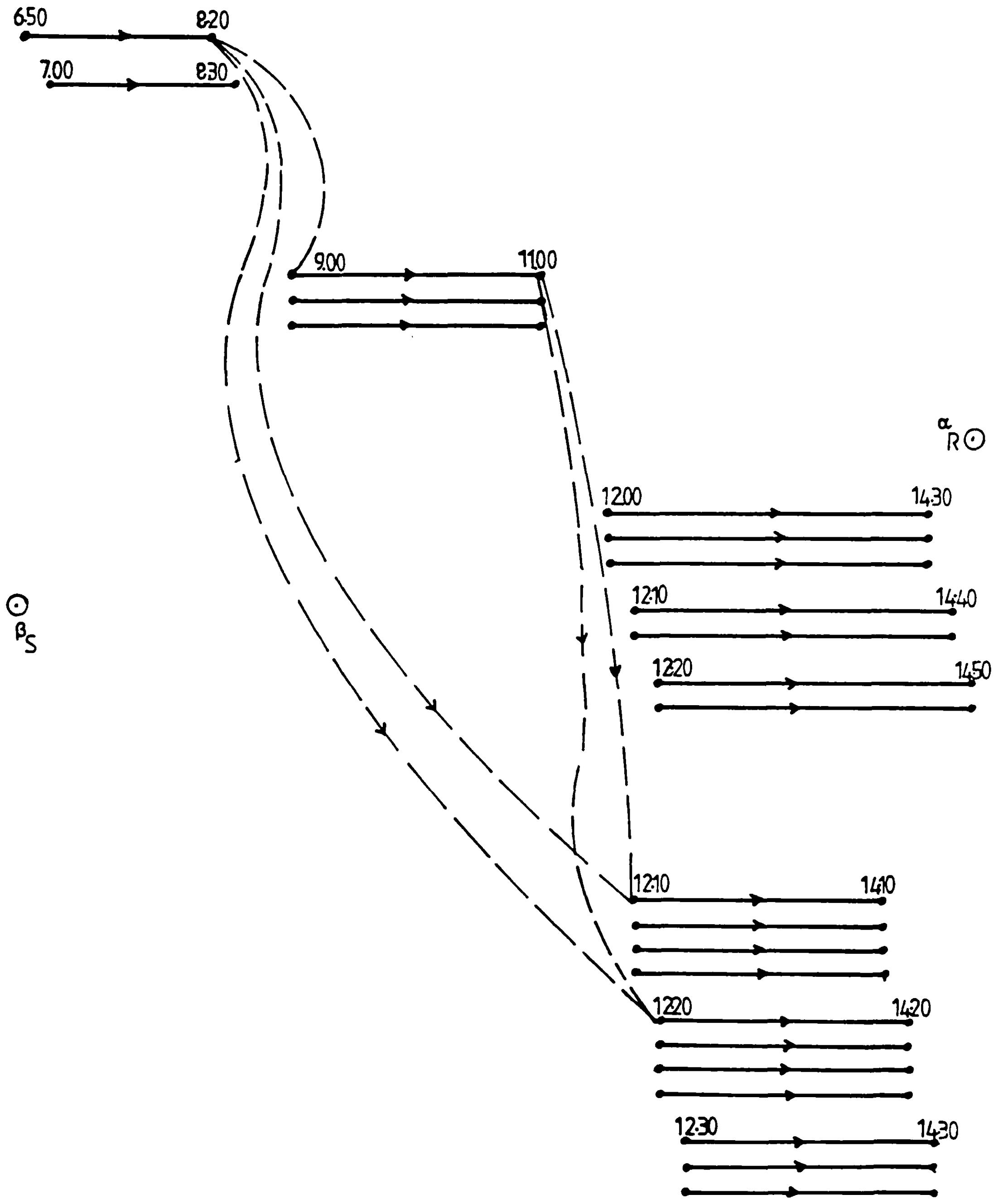
(c) **source-arcs and sink-arcs** : like G, a super-source β_S and super-sink α_R is linked with the starting or initial nodes α_i's of the task-arcs and all the finishing nodes β_j's of the task-arcs are linked to α_R.

It is worthwile noting that although we know which task-arc correspond to wich task , we assume when forming the source-arcs and sink-arcs that each task-arc of G_t corresponds to a different task . Consequently , we can consider network G_t of figure 4-4 as the network representative of a 9 task-CSP with a work duty period of 7 hours 30 minutes .Hence algorithm A8 can be applied to G_t . The resulting network G_t is represented in figure 4-5 . For the purposes of clarity , only the linking-arcs corresponding to paths starting from the first task-arc have been represented . The only modification that should be added to Algorithm A8, before applying it to graph G_t , is that 2 task-arcs corresponding to the same task of TCSP cannot be linked .

Finally , for each set of task-arcs of G_t , corresponding to a same task of

TCSP, one and only one task-arc is picked up in the optimal solution .

Figure 4-5 : Network $\bar{G}_t$



2-3 Algorithm A11.

This section deals with the procedure of converting a network G_0 representative of a TCSP into a network G_t which has the same properties as network G representative of a CSP .In the TCSP each task is defined by a continuous time-interval in which it must start (this time -interval will be refered to as starting-time-interval) and by the duration of the task . Consequently , network G_0 will consist only of task-arcs ; the linking-arcs being impossible to represent as shown in the previous example .

2-3.1 The Algorithm .

The algorithm is based on the following assumption : " the starting time interval (τ_i^1, τ_i^2) of each task i of the TCSP is assumed to be discrete ie task i can start every M minutes after τ_i.

Step 0 : Consider the starting time intervals of all the tasks and set the step-interval M ; Set i = 0 and GOTO STEP 1 ;

Step 1 : i = i + 1 ;Consider task i with interval $[\tau_i^1, \tau_i^2]$. Represent task i by task-arcs starting respectively at τ_i^1,τ_i^1 +M,τ_i^1 +2M,...etc... and of duration that of task i ; GOTO STEP 2 ;

Step 2 : If all tasks i's have been scanned GOTO STEP 3; Else GOTO STEP 1 ;

Step 3 : Link each task-arc of G_t with the source (resp.sink) to form the source- (resp. sink-) arcs ;GOTO STEP 4 ;

Step 4 : Link every pair of task-arcs i and j of G_t (i precedes j) if all following conditions are satisfied :

(a) starting time of task j is greater than finishing time of task i ;

(b) the difference between the finishing time of task j and the starting time of task i is less than or equal to the work duty period T_0;

(c) task-arcs i and j do not correspond to the same task .

2-3.2 The Size of G_t .

Depending on the size of the starting time intervals we can choose the step-interval in such a way that each task of the TCSP is represented by at most p task-arcs . Hence if n is the number of tasks of the TCSP , G_t will correspond to at most a pn-task CSP.

2-4 The Problem Formulation .

After applying algorithm A8 to network G_t (obtained in section 2-3 as output of algorithm A11) we obtain a network $\bar{G}_t$ (similar to network $\bar{G}$ of section 3,chapter3) in which all paths have duration less than or equal to the work duty period T .

The formulation of the TCSP based on $\bar{G}_t$ is exactly the same as that of the CSP which is based on G (see section 4 , chapter 3) .

3- The Multiple Vehicle Type Crew Scheduling Problem (VCSP) .

3-1 Introduction .

The multiple vehicle type crew scheduling problem (VCSP) is defined as :

" minimize the cost of covering a certain number (say, n) of tasks by a given number (say , K) of crews in such a way that :

(a) the crew and the corresponding vehicle form a single resource package ;

(b) each task must be covered once only by a single crew ;

(c) the work duty period T_0 is fixed and is the same for all crews ;

(e) the K vehicles are of L $(1 \leq L \leq K)$ types and the set of vehicle types that can cover a specific task is given for all tasks ."

If in condition (e) the number M of types of vehicle is equal to 1 then the VCSP becomes a CSP . Also if there is no task that can be covered by more than one type of vehicle the VCSP will consist of M different CSP's .If we assume that each type of vehicle is housed at a different depot the VCSP will be equivalent to the multiple depot CSP (section 7-2, chapter 3) with the extra condition that to each depot there is a certain number of tasks that can be covered by the vehicles housed at this depot . Consequently a network G_V representative of a VCSP which is similar to the network G_D representative of a multiple depot CSP can be constructed .

An example explaining the construction of G_V is given in section 3.2. Section 3.3 deals with the general procedure of forming G_V . Subsequently the problem is formulated as a minimum cost network flow problem plus additional set partitioning type constraints .

3-2 Example of Constructing G_V.

Consider the 5 task - 3 vehicle VCSP represented in table 4-4 in which 2 vehicles are assumed to be of the same type and the last vehicle of another type. The work duty period is assumed to be 600 minutes .Column 4 indicates the type of vehicle that can cover the corresponding task . Thus task-arc (α_1,β_1) and (α_4,β_4)

(resp. (α_2,β_2) and (α_5,β_5)) can be covered only by the 1st (resp. 2nd) vehicle type.Whereas task-arc (α_3,β_3) can be covered by either type . The graph G representative of the corresponding 5 task - CSP is depicted in figure 4-6 .

Table 4-4 : A 5 Task- VCSP .

Task	Starting time (minutes)	Duration time (minutes)	Vehicle Type
1	50	100	1
2	200	150	2
3	400	100	1;2
4	550	200	1
5	550	100	2

Now assume that the vehicles of the 1st (resp. 2nd) type are housed at depot 1 (resp. 2) . The VCSP becomes a multiple depot CSP . Consequently , the same procedure, described in section7-2 (chapter3), for constructing the graph G_D representative of a multiple depot CSP can be applied here . The resulting network G_V representative of a VCSP is represented in figure 4-7 . It is similar to G_D except that :

(a) In each subnetwork k (k=1,2) of G_V , only the task-arcs that can be covered by vehicle's type k have been considered . Thus in the first subnetwork corresponding to type 1, task-arcs (α_2^1,β_2^1) and (α_5^1,β_5^1) have been omitted since they cannot be covered by this type of vehicle . For the same reason task-arcs (α_1^2,β_1^2) and (α_4^2,β_4^2) are not represented in the second subnetwork corresponding to the vehicles of type 2 ;

(b) the flow in (β_s,D_1^1) and (D_2^1,α_R) is equal to the number of vehicles of the first type ie 2 . Similarly the flow in (β_s,D_1^2) and (D_2^2,α_R) is equal to 1 ;

(c) there is no cost attached to the arcs departing from D_1^1 and D_1^2 or arriving at D_2^1 and D_2^2 .

Figure 4-6 : Network G.

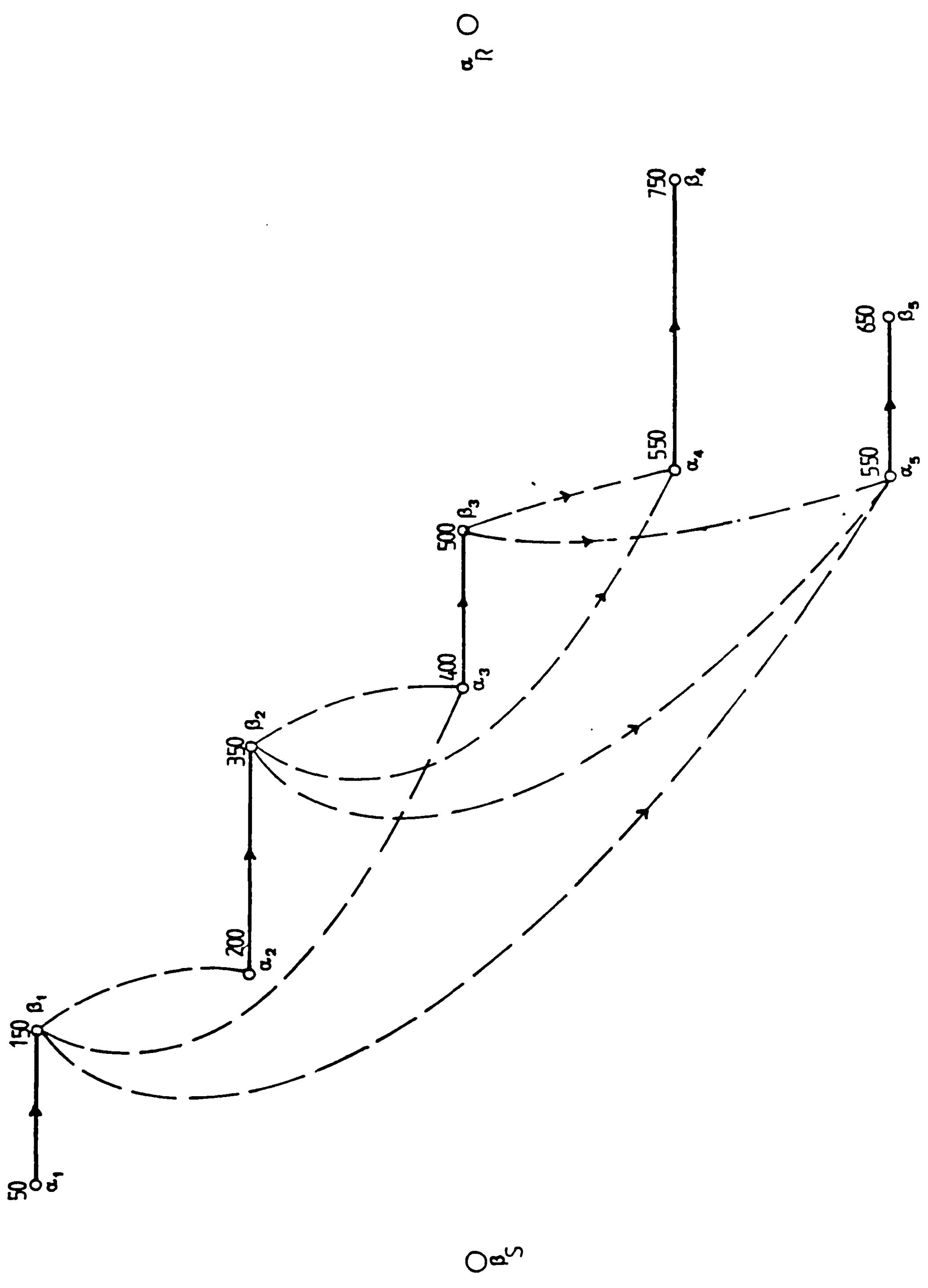


Figure 4-7 : Network G_V.

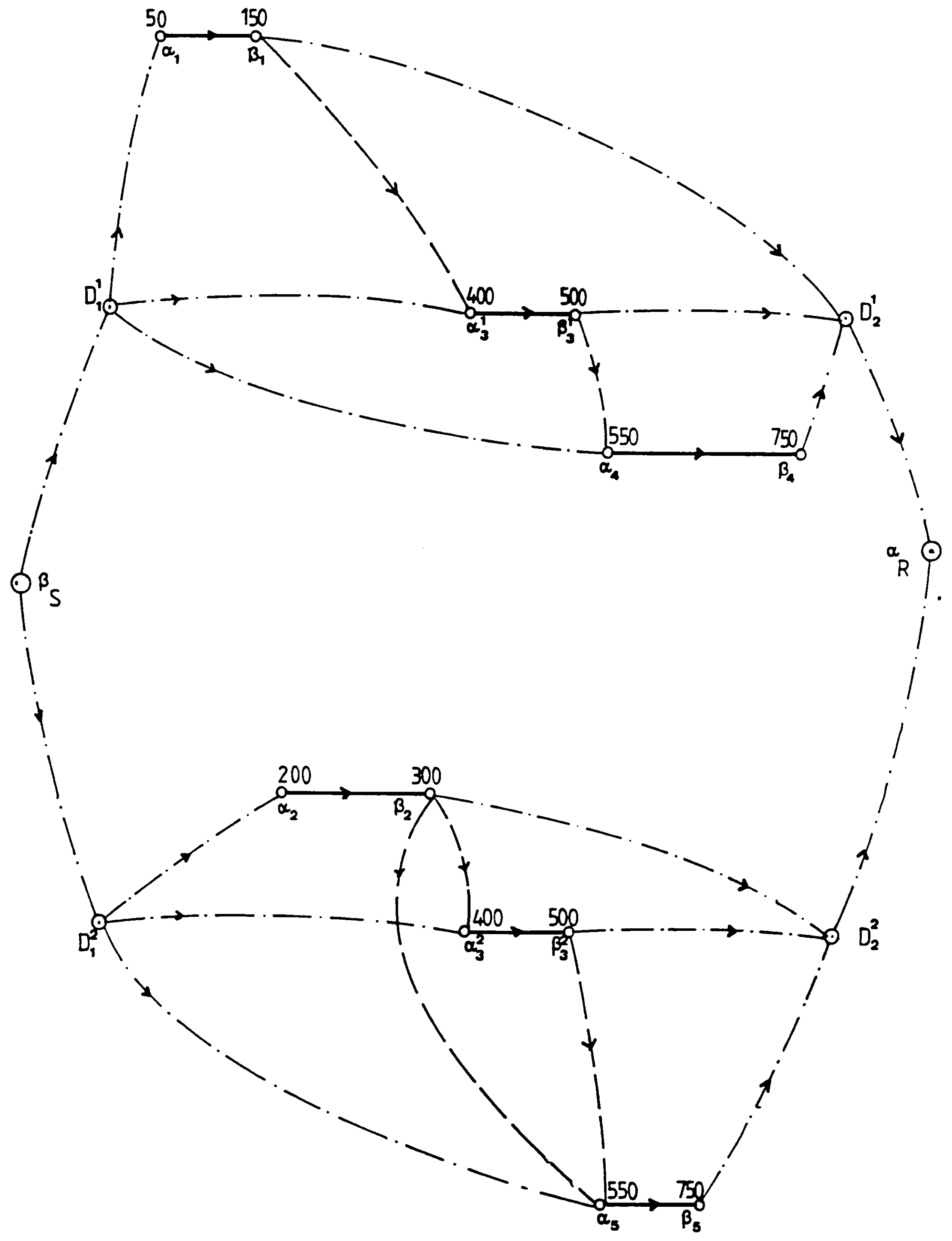
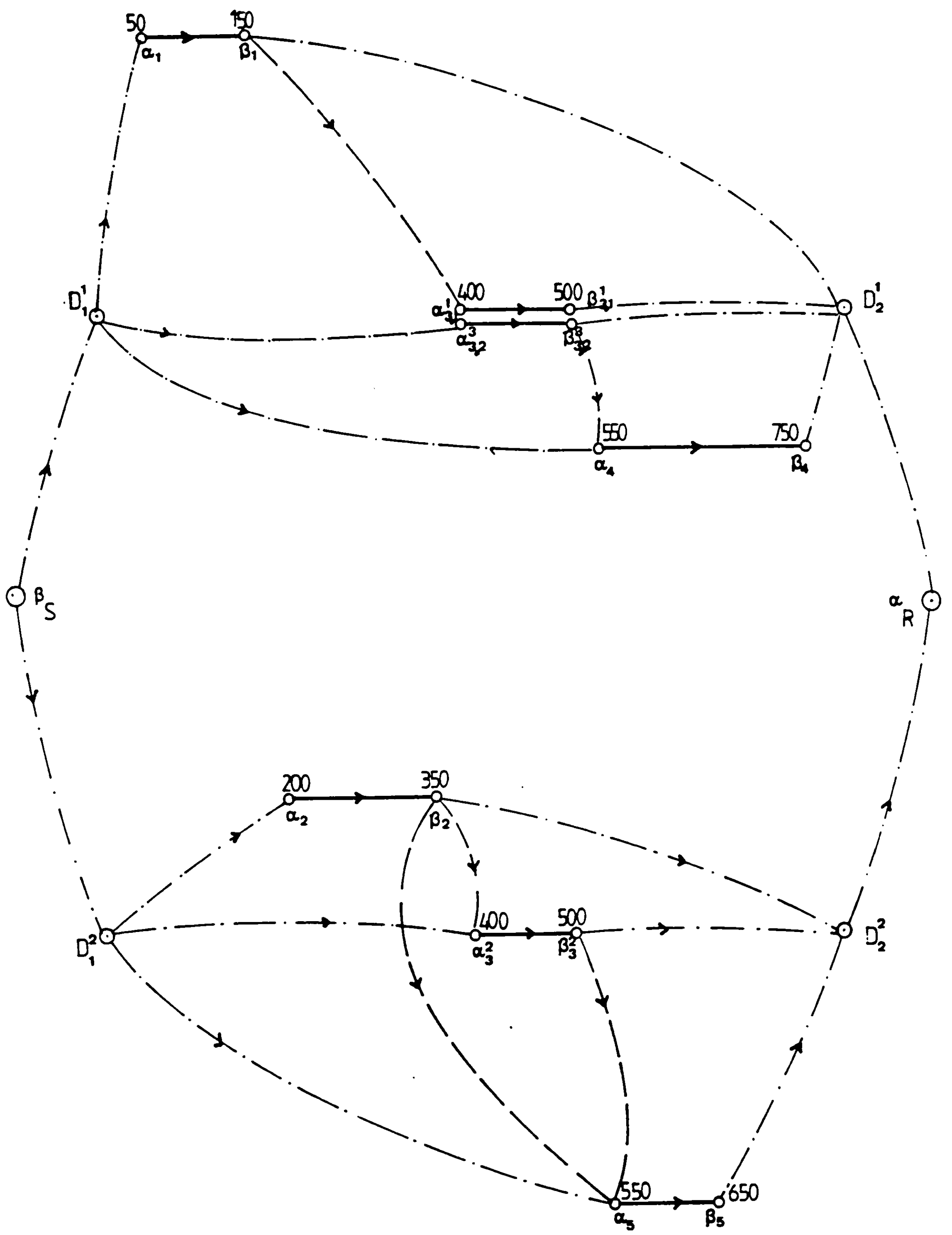


Figure 4-8 : Network $\bar{G}_V$.



3-3 Construction of Network G_V representative of a VCSP.

If we assume that each type of vehicle is housed at a different depot , network G_V becomes exactly the same as network G_D representative of a multiple depot CSP (see section 7,chapter 3) with the exception that :

(i) All source-arcs and sink-arcs of G_v have zero cost ;

(ii) In each subnetwork G_k (k=1,...,L) (where L is the number of vehicle's types) the task-arcs that cannot be covered by type k of vehicle are omitted .

Consequently , from this point onwards the VCSP can be considered as a multiple depot CSP and both the formulation and the solution technique presented in section 7 of chapter 3 can be applied to solve the VCSP .

Finally , it is worthwhile to mention that after applying algorithm A8 to G_v we obtain the expanded graph $\bar{G}_v$ of figure 4-8 in which no path has length greater than the work duty period .

4- Solving the GCSP with GET .

4-1 The Algorithm A12 .

Now that the graph expansion technique (GET) has proved to be quite effective for both the VCSP and the TCSP (in the case where the time-windows are relatively small) it becomes worthwhile applying the technique to the GCSP . In this section an algorithm A12 is described which expands the graph G_T representative of a GCSP into a graph $\bar{G}_T$ with the following properties :

(i) All paths of $\bar{G}_T$ have length equal to or smaller than the work duty period.

(ii) All the tasks in any one path of G_T can be covered by at least one common vehicle's type ;

(iii) The starting time of each task in any one path will be within the corresponding allowed time-window .

Algorithm A12 is a 2-phase algorithm which consists of a combination of three algorithms . Like the TCSP it is not possible to directly represent the GCSP by an acyclic graph . So the first phase of the algorithm consists of constructing the acyclic graph G_T representative of the GCSP . For this , we need to apply sequentially algorithm A11 and algorithm A10 (the VSCP version) . In the second phase the graph's expansion algorithm A8 is applied to G_T to obtain $\bar{G}_T$.

Algorithm A12 proceeds as follows :

PHASE 1 :

Step 1 : By dropping the multiple vehicle constraint (ie by assuming we have only one single vehicle type) we obtain a TCSP . Apply algorithm A11 to construct the graph G_t representative of a TCSP ;

Step 2 : If , having G_t as input , we take into consideration the multiple vehicle type constraint the problem becomes a VCSP . Apply algorithm A10 with graph G_t as input to construc the graph G_T representative of both the TCSP and the VCSP;

PHASE 2

Step 3 : Apply the graph's expansion algorithm A8 to G_T. The resulting expanded graph will be refered to as $\bar{G}_T$.

4-2 Example .

To illustrate the application of Algorithm A14 consider the GCSP defined by the input data of table 4-5 . We assume 2 vehicles of different type are available to cover all the 4 tasks. Also the work duty period is assumed to be 7 hours 30 minutes .

The application of step 1 of Algorithm A12 will result in the network of figure 4-4. Applying step 2 to G_t we obtain network G_T , of figure 4-9 , representative of a GCSP. Finally, at the end of Algorithm A12 we get the expanded network G_T of figure 4-10 .

Table 4-5 : A 4 Task-GCSP .

Task	Window Starting time (hr & mn)	Duration time (minutes)	Type of vehicle
1	6.50-7.00	90	1
2	9.00	120	1;2
3	12.00-12.20	150	2
4	12.10-12.30	120	1

When applying algorithm A11 (ie step 1 of algorithm A12) the smallest time unit was taken to be 10 minutes . For the sake of clarity not all the linking-arcs of graph $\bar{G}_T$ of figure 4-10 have been represented .The ones represented are indicated by dotted lines in the figure .

4-3 Problem Formulation .

Like the CSP and the TCSP , the GCSP can be formulated as a minimum cost network flow problem plus additional partitioning type constraints . The formulation which is based on network $\bar{G}_T$ is exactly the same as that of the TCSP (see section 2-4) with the exception that $\bar{G}_t$ is replaced by $\bar{G}_T$.

Relaxing the integrality constraints , we have been able to solve (ie find the integer programming solution) all the 100 randomly generated GCSP's using just an LP package . The results are presented in section 6.

Figure 4-9 : Network G_T.

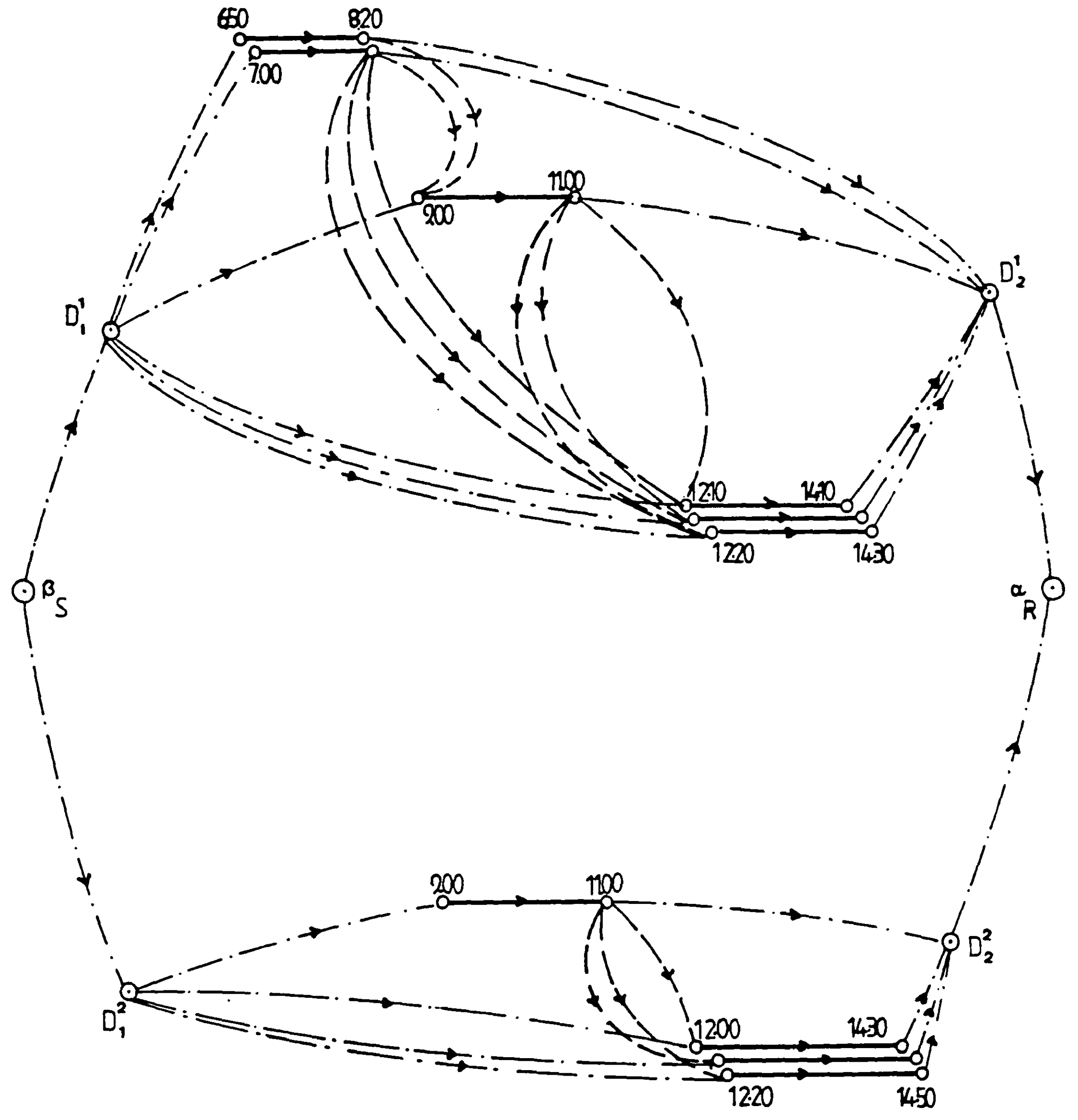
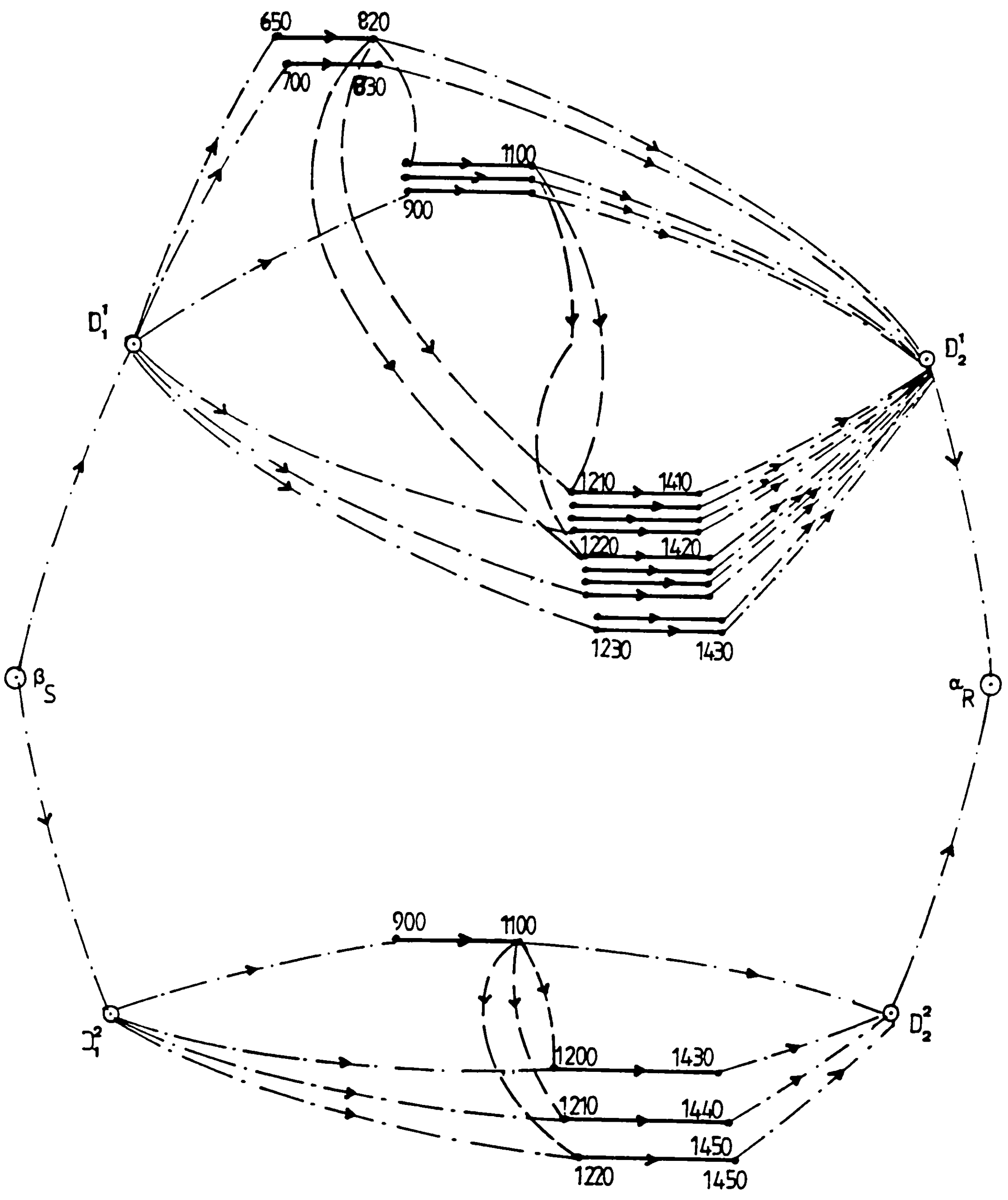


Figure 4-10 : Network $\bar{G}_T$.



5- A Modified Version of the Graph Expansion Technique .

5-1 Introduction .

As the average task time-window increases the size of the largest GCSP that can be solved decreases . Thus , if we consider an average time-window of 10 hours and a step size of 15 minutes the largest size GCSP that can be handled will not have more than 10 tasks . This is due on the one hand to the enormously large size of the expanded graph $\bar{G}_T$ on which the problem formulation is based and on the other hand to the limitations imposed by the existing LP packages .

In this section we present a modified version of the graph expansion technique which produces an expanded network $\bar{G}_T'$ smaller than $\bar{G}_T$ as the average time window increases . Actually the importance of this technique becomes more apparent when considering the GCSP with rest-periods as the network representative obtained with this technique is much smaller than the one obtained with the previous technique .

Algorithm A13 , which expands the network G_T representative of a GCSP into another network $\bar{G}_T'$ is described in section 5-2. It is then illustrated by the example of section 5-3.Like all the CSP's considered so far the problem was first formulated as a minimum cost network flow problem plus additional set partitioning type constraints (the formulation is exactly the same as the one presented in section 4-3 except that the graph on which it is based is $\bar{G}_T'$) , then it was solved using just an LP package . The corresponding results are presented in section 6 .

5-2 Algorithm A13 .

This algorithm consists of expanding network G_T representative of a GCSP into another network $\bar{G}_T'$ which has the same properties as network $\bar{G}_T$ but which has the additional advantage of a smaller size as the average time-window gets larger .

Before we explain the main idea of the algorithm we will introduce some useful definitions and remarks that will facilitate the description of the algorithm .

Given V types of vehicles , network G_T consists of V independent subnetworks $N_1,\dots,N_V$. By "independent" we mean that 2 nodes belonging to different subnetworks need never be linked . In this new expansion each subnetwork will be considered separately and independently of the others ie each subnetwork N_i ($i=1,\dots,V$) will be expanded into another subnetwork $\bar{N}_i$ ($i=1,\dots,V$) in which all paths have length less than or equal to the work duty period .

The size of $\bar{G}'_T$ in terms of nodes (of the network) can be reduced by half if instead of representing the tasks of GCSP by task-arcs we represent them by task-nodes .

Given a task i and its starting time-window $[ST_i^1,ST_i^2]$ we saw in algorithm A12 (section 4-1) that this time-window was divided into smaller discrete intervals $[ST_i^1,ST_i^1+M,ST_i^1+2M,\dots,ST_i^2=ST_i^1+KM]$ where M is the step size , K is equal to $(ST_i^2-ST_i^1)/M$ and $ST_i^1+kM(k=0,1,\dots,K)$ is a possible starting time for task i. In graph theoretic terms we represent this fact by assuming that K task-nodes $n_0^i,n_1^i,\dots,n_K^i$ in G_T represent task i . If task-node n_j^i is picked up in the optimal solution this will mean that task i is to start at time ST_i^1+jM . In what follows to each task-node n of subnetwork N_i we will associate 2 numbers : a node starting time p_n and a node finishing time q_n . These are respectively the times at which the corresponding task would start and finish if node n was picked up in the optimal solution .

5-2.1 Main Idea of the Algorithm .

The main idea of the algorithm can be explained as follows : assume that the work duty period is equal to 6 hours and that the schedule is performed over a planning period of 24 hours . Also let M be the step size . For the sake of simplicity we let M be one hour . A path P in subnetwork N_i must have a length of at most 6 hours . This means that if for example it starts at 10.00am it must finish before 4.00pm . Within this time-period the only task-nodes of N_i that can be covered by path P are those nodes n for which :

$$p_n \geq 10.00\text{am} \qquad \text{and} \qquad q_n \leq 4.00\text{pm} \qquad (4\text{-}1)$$

All other nodes have no chance of belonging to P .

Let E^i_{10} be the subnetwork of N_i associated with the task-nodes of N_i that satisfy (4-1) . Now if we let P starting at time τ (τ=0,1,2,...,24hours) , E^i_τ can be defined as follows :

$$E^i_\tau = (X^i_\tau, U^i_\tau) \qquad (4\text{-}2)$$

where

$$X^i_\tau = \{ \text{ task-nodes of } N_i \,/\, p_n \geq \tau \text{ and } q_n \leq \tau+6 \ \} \qquad (4\text{-}3)$$

and

$$U^i_\tau = \text{set of all linking-arcs of } N_i \text{ associated with } X^i_\tau. \qquad (4\text{-}4)$$

Clearly , each feasible path P (by "feasible" we mean "of length at most 6hours") of N_i belongs to at least one E^i_τ (τ=0,1,...,24) . Also each task-node of N_i can be in several E_τ's.

Duplicating each task-node n of N_i as many times as it appears in the E^i_τ's we obtain network $\bar{G}'_T = N_1 \cup N_2 \cup ... \cup N_V$ where $N_i = E^i_0 \cup E^i_1 \cup ... \cup E^i_{24}$.

5-2.2 The Algorithm A13 .

As we mentioned above since network G_T representative of a GCSP consists of V (number of vehicle's types) "independent" subnetworks N_i's , in this graph expansion technique each N_i will be expanded separately and the more general network $\bar{G}'_T$ will consist of the union of all expanded subnetworks $\bar{N}_i$'s.

The following is an algorithm which describes how to expand subnetwork N_i of G_T into another subnetwork $\bar{N}_i$ in which all paths have length at most equal to the work duty period .

Step 0 : Set $\tau = 0$; GOTO STEP 1 ;

Step 1 : If no task-node n of N_i has a node starting time p_n equal to τM GOTO STEP 2 ; Else consider subnetwork $E^i_\tau = (X^i_\tau, U^i_\tau)$ where:

$$X^i_\tau = \{\text{task-nodes n of } N_i / p_n \geq \tau M \text{ and } q_n \leq \tau M + T_0\}$$

and $\qquad U^i_\tau$=set of all linking-arcs of N_T associated with X^i_τ.

GOTO STEP 2;

Step 2 : If $\tau M \geq 24\text{-}T_0$ (hours) GOTO STEP 3 ;Else set $\tau=\tau+1$ and GOTO STEP 1;

Step 3 : Duplicate each task-node n of N_i as many times as it appears in the E^i_τ's ;

$$N_i = E^i_1 \cup E^i_2 \cup \ldots \cup E^i_\tau$$

where $\qquad \tau$ = number of E_τ's generated ; GOTO STEP 4 ;

Step 4 : Reduction Process .

Remove from each E^i_τ ($\tau=1,\ldots,\tau$) all the "isolated" task-nodes.

(A task-node is isolated if it has no predecessor).

5-2.3 Remark .

(a) In step 2 of the algorithm the inequality condition

$$\tau M \geq 24\text{-}T_0 \text{ (hours)} \qquad (4\text{-}5)$$

assumes that both M and T_0 are given in hours . If , however , they are expressed in minutes then the condition becomes :

$$\tau M \geq 24\text{x}60 - T_0 \text{ (minutes)} ; \qquad (4\text{-}6)$$

(b) Knowing that the planning period is one day of 24 hours the logical question that one might ask is :" why have we not considered the E^i_τ's for which :

$$24 - T0 \leq \tau M \leq 24 \qquad (4\text{-}7)$$

The reason behind this is that all the E^i_τ's that satisfy (4-7) are in fact included in $E^i_{\tau 0}$ with $\tau_0 = (24\text{-}T_0)/M$. Hence there is no need to consider the subsequent E^i_τ's.

5-3 Example .

Consider the 3 task- GCSP , of table 4-6 , with a work duty period of 300 minutes and with M = 1 hour (60 minutes) . the network G_0 which consists only of the task-arcs is depicted in figure 4-11 . The dotted lines represent the starting time windows of the tasks .

Applying the first phase of algorithm A12 to the input data of the problem we obtain network G_T , representative of a GCSP , of figure 4-12 . For convenience , the task-arcs have been replaced by task-nodes and for the sake of clarity the source-arcs and sink-arcs associated with task-nodes corresponding to task 1 have not all been represented .

Figure 4-11 : Network G_0

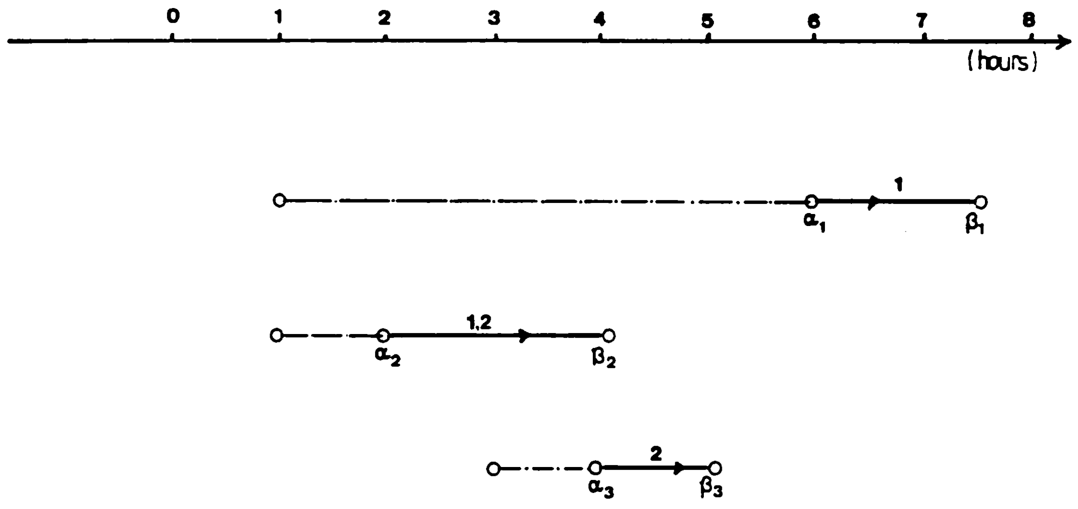


Table 4-6 : A 3 Task-GCSP .

Task	Window Starting time (minutes)	Duration time (minutes)	Type of vehicle
1	60-360	90	1
2	60-120	120	1;2
3	180-240	60	2

Figure 4-12 : Network G_T

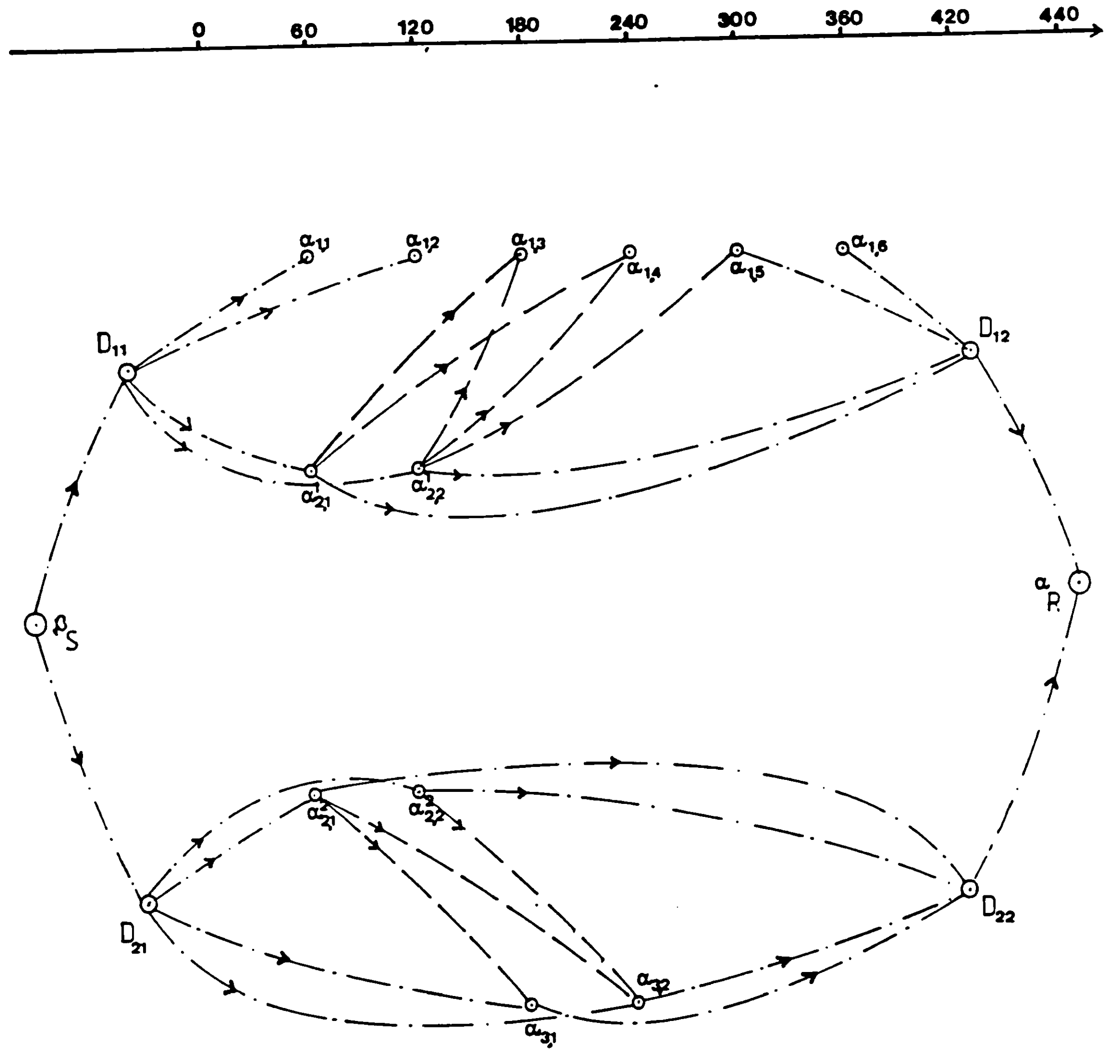


Figure 4-13 : Network $\tilde{G}_T'$

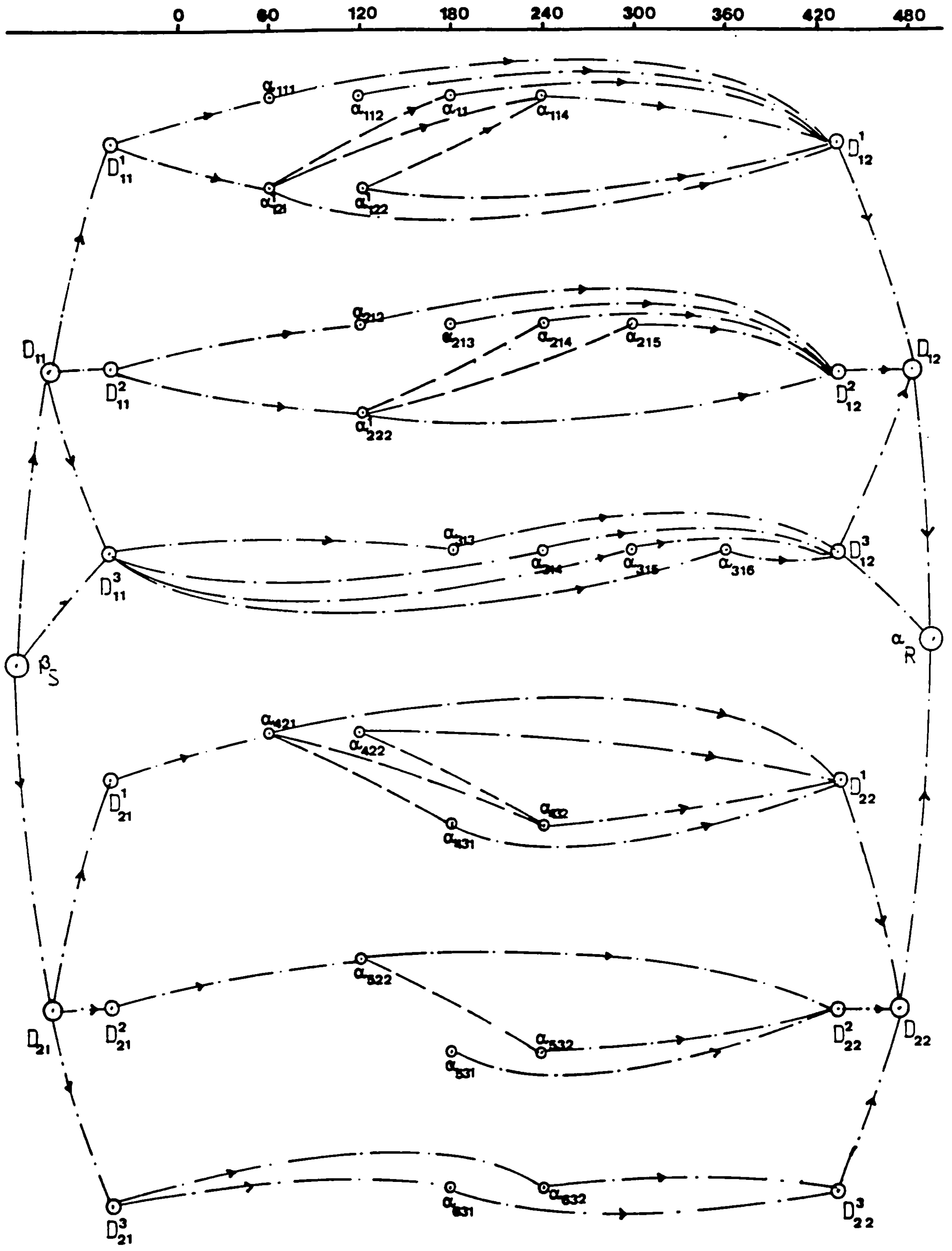
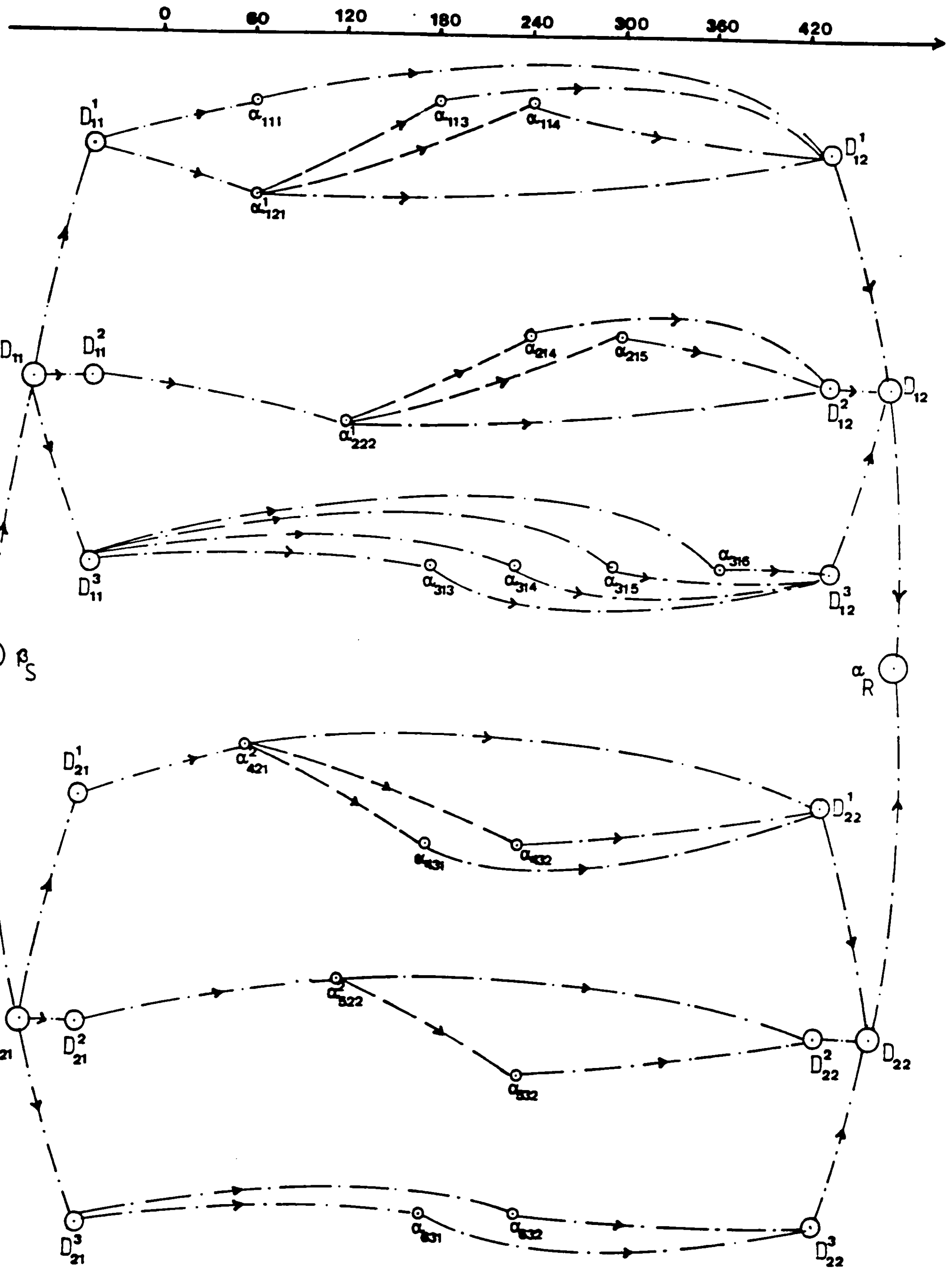


Figure 4-14 : Network $\bar{G}_T'$



Network $\tilde{G}_T'$ of figure 4-13 is the expanded network produced from G_T by applying the first 3 steps of algorithm A13. The reduced version (network $\bar{G}_T'$) shown in figure 4-14 has been obtained from $\tilde{G}_T'$ by removing all the task-nodes that have no predecessor.

6- Computational Results .

Out of a total of 190 problems , of size varying between 10 and 50 tasks , randomly generated for each one of TCSP,VCSP and GCSP , the optimal integer solution was found in all cases by using just a linear programming code [112] . The LP code was applied to the linear relaxation of each problem.

6-1 Efficiency of the Graph Expansion Technique when the Time-Windows are Small .

To test the algorithms presented in this chapter more than 500 problems of size ranging from 10 to 50 tasks were randomly generated in two groups . The results are summarized in table 4-7.

For each one of the TCSP , the VCSP and the GCSP a total of 140 problems were considered in the first group and 50 problems in the second group . Our aim in considering the problems of the second group , which were all generated from the same 25 task-problem by varying the cost coefficients , was to try to generate a problem for which the solution of the linear relaxation of the problem was not integer. The reason for adopting this approach has already been dealt with in section 8 (chapter3). As it can be seen from table 4-7 we have been unsuccessful in achieving this aim . Thus , all the 150 problems generated in this group (50 for each one of TCSP,VCSP and GCSP) have been solved to optimality by a linear programming package.

For the problems of the first group , the work duty period was assumed , in all cases , to be equal to 6 hours . The number of crews was taken to be the minimum number K^* feasible for each problem . The duration of each task was assumed to vary within the range [45minutes,2hours30minutes].Also the cost coefficients c_{ij} associated with linking-arc (β_i,α_j) has been generated according to the formula :

$$c_{ij} = d_{ij}$$

where d_{ij} = duration of linking-arc (β_i,α_j)

= starting time of task j - finishing time of task i

For the VCSP the starting time of each task was uniformly generated in the range [00.00-24.00hours] and for the TCSP only one vehicle type was assumed .

The remaining parameters were generated as follows :

(i) Generating the Starting Time Windows .

For both the TCSP and GCSP the step size was taken to be M = 15 minutes and the length of each time window was uniformly generated in the range [0-60minutes].The beginning of the time window was made to vary in the range [0,24hours].

(ii) Generating the Vehicle Types .

Two and three types of vehicle were assumed for VCSP's of various sizes.For problems with 2 vehicle types the approach we took in deciding which type of vehicle will be assigned to which task was purely random and based on the following rule :
"For each task i , i=1,...,n generate a random number α_i in the range [0,1].

(a) If $0.00< \alpha_i \leq 0.33$ then task i can be covered by vehicle type 1 only ;

(b) If $0.33< \alpha_i \leq 0.66$ then task i can be covered by vehicle type 2 only ;

(c) If $0.66< \alpha_i \leq 1$ then task i can be covered by both types of vehicle . "

For problems with three types of vehicle the same idea was used to decide for each task which type of vehicle will be assigned to it .

(iii) Comparison of the 2 Expanded Networks $\bar{G}_T$ and $\bar{G}_T'$.

Out of the 190 randomly generated TCSP's it has been noticed that in 60% of the cases the size of network $\bar{G}_T$ was slightly smaller than that of network $\bar{G}_T'$. Network $\bar{G}_T$ is the expanded version of network G_T (representative of a GCSP) obtained from algorithm A12 which is based on the expansion technique presented in the previous chapters , whereas network $\bar{G}_T'$ is the homologous of G_T obtained with the expansion technique of the previous section .

Subsequently , the time window of each task was allowed to vary uniformly in the range [00.00-24.00hours] and 100 randomly generated GCSP's were generated to compare the size of $\bar{G}_T$ and $\bar{G}_T'$. In 73 problems the size of $\bar{G}_T'$ was smaller than that of $\bar{G}_T$. For the rest , network $\bar{G}_T$ had a smaller size in 12 problems and in 15 problems both networks had the same size .

6-2 Inefficiency of the GET for GCSP's with Large Time Windows.

As we mentioned previously the results of table 4-7 were obtained by assuming that the time-window of each task varies uniformly in the range [00-60minutes] and that the step size was supposed to be 15 minutes .Clearly , this means that each task of the GCSP will be represented in G_T by at most 4 task-nodes (in each subnetwork N_T of G_T).

Let M be the step size and D be the average length of a task time window . If D increases and M remains 15 minutes then the largest size of GCSP that we will be able to solve with existing LP packages will decrease . The only example we have considered to illustrate this claim was by letting D=10 hours and M=15 minutes . The largest size GCSP we could solve with the GET involved not more than 10 tasks and 2 types of vehicle . This was due to the limitations imposed by the LP package which

Table 4-7 : Average Computational Results for the TCSP , VCSP and GCSP .

Problem	Number of Tasks	Number of Problems	Size of the LP Problem [rowsxcolumns]			Size of Expanded Graph [Task-Arcs]			Time+ Expansion Time			Time+ LP Solution Time		
			TCSP*	VCSP**	GCSP***	TCSP	VCSP	GCSP	TCSP	VCSP	GCSP	TCSP	VCSP	GCSP
Group 1	10-14	20	78x 320	45x 159	123x 539	63	30	108	0.6	0.2	0.9	2.1	0.9	5.5
	15-19	20	111x 461	71x 266	147x 637	91	51	127	0.8	0.4	0.9	4.6	1.9	7.7
	20-24	20	172x 739	88x 330	224x 993	147	63	199	0.9	0.6	2.1	8.9	2.3	18.0
	25-29	20	230x1003	129x 515	480x2222	200	100	450	2.3	0.9	8.9	17.9	5.8	75.8
	30	20	288x1278	149x 600	526x2438	256	117	494	3.2	1.0	10.1	19.6	8.2	148.3
	40	20	437x1966	204x 829	694x3219	395	162	652	7.2	1.6	19.7	62.1	11.1	169.5
	50	20	507x2268	274x1132	935x4355	455	222	883	9.3	2.4	28.5	91.3	20.1	NC
Group 2	25	50	216x 946	118x 468	432x1999	189	91	405	1.8	0.8	7.7	16.2	4.8	53.1

* TCSP = Time-window CSP ;
** VCSP = Multiple vehicle types CSP ;
*** GCSP = General CSP ;
+ Seconds of CYBER 855 (Fortran Compiler) .

could not handle the corresponding linear problem of 850 rows and 3500 columns .

Now if we increase M , as D increases , in such a way that D/M $\leq$ 4 then whatever value of D we consider problems of the same size as those of table 4-7 could be handled with GET . However , although the problems will certainly be solved with just an LP package , the solution obtained although integer might be just a heuristic one .

This problem of solving large GCSP's with large task time-wimdows will be dealt with in the next chapter .

CHAPTER 5

A TREE SEARCH PROCEDURE FOR THE GCSP

1- INTRODUCTION.

We consider a tree search procedure that has proved to be quite efficient for reasonably large size GCSP's involving up to 50 tasks and with a task's time window allowed to vary in the range [00.00-24.00 hours] . This type of problems was impossible to solve with the graph's expansion technique considered in the previous chapter . This was due to the enormously large size of the expanded network which produced an LP problem that could not be handled by any code.

The success of any tree search procedure depends entirely on the quality of the lower bound and the upper bound to the problem . To obtain a good lower bound several different integer programming formulations were considered and were subsequently all relaxed in a Lagrangian fashion[71] . The different resulting lower bounds were compared and two formulations were picked-up as possible candidates to be embedded in the tree search procedure . All this is discussed in section 2.

The first formulation which is based on a shortest constrained path problem (SCPP) [56] was chosen because over all the problems tested it produced each time a better lower bound than all the other formulations . Because of the relatively long computing time required to get this bound a second candidate , a shortest path problem (SPP) based formulation , was also considered . The choice of the SPP formulation was motivated by two factors . As far as the quality of the lower bound is concerned it performed as well as all the remaining formulations (except the SCPP) and from the computing time point of view it was much better than all the others .

At this point it is worthwhile to mention that an algorithm that runs in polynomial time has been suggested , in section 2 , to solve the SCPP which is considered to be , in general , an NP-complete problem [66] . We are not saying that this algorithm can solve any SCPP but we can claim that we have been able to exhibit a particular case of the SCPP which can be solved in polynomial time .

As far as the upper bound to the problem is concerned two efficient heuristics are presented in section 4 . Actually this section deals with the whole tree search procedure used for our problem . Each step of the procedure is explained in detail .

We know that when the task's time window tends to be small the graph's expansion technique is the most suitable technique to solve the GCSP (see chapter 4) but when the task's time window becomes large it is preferable to use the tree search procedure. In section 6 we discuss the variations of the efficiency of the reduction test when the task average time window varies .

Finally the computational results are presented in two sections . The ones related to finding the lower bound are presented in section 3 and the computational results for the tree-search procedure , including those of the heuristics , are dealt with in section 5 .

2- SIX GCSP FORMULATIONS .

The aim of this section is to obtain a good lower bound that will be embedded in the tree search procedure . To achieve this six different integer programming formulations of the GCSP are considered . For each formulation the Lagrangian relaxation , one of the most successful integer programming techniques for obtaining lower bounds , has been applied . All problem formulations are based on a network G_T representative of the GCSP ($G_T=(V_T,A_T)$) except formulation 6 which is based on the expanded network $\bar{G}_T'$ obtained with algorithm A13 (section 5,chapter4) .

To remind the reader of the structure of network G_T we consider a 10 task-GCSP the data of which are given in table 5-1 and which is represented by network G_0 of figure 5-1 . The dotted lines represent the starting time windows of the tasks . Note that only the task-arcs can be represented (by the segments at the end of each time-window) . The task time window varies between half an hour (for task 7) to 20 hours (for task 1) . Three crews operating 3 vehicles of 2 types are assumed . The work duty period is 7 hours . Using algorithm A12 of section 4 (chapter 4) we obtain network G_T , of figure 5-2 , representative of the 10 task-GCSP . For convenience the task-arcs have been repesented by task nodes and for the sake of clarity only the linking-arcs departing from task node α_{11} have been represented .

Table 5-1 : A 10 Task-GCSP.

Task	Starting Time-Window [minutes]	Duration [minutes]	Vehicles Type
1	[60-1260]	135	2
2	[75- 165]	75	1
3	[75-1080]	90	1;2
4	[90- 660]	105	1;2
5	[105- 840]	60	2
6	[150- 885]	105	1
7	[165- 195]	75	1
8	[255- 735]	90	1
9	[330-1320]	60	1;2
10	[345- 765]	150	2

Figure 5-1 : Network G_0.

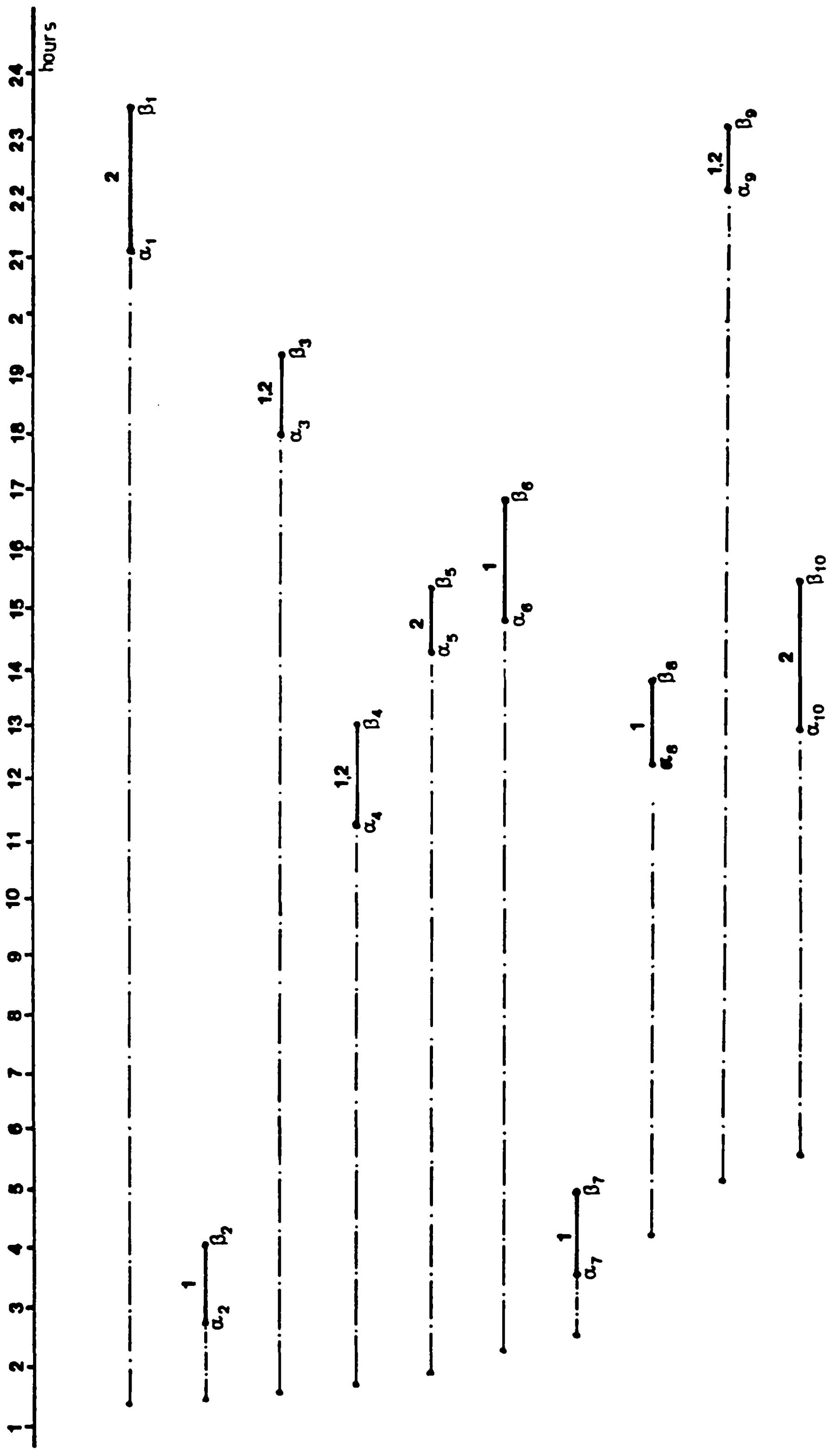
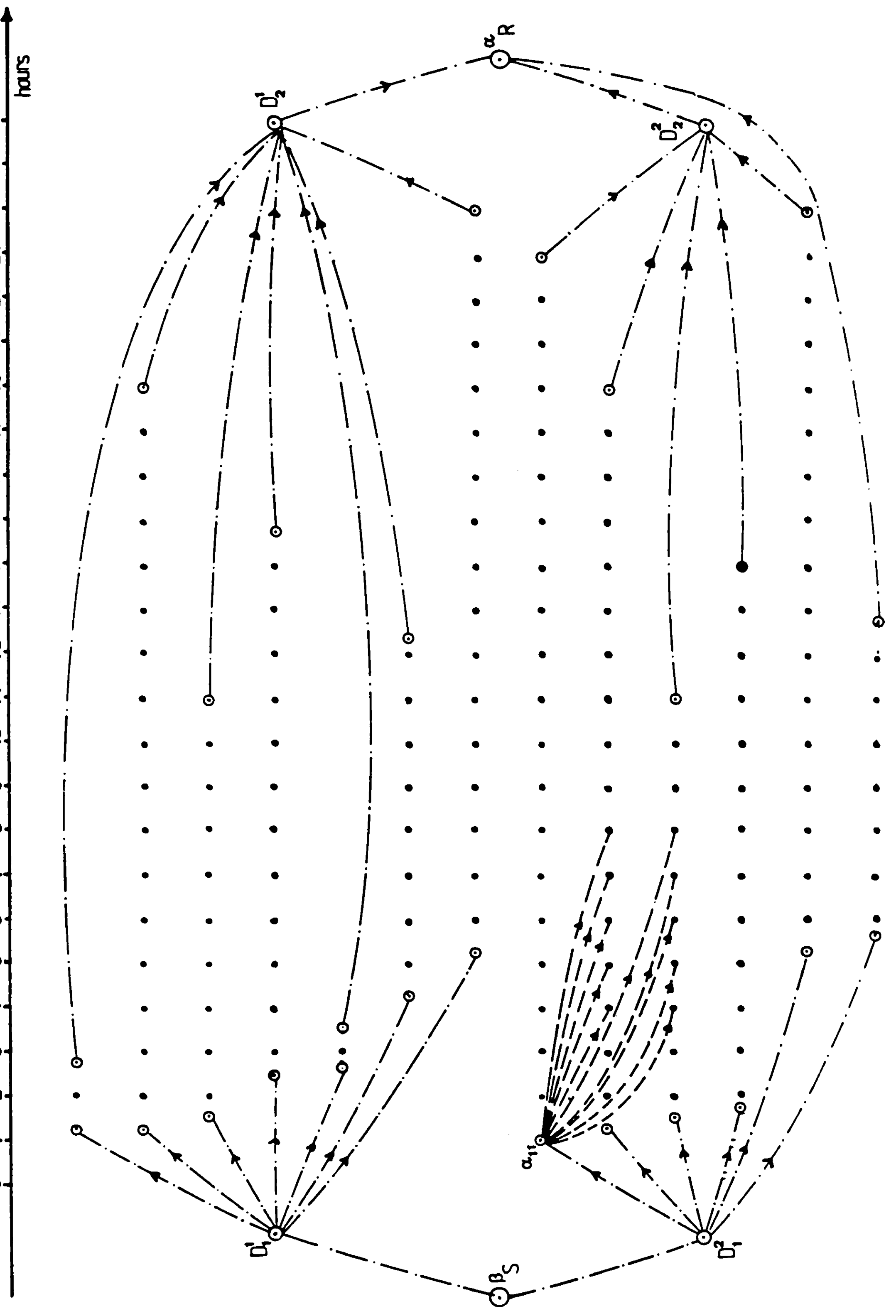


Figure 5-2 : Network G_T.



2-1 Network Flow Formulation 1.

If we let $x_{ijk} = 1$ if arc (i,j) εA_T is covered by crew k in the optimal solution ;

$= 0$ otherwise ;

then the problem becomes :

$$\text{Min} \sum_{k=1}^{K} \Big(\sum_{A_T} c_{ij} x_{ijk} \Big) \tag{5-1}$$

subject to :

$$\sum_{(i,j) \in A_T} t_{ij} x_{ijk} \leq T \qquad \text{for } k=1,\dots,K \tag{5-2}$$

$$\sum_{(i,j) \in F_p} \sum_{k=1}^{K} x_{ijk} = 1 \qquad \text{for } p=1,\dots,N \tag{5-3}$$

$$k=1,\dots,K \left\{ \begin{array}{lll} \displaystyle\sum_{j \in \Gamma(i)} x_{ijk} = \sum_{m \in \Gamma^{-1}(i)} x_{mik} & \text{for each node } i \in V_T & (5\text{-}4) \\ \displaystyle\sum_{i \in \Gamma(S)} x_{Sik} = \sum_{i \in \Gamma^{-1}(R)} x_{iRk} = 1 & S : \text{source} ; R : \text{sink} & (5\text{-}5) \end{array} \right.$$

$$x_{ijk} \in \{0,1\} \tag{5-6}$$

where : K = number of crews ;

N = number of tasks ;

T = work duty period .

F_p= set of all arcs of G_T whose initial node correspond to task p .

$\Gamma(i)$ (resp. $\Gamma^{-1}(i)$) = set of all immediate successors (resp. predecessors) of node i .

In the objective function , only the costs of the linking-arcs have been considered . The reason for not considering the costs of the other arcs has already been explained in chapter 2 (section 2-3) . Constraints (5-3) guarantee that each task must be covered once and only once by a single crew . This is accomplished by making sure that among all the task-nodes that represent task i (i=1,...,N) only one task-node will be picked-up in the optimal solution . Constraints (5-2) ensure that the length of each path of network G_T does not exceed the work duty period . In other words each crew is guaranteed that they will not be working more than the work duty period . The flow conservation at each node of G_T is expressed by constraints (5-4) and (5-5).

2-1.1 Two Problem Relaxations .

Two straightforward Lagrangian relaxations can be derived from the above formulations as follows :

Relaxation 1 :

Let us assume $\lambda_k \geqslant 0$ (k=1,...,K) and υ_p (p=1,...,N) are two Lagrange multipliers attached to constraints (5-2) and (5-3) respectively. Relaxing constraints (5-2) and (5-3) in a Lagrangian fashion , we obtain :

$$\text{Min} \sum_{(i,j)\in A_T} \sum_{k=1}^{K} (c_{ij} + \lambda_k t_{ij} + \nu_p) x_{ijk} - T\sum_{k=1}^{K} \lambda_k - \sum_{p=1}^{N} \nu_p \qquad (5\text{-}7)$$

subject to : constraints (5-4) and (5-5)

The relaxed problem being a minimum cost network flow problem , it can efficiently be solved by any network flow algorithm [63].

Relaxation 2 :

Let us assume $\lambda_k \geqslant 0$ (k=1,...,K) are the Lagrange multipliers attached to constraints(5-2) and δ_{ik} (k=1,...,K; i ε V_T) are the Lagrange multipliers attached to constraints (5-4) and (5-5) . Relaxing constraints (5-2),(5-4) and (5-5) in a Lagrangian way ,we get :

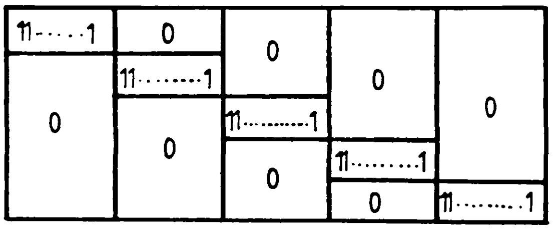
$$\text{Min} \sum_{(i,j)\in A_T} \sum_{k=1}^{K} (c_{ij} + \lambda_k t_{ij} + \delta_{ik} - \delta_{jk}) x_{ijk} - T \sum_{k=1}^{K} \lambda_k \tag{5-8}$$

subject to :

$$\sum_{(i,j)\in F_p} \sum_{k=1}^{K} x_{ijk} = 1 \qquad p=1,\dots,N \tag{5-9}$$

$$x_{ijk} \in \{0,1\} \tag{5-10}$$

The relaxed problem can easily be solved by inspection . It has a matrix of the form :



2-2 Network Flow Formulation 2 .

Now , instead of considering variables of the type x_{ijk} we define the following variable :

Let $x_{ij} = 1$ if arc (i,j) is in the optimal solution ;

$= 0$ otherwise ; (5-11)

The problem becomes :

$$\text{Min} \sum_{(i,j) \in A_T} c_{ij} x_{ij} \qquad (5\text{-}12)$$

subject to :

(x_{ij}) must form K paths of length T or less (5-13)

$$\sum_{(i,j) \in F_p} x_{ij} = 1 \qquad p = 1,\ldots,N \qquad (5\text{-}14)$$

$$\sum_{j \in \Gamma(i)} x_{ij} = \sum_{j \in \Gamma^{-1}(i)} x_{ji} \qquad \text{for each node } i \in V_T$$

(5-15)

$$\sum_{i \in \Gamma(S)} x_{Si} = \sum_{j \in \Gamma^{-1}(R)} x_{jR} = K \qquad S : \text{source} ; R : \text{sink}$$

2-2.1 The Time Constraints .Expressing linearly the time constraints , using x_{ij} , is itself a difficult problem . For this , we consider 3 options to represent these constraints :

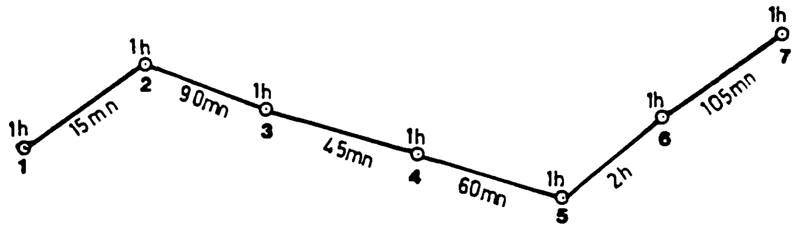
(i) First option : It consists of listing all paths of G_T whose length exceeds T (duty period) , and then for each such path P express the time constraints as :

$$\sum_{P} t_{ij} x_{ij} \leq T \qquad (5\text{-}16)$$

Clearly this option is impractical . The number of such constraints grows exponentially with the size of the problem (ie number of tasks).

(ii) Second option : Each path P of length > T consists of elementary subpaths P_i (i=1,2,3,...) whose length is greater than T but which do not contain any subpath of length > T. Thus in the path (of figure 5.3) of length 14hours 15 minutes, there are 4 elementary paths (1,2,3,4) ; (2,3,4,5) ; (3,4,5,6) and (5,6,7) .In this example T is assumed to be equal to 360 minutes.

Figure 5-3 : An Infeasible Path



Clearly by imposing the time constraints on all these elementary paths of G_T, we implicitly impose them on any path of length greater than T. Hence instead of considering all the paths of G_T of length greater than T, we need only consider the "elementary" paths and add to the problem the following constraints :

$$\sum_{p} t_{ij}x_{ij} \leq T \qquad p : \text{elementary path} \qquad (5\text{-}17)$$

Unfortunately we had to discard this approach because the number of such paths is still large .Thus for a GCSP of Ntasks and M vehicles' types the number of such constraints is approximately :

$$FNMa^5$$

where F is a constant, a is the average number of nodes that can be linked with a node.

(iii) Third option : A CSP version of this option has been given in chapter2, section 2-5.7.We saw in section 5 (chapter 2) that given a path whose length exceeds the work duty period the corresponding time constraint is of the form :

$$\sum_{A(P)} x_{ij} \leq h$$

where A(P) is the set of linking-arcs of P and h is a suitable constant determined by algorithm A6.

2-2.2 The Formulation .

If we let $x_{ij} = 1$ if arc(i,j) is in the optimal solution ;

$= 0$ otherwise ;

The problem becomes :

$$\text{Min} \sum_{(i,j)\in A_T} c_{ij}x_{ij} \qquad (5\text{-}18)$$

subject to :

$$\sum_{(\beta_i,\alpha_j)\in A(Q)} x_{ij} \leq h \qquad Q \in L \qquad (5\text{-}19)$$

$$\sum_{(i,j)\in F_p} x_{ij} = 1 \qquad p = 1,\ldots,N \qquad (5\text{-}20)$$

$$\sum_{j\in\Gamma(i)} x_{ij} = \sum_{p\in\Gamma^{-1}(i)} x_{pi} \qquad \text{for each node } i \in V_T$$

(5-21)

$$\sum_{i\in\Gamma(S)} x_{Si} = \sum_{i\in\Gamma^{-1}(R)} x_{iR} = K$$

$$x_{ij} \in \{0,1\} \tag{5-22}$$

where L is the set of all paths whose length exceeds the work duty period and where K,N,T,F_p,$\Gamma(i)$ and $\Gamma^{-1}(i)$ are as defined in section 2-1. S and R are the source and the sink of G_T respectively.

Like the first type network flow formulation , only the costs of the linking-arcs have been considered in the objective function . The time constraints (5-19) ensure that each crew schedule will not take more than the work duty period . The covering constraints (5-20) guarantee that each task of the GCSP will be covered only once by a single crew. Finally , we have the classical network flow constraints (5-21) which express the conservation of flow at each node of network G_T.

2-2.3 The Solution Technique .

If we let $\underset{\wedge}{\overset{\geqslant 0}{\lambda_k}}$(k=1,...,t) and υ_p(p=1,...,N) be the Lagrange multipliers attached to constraints (5-19) and (5-20) , the relaxed problem (LP), a minimum cost network flow problem , becomes :

$$\text{Min} \sum_{A_T} c_{ij}x_{ij} + \sum_{A_T} v_p x_{ij} + \sum_{A_T} \lambda_k x_{ij} - \sum_{k=1}^{t} \lambda_k h_k - \sum_{p=1}^{N} v_p \tag{5-23}$$

subject to :

constraints (5-21) and (5-23)

where t=|L|.

Constraints (5-19) which express the restriction on the maximum length of optimal paths (of G_T) cannot all be added to the GCSP at the same time because their number increases exponentially with the size of the problem (ie number of tasks). Consequently, they are added when necessary and relaxed in a Lagrangian way. Algorithm A7 has been used to determine a lower bound to the problem.

2-3 An Assignment Formulation .

2-3.1 The Formulation .

If we let $x_{ijk} = 1$ if arc $(i,j) \varepsilon A_T$ is covered by crew k in the optimal solution ;

$= 0$ otherwise ;

then the problem becomes :

$$\text{Min} \sum_{A_T} \sum_{k=1}^{K} c_{ij} x_{ijk} \tag{5-24}$$

subject to :

$$\sum_{A_T} t_{ij} x_{ijk} \leq T \qquad k=1,\ldots,K \tag{5-25}$$

$$\sum_{F_p} \sum_{k=1}^{K} x_{ijk} = 1 \qquad p=1,\ldots,N \tag{5-26}$$

$$\left.\begin{aligned} \sum_{i \in V_T} x_{ijk} &= 1 \qquad \forall j \in V_T \\ \sum_{j \in V_T} x_{ijk} &= 1 \qquad \forall i \in V_T \end{aligned}\right\} k=1,\ldots,K \tag{5-27}$$

$$x_{ijk} \in \{0,1\} \tag{5-28}$$

Constraints (5-25) and (5-26) are similar to constraints (5-2) and (5-3) . Constraints (5-27) are the classical assignment constraints .

It is worthwhile noting that when we express the cost assignment matrix , we should consider the following :

(a) $c_{ij} = 0$ if i=j;

(b) $c_{ij} = \infty$ if there is no arc linking i to j or if i=j=R or if i=j=S;

(c) we assume there is a dummy arc (R,S) of cost 0 (ie $c_{RS} = 0$) .

2-3.2 Example .

Let us consider the GCSP represented by the network G of figure 5.4.Assume T= 5 hours and INTER = 1 hour . Each time-window is divided into small intervals of 1 hour duration each .

The subnetwork G_T corresponding to vehicle's type 2 is given in figure 5.5 and the assignment matrix is represented in table 5.2 .

2-3.3 The Relaxed Problem .

If we let $\lambda_k \geq 0$ (k=1,...,K) and υ_p (p=1,...,N) be the Lagrange multipliers attached to constraints (5-25) and (5-26) respectively , the relaxed problem becomes :

$$\text{Min} \sum_{A_T} \sum_{k=1}^{K} (c_{ij} + \lambda_k t_{ij} + \nu_p) x_{ijk} - T \sum_{k=1}^{K} \lambda_k - \sum_{p=1}^{N} \nu_p \qquad (5\text{-}29)$$

subject to :

constraints (5-27) and (5-28)

Clearly this is an assignment problem .

Figure 5-4: A 3 Task-GCSP.

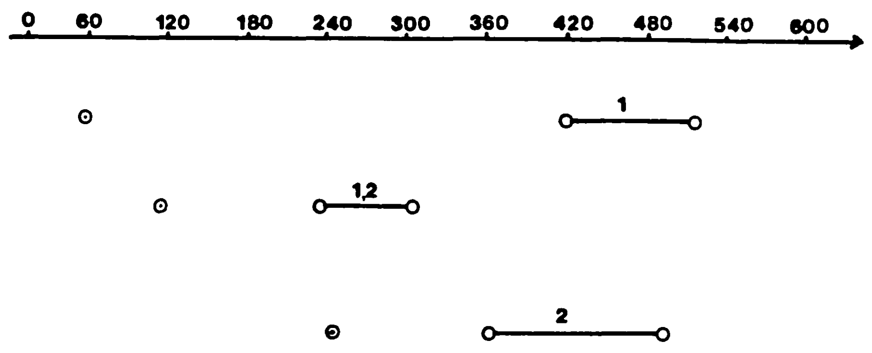


Figure 5-5: Subnetwork G_T.

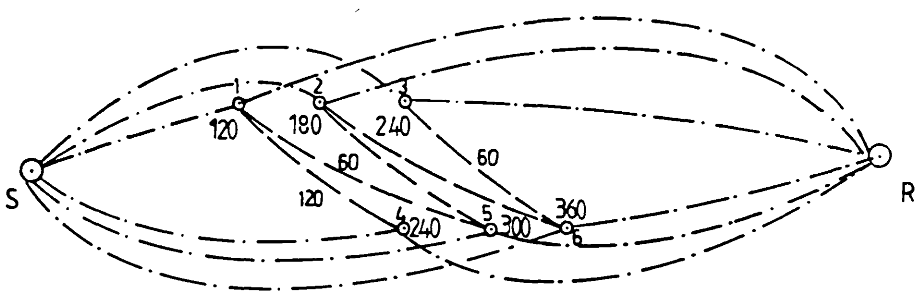


Table 5-2 : The Assignment Cost Matrix.

	S	1	2	3	4	5	6	T
S	∞	0	0	0	0	0	0	∞
1	∞	0	∞	∞	60	120	∞	0
2	∞	∞	0	∞	∞	60	120	0
3	∞	∞	∞	0	∞	∞	60	0
4	∞	∞	∞	∞	0	∞	∞	0
5	∞	∞	∞	∞	∞	0	∞	0
6	∞	∞	∞	∞	∞	∞	0	0
T	∞	∞	∞	∞	∞	∞	∞	∞

2-4 A Shortest Path Formulation .

2-4.1 The Formulation .

let $x_{ijk} = 1$ if arc $(i,j) \varepsilon A_T$ is covered by crew k in the optimal solution ;

$= 0$ otherwise ;

the problem becomes :

$$\text{Min} \sum_{A_T} \sum_{k=1}^{K} c_{ij} x_{ijk} \tag{5-30}$$

subject to :

$$\sum_{A_T} t_{ij} x_{ijk} \leq T \qquad k=1,\ldots,K \tag{5-31}$$

$$\sum_{F_p} \sum_{k=1}^{K} x_{ijk} = 1 \qquad p=1,\ldots,N \tag{5-32}$$

for $k=1,\ldots,K$ shortest path constraints (5-33)

$$x_{ijk} \in \{0,1\} \tag{5-34}$$

Constraints (5-31) and (5-32) are exactly the same as constraints (5-2) and (5-3). Also in the objective function only the costs of the linking-arcs have been considered .

2-4.2 The Relaxed Problem .

If we let λ_k $\geqslant 0$ (k=1,...,K) and υ_p(p=1,...,N) be the Lagrange multipliers attached to constraints (5-31) and (5-32) respectively , the relaxed problem becomes :

$$\text{Min} \sum_{G_T} \sum_{k=1}^{K} (c_{ij} + \lambda_k t_{ij} + \nu_l) x_{ijk} - T \sum_{k=1}^{K} \lambda_k - \sum_{p=1}^{N} \nu_p \qquad (5\text{-}35)$$

subject to :

constraints (5-33) and (5-34)

2-4.3 Solving The Shortest Path Problem .

Since graph G_T is acyclic , there exists a numbering of its nodes such that there exists an arc directed from i to j only if i < j . Assuming that the nodes are so numbered we have applied dynamic programming [23,24,57,122] which is :

U_j = shortest path from S to node j

$$U_s = 0$$

$$U_j = \min_{k<j} \{ U_k + a_{kj} \} \quad , \quad j=1,2,\ldots,N,R \qquad (5\text{-}36)$$

a_{kj} is the cost of the arc that links node k to node j ;

$a_{kj} = +\infty$ if there is no arc between k and j ;

$a_{Sj} = a_{jR} = 0$ for all j ε V_T ;

Now if we let E_j = set of all direct predecessors of j , equations (5-36) become :

$$U_S = 0$$

$$U_j = \min_{k \in E_j} \{ U_k + a_{kj} \} \quad , \quad j=1,2,\ldots,N,R \qquad (5\text{-}37)$$

The corresponding algorithm (based on (5-37)) runs in time $O(n^2)$.

2-5 A Shortest Weight-Constrained Path Formulation .

2-5.1 The problem Formulation .

The shortest path formulation described in the previous section can be written as follows :

$$\text{Min} \sum_{(i,j)\in A_T} \sum_{k=1}^{K} c_{ij} x_{ijk}$$

subject to :

$$\sum_{F_p} \sum_{k=1}^{K} c_{ij} x_{ijk} = 1 \qquad p=1,\dots,N$$

$$\left\{ \begin{array}{l} \sum_{A_T} t_{ij} x_{ij1} \le T \\ \text{shortest path constraints} \end{array} \right.$$

$$\left\{ \begin{array}{l} \sum_{A_T} t_{ij} x_{ij2} \le T \\ \text{shortest path constraints} \end{array} \right.$$

.... etc ...

$$\begin{cases} \sum_{A'_T} t_{ij} x_{ijK} \leq T \\ \text{shortest path constraints} \end{cases}$$

$$x_{ijk} \in \{0,1\}$$

Alternatively it can be formulated as :

$$\text{Min} \sum_{A'_T} \sum_{k=1}^{K} c_{ij} x_{ijk} \tag{5-38}$$

subject to :

$$\sum_{F_p} \sum_{k=1}^{K} x_{ijk} = 1 \qquad p=1,\ldots,N \tag{5-39}$$

for k=1,...,K shortest weight-constrained path constraints (5-40)

$$x_{ijk} \in \{0,1\} \tag{5-41}$$

2-5.2 The Relaxed Problem .

If we let λ_p (p=1,...,N) be the Lagrange multipliers attached to constraints (5-39), the relaxed problem becomes :

$$\text{Min} \sum_{A'_T} \sum_{k=1}^{N} (c_{ij} + \lambda_p) x_{ijk} - \sum_{p=1}^{N} \lambda_p \tag{5-42}$$

subject to : constraints (5-40) and (5-41)

This is a shortest-weigth-constrained path problem [56] .

2-5.3 Solving the Shortest Weight-Constrained Path Problem. Algorithm A14.

Meggido[118] has proved that this problem is NP-complete even if the graph G is directed . However , nothing has been said about the case where G is acyclic . In what follows we will not attempt to prove (or disprove) that it is also NP-complete for an acyclic graph but we will give a procedure for finding the shortest weight-constrained path in G_T that runs in time $O(Mn^2)$ where :

$$M = \left\lceil \frac{T - \min\limits_{i} \{d_i\}}{INTER} \right\rceil \tag{5-43}$$

in which d_i is the duration of task i ;

T is the work duty period ;

INTER is the time duration between 2 consecutive nodes (of G_T) corresponding to the same task .

In addition to the cost c_{ij} associated with arc (i,j) ε G_T, let W_{ij} and V_i be two weights associated with arc (i,j) and node i respectively .And let the weight of a path $P=(S,i_1,i_2,\dots,i_m,R)$ be :

$$W_p = W_{si_1} + V_{i_1} + W_{i_1 i_2} + V_{i_2} + \dots\dots + V_m + V_{mR}$$

Theorem : The procedure described below for finding the shortest path of weight W less than or equal to T ,from the source S to the sink R,runs in time $O(Mn^2)$ where M is as defined in (5-43).

(A)- Assumptions .

(a) The planning period of 24 hours is divided into small intervals of time (of 15 minutes,or 30 minutes,..etc...). We will refer to these intervals as time-units . In all our computational results we assume a time-unit of 15 minutes ;

(b) The window starting time WST_i , the window finishing time WFT_i , and the duration d_i of each task i are given as multiples of the time-unit ;

(c) WST_i's,WFT_i's and d_i's are expressed in terms of the time-unit ;

(d) In G_T , to each node j we associate the corresponding time K_j.

(B)- Algorithm A14.

We saw , in section 4 , that in an acyclic graph , dynamic programming equations for finding the shortest path from node S to node n are :

v_j = shortest path from node 1 to node j

$$v_1 = 0$$

$$v_j = \min_{k<j} \{ v_k + a_{kj} \} \qquad (j=1,\dots,n)$$

In our case they become :

v_j = shortest path from the source S to node j

$$v_s = 0 \tag{5-44}$$

$$v_j = \min_{k \in F_j} \{ v_k + a_{kj} \} \qquad (j=1,\dots,R)$$

where F_j = set of nodes k (k<j) of G_T that are linked to j .

Now to find the shortest path (of weight smaller than or equal to T) from S to R we need to determine the shortest paths (of weight$\leq$T) from S to all nodes of G_T.

In the shortest path (of weigth$\leq$T) from S to node j the first node i after S is such that :

$$K_j - M_j \leq K_i \leq K_j \tag{5-45}$$

with :

$$M_j = \frac{T - V_j}{\text{INTER}}$$

If we let E_k be the set of nodes that satisfy (5-45) and if we assume a dummy source S_k that is linked to all nodes of E_k with arcs of cost and weight zero , we will have

$$\left\{ \begin{array}{l} \text{Finding the shortest path} \\ \text{(of weight} \leq \text{T)from} \\ \text{S to j} \end{array} \right\} \Longleftrightarrow \left\{ \begin{array}{l} \text{Finding the shortest} \\ \text{path from } S_k \text{ to j} \end{array} \right\}$$

Hence dynamic programming equations for finding the shortest path from S_k to node j become :

$$\begin{aligned} &U_m^{(k)} = \text{shortest path from } S_k \text{ to } m \in E_k \\ &U_{S_k} = 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (5\text{-}46) \\ &U_m^{(k)} = \min_{h \in F_m} \{ U_h^{(k)} + a_{lm} \} \qquad m \in E_k \end{aligned}$$

Now if we generalize this idea and define

$$E_k = \{ i \varepsilon G_T / k \leq K_i \leq k+T \} \qquad k=1,\ldots,k_{max}$$

where M_i is as defined in (5-43) and

$$k_{max} = \frac{1440 - T}{\text{INTER}}$$

and if we assume dummy sources S_k(linked to all nodes of E_k with arcs of cost and weight zero) ,k=1,...,k_{max} dynamic programming equations for finding the shortest path (of weight less than or equal to T) , from S to R , are :

$$\begin{aligned} &U_j^{(k)} = \text{shortest path from } S_k \text{ to } j \in E_k \\ &U_{S_k} = 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (5\text{-}47) \\ &U_j^{(k)} = \min_{h \in F_j} \{ U_h^{(k)} + a_{lj} \} \qquad j \in E_k \end{aligned}$$

Clearly the shortest path (of weight $\leq$ T) from S to j is :

$$U_j^{k_j} \qquad \text{where } k_j = K_j - M_j$$

and the shortest path (of weight $\leq$ T) from S to R is :

$$U_T = \min \{U_j^{k_j} + a_{jT}\} \tag{5-48}$$

(C)- Worst Case Analysis of Algorithm A14 .

In the worst case and for each node j of G_T we have to compute u_j , M_j times. As far as computing time is concerned , this is equivalent to applying the algorithm for finding the shortest path (in G_T) from S to R , M times (M is as defined in (5-43)).

Hence we can say that Algorithm A16 , based on equations (5-47),(5-48) runs in time $O(Mn^2)$.

2-6 A Graph Expansion Based Formulation .

2-6.1 The Problem Formulation .

If we let x_{ij} = 1 if arc(i,j) is in the optimal solution ;

= 0 otherwise ;

The problem becomes :

$$\text{Min} \sum_{\tilde{A}_T} c_{ij}x_{ij} \qquad . \tag{5-49}$$

subject to :

$$\sum_{F_p} x_{ij} = 1 \qquad p=1,\ldots,N \tag{5-50}$$

$$\sum_{j \in \Gamma(i)} x_{ij} = \sum_{p \in \Gamma^{-1}(i)} x_{pi} \qquad \text{for each node } i \in \tilde{V}_T$$

(5-51)

$$\sum_{i \in \Gamma(S)} x_{Si} = \sum_{i \in \Gamma^{-1}(R)} x_{iR} = K$$

$$x_{ij} \in \{0,1\} \qquad (5\text{-}52)$$

where $\underset{\wedge}{\tilde{G}_T} = (\tilde{V}_T, \tilde{A}_T)$ is the expanded version of G_T obtained as output of algorithm A13 ;

N is the number of tasks ;

K is the number of crews ;

$\Gamma(i)$ (resp. $\Gamma^{-1}(i)$) is the set of successors (resp.predecessors) of node i ;

F_p is the set of nodes of G_T correspnding to task p .

This formulation has already been presented in chapter 4 (section 4). It is based on network $\tilde{G}_T$, which is the expanded version of G_T , obtained by applying algorithm A13 to G_T. In chapter 4 , the problem was solved by relaxing the integrality constraints (5-52) and using an LP package . In this section we will be relaxing constraints(5-50) in a Lagrangian fashion and solving the resulting minimum cost network flow problem using a netflow package which can handle problems of much larger size than the ones solved with the LP code.

2-6.2 The Relaxed Problem.

If we let $\lambda_p(p=1,...,N)$ be the Lagrange multipliers attached to constraints (5-50) the relaxed problem becomes :

$$\text{Min} \sum_{\tilde{T}} (c_{ij} + \lambda_p)x_{ij} - \sum_{p=1}^{N} \lambda_p \tag{5-53}$$

subject to :

constraints (5-51) and (5-52)

where constraints (5-51) and (5-52) are the network flow constraints.

3- Computational Results of the Six GCSP's Formulations .

3-1 The Test Problems .

All the results considered in this chapter have been obtained by testing all the different formulations and algorithms on ten randomly generated GCSP's of size varying from 10 to 50 tasks . Tables 5-3 and 5-4 give the size , in terms of number of task-nodes and arcs , of network G_T representative of the GCSP and its expanded network $\tilde{G}_T$ respectively .

As we saw in the previous section all the first five GCSP's formulations were based on G_T whereas the graph expansion based formulation was based on $\tilde{G}_T$.

Depending on the number n of tasks of the GCSP considered each problem has been assumed to have the following number V of vehicle types :

(a) if $n \leq 15$ then $V = 2$

(b) if $20 < n \leq 30$ then $V=3$

(c) if $30 < n \leq 45$ then $V=4$

(d) if $n = 50$ then $V=5$

Table 5-3 : Size of Network G of the 10 Test-Problems .

Problem	Number of Tasks	Number of Vehicle Types	Number of Crews	Network 1	Network 2	Network 3	Network 4	Network 5	Number of Total Vertices	Number of Total Arcs
1	5	2	2	1644 [90]	613 [58]	*	*	*	148	2257
2	10	2	3	2987 [113]	3891 [125]	*	*	*	238	6878
3	15	2	4	5241 [128]	10309 [177]	*	*	*	305	15550
4	20	3	5	5551 [130]	6344 [142]	19212 [245]	*	*	517	31107
5	25	3	6	7532 [150]	7946 [160]	24018 [272]	*	*	582	39496
6	30	3	7	15667 [218]	16935 [226]	11831 [189]	*	*	633	44433
7	35	4	8	7852 [167]	10826 [181]	18882 [236]	13934 [204]	*	788	51494
8	40	4	10	4365 [138]	12420 [190]	14035 [207]	30902 [313]	*	848	61722
9	45	4	12	13380 [149]	14697 [217]	19905 [243]	36029 [347]	*	956	84011
10	50	5	12	53404 [302]	18718 [238]	16715 [247]	14748 [225]	12339 [208]	1220	115924

Table 5-4 : Size of Network $\tilde{G}$ of the 10 Test-Problems .

Problem	Number of Tasks	Number of Vehicle Types	Number of Crews	Total Number of Nodes	Total Number of Arcs	Maximum Nodes in a Network	Maximum Arcs in a Network
1	5	2	2	1708	10330	58	561
2	10	2	3	3504	40770	86	1572
3	15	2	4	5430	121624	170	5373
4	20	3	5	9952	235836	219	9805
5	25	3	6	11368	321275	253	13184
6	30	3	7	13727	461330	218	10908
7	35	4	8	18082	623145	256	15781
8	40	4	10	20440	815083	283	18653
9	45	4	12	21069	872538	285	19125
10	50	5	12	21766	964325	212	19590

Table 5-5 : Computational Results for the Network Flow Formulation 1 .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	[1] Running Time+	[1] Lower Bound	[1] Gap [%]	[2] Running Time+	[2] Lower Bound	[2] Gap [%]
1	5	2	2	45	2.27	40.19	10.75	*	*	*
2	10	2	3	105	4.46	99.53	5.21	*	*	*
3	15	2	4	165	8.94	156.60	5.09	*	*	*
4	20	3	5	225	*	*	*	15.80	208.82	7.19
5	25	3	6	285	*	*	*	20.01	250.74	12.02
6	30	3	7	345	*	*	*	22.44	331.41	3.94
7	35	4	8	405	*	*	*	26.34	396.45	2.11
8	40	4	10	450	*	*	*	31.48	371.88	17.36
9	45	4	12	495	*	*	*	30.52	473.37	4.37
10	50	5	12	570	*	*	*	45.35	522.01	8.42

+ Seconds of CYBER 855 (Fortran Compiler).

Table 5-6 : Computational Results for Network Flow Formulation 2 .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	Running Time+	Lower Bound	Gap [%]
1	5	2	2	45	1.56	40.09	10.91
2	10	2	3	105	2.20	92.05	12.33
3	15	2	4	165	3.667	138.98	15.76

Table 5-7 : Computational Results for the Assignment Formulation .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	Running Time+	Lower Bound	Gap [%]
1	5	2	2	45	2.32	40.89	9.14
2	10	2	3	105	5.12	99.61	5.12
3	15	2	4	165	10.75	157.01	4.84

+ Seconds of CYBER 855 (Fortran Compiler).

Table 5-8 : Computational Results for the Shortest Path Formulation .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	[1] Running Time+	[1] Lower Bound	[1] Gap [%]	[2] Running Time+	[2] Lower Bound	[2] Gap [%]
1	5	2	2	45	0.97	41.15	8.55	*	44.48	1.14
2	10	2	3	105	2.62	99.81	4.94	*	102.73	2.16
3	15	2	4	165	5.45	157.46	4.57	*	163.42	0.97
4	20	3	5	225	11.24	209.61	6.84	*	218.00	3.11
5	25	3	6	285	14.36	251.57	11.73	*	276.42	3.01
6	30	3	7	345	16.35	335.20	2.84	*	344.33	0.19
7	35	4	8	405	18.72	335.38	17.19	*	388.35	4.11
8	40	4	10	450	22.88	419.13	6.86	*	438.17	2.63
9	45	4	12	495	23.44	450.60	8.97	*	469.06	5.24
10	50	5	12	570	30.13	550.79	3.37	*	556.78	2.32

+ Seconds of CYBER 855 (Fortran Compiler).
[1] The initial Lagrange multipliers are set to 0 .
[2] The initial Lagrange multipliers are set to $-\min_i c_i$.

Table 5-9 : Computational Results for the Shortest Weight Constrained Path Formulation.

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	Running Time+	Lower Bound	Gap [%]
1	5	2	2	45	2.59	44.21	1.76
2	10	2	3	105	10.62	104.80	0.19
3	15	2	4	165	21.45	162.37	1.59
4	20	3	5	225	61.24	221.06	1.75
5	25	3	6	285	79.37	278.90	2.14
6	30	3	7	345	80.36	341.65	0.97
7	35	4	8	405	82.87	404.84	0.04
8	40	4	10	450	147.88	444.06	1.32
9	45	4	12	495	130.76	490.94	0.82
10	50	5	12	570	195.13	558.89	1.95

+ Seconds of CYBER 855 (Fortran Compiler).

Table 5-10 : Computational Results for the Graph Expansion Based Formulation .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	Running Time+	Lower Bound	Gap [%]
1	5	2	2	45	8.04	44.01	2.20
2	10	2	3	105	25.81	104.33	3.50
3	15	2	4	165	72.43	164.09	0.55
4	20	3	5	225	142.79	222.95	0.91
5	25	3	6	285	192.77	282.18	0.99
6	30	3	7	345	267.87	340.10	1.42
7	35	4	8	405	362.67	403.54	0.36

+ Seconds of CYBER 855 (Fortran Compiler).

3-2 The Generation Process .

For each task i , let :

WST_i be the window starting time of task i ;

WFT_i be the window finishing time of task i ;

d_i be the duration of task i ;

then we have :

$WST_i = 1440\,\alpha$ (minutes)

$WFT_i = WST_i + (1440\text{-}WST_i)\alpha$ (minutes)

$d_i = 45 + 120\alpha$ (minutes)

where α is a randomly generated number in the range [0,1].

The work duty period was assumed in all cases to be equal to 7 hours (ie 420 minutes) and the smallest time interval was taken to be 15 minutes .

3-3 Some Details about the Lagrangian Relaxation .

Except in the case of the shortest path and shortest weight constrained path formulations the initial Lagrange multipliers have been set to zero . In all cases , the step size of the subgradient optimization was taken to be equal to 4 and the value of the initial upper bound was that given by the heuristic (see section 5-4). For all relaxations we performed 20 subgradient iterations . However for the second relaxation of the first type network flow formulation 100 subgradient iterations were performed .

3-4 Comparison of the Computational Results .

For each problem the following pieces of information have been considered (tables 5-5 thru 5-10) :

(i) the size of the problem in terms of the number of tasks (column 2) and the number of crews (column 4);

(ii) the number of vehicle's types (column 3) . It is worthwhile noting at this point that the assignment of tasks to vehicle's types has already been explained in section 6 (chapter 4);

(iii) the value of the optimal solution (column 5) was obtained later after the use of the tree search procedure (see sections 4 and 5) .

For each formulation and for each problem we presented the following information :

(iv) the value of the lower bound obtained by relaxing the corresponding formulation in a Lagrangian fashion ;

(v) the gap between the lower bound and the optimal solution value . This gap gives an idea of how far is the lower bound from the optimal solution value . It is equal to :

$$\text{gap} = \frac{\text{optimal solution value - lower bound}}{\text{optimal solution value}}$$

(vi) and finally the computing time expressed in CP seconds of the CYBER 855 using the fortran compiler FTN5 .

The following general remarks concerning each formulation can be derived from tables 5-5 thru 5-10 .

3-4.1 The First Type Network Flow Formulation (Table 5-5) .

Both relaxations of the GCSP have been considered to determine a lower bound to the problem . With the 1st relaxation only problems of up to 15 tasks could be tackled . This was due to the limitations imposed by the network flow package which is considered quite effective in solving network flow problems . For larger problems of up to 50 tasks we needed to use the second relaxation which does not require any special purpose package since it can easily be solved by inspection .

It is worthwhile noting that the table does not compare the performance of the 2 relaxations since it can easily be proved [68] that the bound obtained with the 2nd relaxation can be in the best case as good as the one obtained with the first relaxation . The use of the second relaxation was motivated by our wish to solve larger GCSP's .

3-4.2 The Second Type Network Flow Formulation (Table 5-6).

Like the previous formulation and due to the same reason only problems of up to 15 tasks could be tackled with the network flow package used . It is worthwhile mentioning here that each time we add new time constraints the Lagrangian relaxation procedure is started by setting the initial multipliers equal to those which have produced the best bound with the previous problem (ie before adding the current time constraints) .

3-4.3 The Assignment Formulation (Table 5-7) .

Several efficient algorithms and packages have been devised to solve the assignment problem [39] . Due to the limitations imposed by the package only problems of up to 15 tasks could be tackled .

3-4.4 The Shortest Path Formulation (Table 5-8) .

Table 5-8 compares the performance of the Lagrangian relaxation of the shortest path formulation when the Lagrange multipliers are set to 0 and when they are set to the lowest arc's cost . Clearly , in the latter case we can see that the convergence to the optimal solution (although not completely achieved) is much faster .

3-4.5 The Shortest Weight Constrained Path Formulation (Table 5-9).

Like the previous formulation all the 10 randomly generated problems of size varying between 10 and 50 tasks could be tackled . Taking into account the better performance of the Lagrangian relaxation of the shortest path formulation when the multipliers are set initially to the minimum arc's cost we decided to adopt the same strategy with this formuation of the GCSP .

3-4.6 The Graph Expansion Based Formulation (Table 5-10).

Due to the limitations imposed by the network flow package used , only problems of up to 35 tasks could be considered . Comparing these results (table 5-10) with those of tables 5-6 and 5-7 the natural question that comes to one's mind is : "knowing that the size of the expanded network G_T is much larger than that of $\widetilde{G}_T$ why is it that with the network flow formulation based on G_T only problems of up to 15 tasks could be tackled while with this approach which is based on $\widetilde{G}_T$ much larger GCSP's could be handled ?" . This is due to the fact that the resulting minimum cost network flow problem is solved not on the entire network (whether G_T or $\widetilde{G}_T$) in one go but on the corresponding subnetworks (see section 5,chapter 4) and since the subnetworks of $\widetilde{G}_T$ are much smaller than those of G_T (although many more in number) this answers the above question .

3-4.7 Choosing the Most Performant Formulation to be Embedded in the Tree Search Procedure .

Two formulations have been chosen as candidates to be embedded in the tree search procedure . As far as the quality of the lower bound and the size of the GCSP are concerned the shortest weight constrained path formulation is clearly the one which has performed better than all the others . It will be the first candidate formulation to be embedded in the tree search procedure (section 4) for obtaining lower bounds .

The choice of another candidate was motivated by the fact that a lot of computing time was required to obtain this bound in the case of the shortest weight constraint path . This other candidate is the shortest path formulation for which a much smaller computing time was necessary to get a lower bound not as good as that of the first candidate but of the same quality as all the others.

4 - The Tree Search Procedure [7,20,50,101].

4-1 Description of the Tree Search Algorithm .

In this section we will give a general description of the general tree search algorithm . The details are considered in the other sections .

Step 0 : Apply a heuristic algorithm to determine a feasible solution to the problem ie an upper bound (say Z_u) ; go to step 1 ;

Step 1 : Consider node 0

* Determine a lower bound to the problem by solving the relaxed problem (shortest path or constrained shortest path) ;let Z_b be this lower bound ;

* If $Z_b=Z_u$ STOP ,the solution produced by the heuristic is optimal ;

* Else , paritition node 0 into nodes 1,2,...(see section 4.5) ;

* Set i=1 and go to step 2 ;

Step 2 : Consider node i

* Reduce the size of the graph corresponding to node i (see section 4.6) ;

* Solve the relaxed problem (see section 4.11)

* Go to step 3 ;

Step 3 : Fathoming tests

* If node i is fathomed go to step 4 ;

* Else , go to step 5 ;

Step 4 : Backtracking Process

* Backtrack to ancestor's nodes ;

* Set i to be the first non-explored node(and non-fathomed) ;

* In case all the nodes have been fathomed , STOP , the optimal solution to the problem is the feasible one with minimum cost ;

* Else , go to step 2 ;

Step 5 : Branching process

* Go to step 6 ;

Step 6 : Partitioning process

* Partition node i into nodes $i_1,i_2,...$

* Set $i=i_1$ and go to step 2 ;

4-2 Comparison of Two Heuristics .

4-2.1 A Greedy Heuristic .

The algorithm consists of the following steps :

Step 0 : * Set R=0 ; i=0

* For each type of vehicle p_i (i=1,...,V) , form a list LIST(i) of all tasks that can be covered by type p_i ;(V is the number of vehicle types);

* Go to step 1 ;

Step 1 : * Set i=i+1 ; consider type of vehicle p_i ; go to step 2;

Step 2 : * Set R=R+1 ; form route R ; go to step 3 ;

Step 3 : * Remove all the tasks of R from LIST(j) , j=i,...,V ;

* if LIST(i) is empty go to step 4 ; else go to step 2 ;

Step 4 : * If LIST(j) = 0 for j=i+1,...,V STOP ;else go to step 1 .

The main step in the above algorithm is step 2 which consists of forming route R . It can be described as follows : " Let i* be the last task in route R . Also let ST_{i*} and FT_{i*} be respectively the starting and finishing times of task i* in route R .

Step 2-0 : * Determine the non-assigned task i_0 which can be covered by the current vehicle type and which has the least window starting time WST_i ;

* Set $i^* = i_0$ and $ST_{i*} = WST_i$; go to step 2-1 ;

Step 2-1 : * Among the non-assigned tasks which can be covered with the current type of vehicle determine the one (say i_c) which satisfies both following properties :

(a) task i_c can be linked to task i* ie $FT_{i*} < WFT_i$;

(b) the addition of task i_c to route R will not produce an overflow of the route R ie the length of the resulting path will not exceed the work duty period ;

* Go to step 2-2 ;

Step 2-2 : If no task has ben found in step 2-1 return to the main algorithm ie step3;Else set $i^* = i_c$, ST_{i*} = minimum(WST_i,FT_{i*}) ; go to step 2-1 .

Example : If we apply the above algorithm to the 10 task-GCSP of figure 5-1 we obtain the following solution :

* **Vehicle type 1 :** It consists of the following two routes :

Route 1 :

- Task 2 starts at time 75 ;
- Task 7 starts at time 165 ;
- Task 4 starts at time 255 ;
- Task 8 starts at time 375 ;

Route 2 :

- Task 6 starts at time 150 ;
- Task 3 starts at time 270 ;
- Task 9 starts at time 385 ;

* **Vehicle type 2 :** It consists of the following route :

Route 3 :

- Task 10 starts at time 345 ;
- Task 5 starts at time 510 ;
- Task 1 starts at time 585 ;

Total Cost : 105

4-2.2 A Heuristic Based on The Shortest Weight Constrained Path .

This heuristic is based on an adapted version of the approach described in section 2-5 .

The problem with the procedure suggested in that section which consists of finding the shortest weight constrained path in G_T is that some tasks might be repeated. By this we mean that in that shortest path we might find two nodes (or more) corresponding to the same task .

In what follows we first suggest a method of tackling this problem (ie finding the shortest weight constrained path without repeated tasks) and then we describe the corresponding heuristic for finding a feasible solution to the GCSP .

(A)- Finding the Shortest Weight Constrained Path without Repeated Tasks .

The algorithm of section 5 was based on the following dynamic programming equations :

$$U_j^{(k)} = \text{shortest path from } S_k \text{ to } E_k$$
$$U_{S_k} = 0 \qquad (5\text{-}47)$$
$$U_j^{(k)} = \min_{l \in F_j} \{U_l^{(k)} + a_{lj}\} \quad j \in E_k$$

where E_k,F_j and S_k are as defined in section 2-5.3 .

Now if we assume T = 7 hours and the minimum task duration equal 1 hour , then each route will contain at most 5 tasks (the minimum cost of linking 2 tasks being 15 minutes) .Hence in any feasible path of G_T , each node k cannot have more than 4 task-parents .A task-parent is defined as follows :
"if a path P of G_T consists of nodes $P=(i_1,i_2,i_3,i_4,i_5)$ and if $F=(f_1,f_2,f_3,f_4,f_5)$ is the corresponding set of tasks then f_j (j=1,...,4) is called the task-parent of nodes $i_{j+1},i_{j+2},...,i_5$. Consequently , if we define :
$F_j' = \{ k<j$ / k is linked to j and the task corresponding to j is not a task-parent of K$\}$
then equations (5-47) with F_j replaced by F_j' will give the shortest weight constrained path , without repeated tasks , from the source S to the sink R .

Before going to the next section , it is worthwhile noting that the corresponding procedure still runs in time $O(n^2)$.

(B)- The Algorithm .

(i) Assumptions . It is clear that with the current cost coefficients the shortest weight constrained path , without repeated tasks , from S to R will have cost zero . This is because S and R are linked to all nodes of G_T with arcs of cost 0 . For this let us assume $C_{max} = \max c_{ij}$ and let us define new costs :

$$c'_{ij} = c_{ij} - c_{max} \quad \text{for all } (i,j) \, \varepsilon \, G_T$$

This implies that the new costs are non-positive . By choosing the costs in that way we can achieve actually two aims :

(i) minimze gap = duty period - length of route . Hence we will use the minimum number of crews to cover all tasks ;

(ii) minimize the total cost .

(ii) The Algorithm

Step 0 : Let $G_1,...,G_m$ be the subnetworks of G_T corresponding to the m types of vehicle ; Go to step 1

Step 1 : For each i=1,...,m determine the shortest weight constrained path P_i without repeated nodes in G_i ; Go to step 2 ;

Step 2 : Let P* be the shortest path among $P_1,...,P_m$ and let F* be the corresponding set of tasks ; Remove all the nodes corresponding to the tasks of F* from all G_i's ; Go to step 3 ;

Step 3 : If all G_i 's are empty STOP ; Else go to step 1 .

4-3 A Simple Reduction Test .

The size of network G_T can be reduced further by eliminating the arcs that have no chance to be in the optimal solution . The following is a very simple test that has proved to be quite efficient .

Consider an N task- K crew GCSP . In graph theoretic terms , a feasible solution x_0 of the problem consists of K paths of G_T containing N nodes corresponding to the N tasks . The natural question that comes to mind is :" how many non-required arcs does x_0 possess?". It can easily be proved that x_0 consists of N-K linking-arcs. Now assume we have a feasible solution x_0 with value z_0 . What is the maximum cost c_{max} an arc of x_0 can have ?. Let c_{min} be the minimum cost of a linking-arc of x_0. Since x_0 has got exactly N-K linking-arcs , in the worst case N-K-1 of these arcs will have cost c_{min} . Hence ,

$$c_{max} < z_0 - (N-K-1)\, c_{min} \qquad (5\text{-}54)$$

in our case we have made the assumption in all our problems that c_{min} =15 minutes . Hence any arc of G_T whose cost violates condition (5-54) is eliminated since it will have no chance of being in the optimal solution .

In the tree search , the feasible solution x_0 is at first provided by one of the two heuristics suggested in section 4-2 . In the process of going down the tree , whenever a better feasible solution is found , constraint (5-54) is applied to remove more arcs from G_T and thus reducing its size . This is particularly useful when we know that , in general ,in a tree search procedure the optimal solution is often found at the early stage of building the tree .

4-4 The Branching Process .

At first we tried the classical branching which consists of branching on a single variable . The results obtained were not satisfactory . So , we decided to adopt another branching strategy which can be described as follows :

Consider the following set of constraints which are found in both the shortest path formulation and the constrained shortest path formulation :

$$\sum_{F_p} \sum_{k=1}^{K} x_{ijk} = 1 \qquad p=1,...,N \qquad (5\text{-}55)$$

where N is the number of tasks of the problem ;

K is the number of crews ;

F_p is the set of arcs ,of network G_T, whose initial node corresponds to task p;

x_{ijk} = 1 if arc (i,j) is covered by crew k in the optimal solution ;

= 0 otherwise .

These constraints ensure that each task must be covered once only by a single crew . They mean that among all the arcs leaving all the nodes of G_T, corresponding to task p (p=1,...,N) , only one arc should be picked up in the optimal solution .

Now if we want 2 tasks p and q (p precedes q) to be directly linked in the optimal solution we need to impose the following constraint :

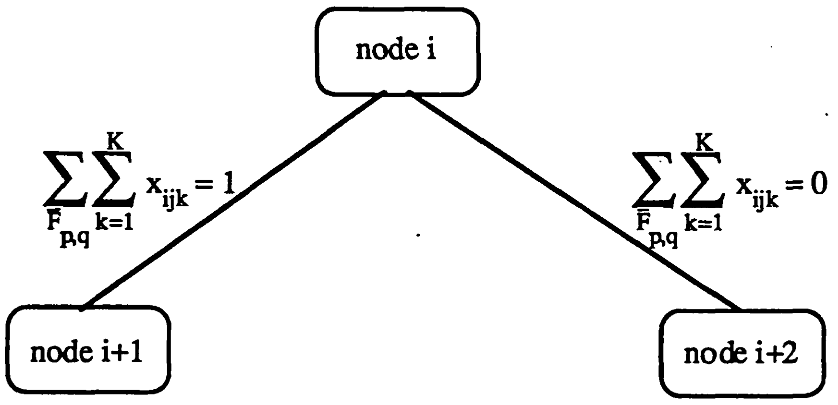
$$\sum_{\bar{F}_{p,q}} \sum_{k=1}^{K} x_{ijk} = 1 \tag{5-56}$$

where $\bar{F}_{p,q}$ is the set of all arcs of G_T whose initial node corresponds to task p and terminal node to task q .

If we do not want p and q to be directly linked in this order we just have to impose :

$$\sum_{(i,j) \in F_{p,q}} \sum_{k=1}^{K} x_{ijk} = 0 \tag{5-57}$$

Having chosen 2 tasks p_0 and q_0 (how ? see section 4-7) our branching strategy is as follows :


node i
$\sum_{\bar{F}_{p,q}} \sum_{k=1}^{K} x_{ijk} = 1$
$\sum_{\bar{F}_{p,q}} \sum_{k=1}^{K} x_{ijk} = 0$
node i+1
node i+2

Before saying something on the choice of tasks p_0 and q_0 , and before giving more details about the branching strategy (see section 4.7) we first consider the partitioning process .

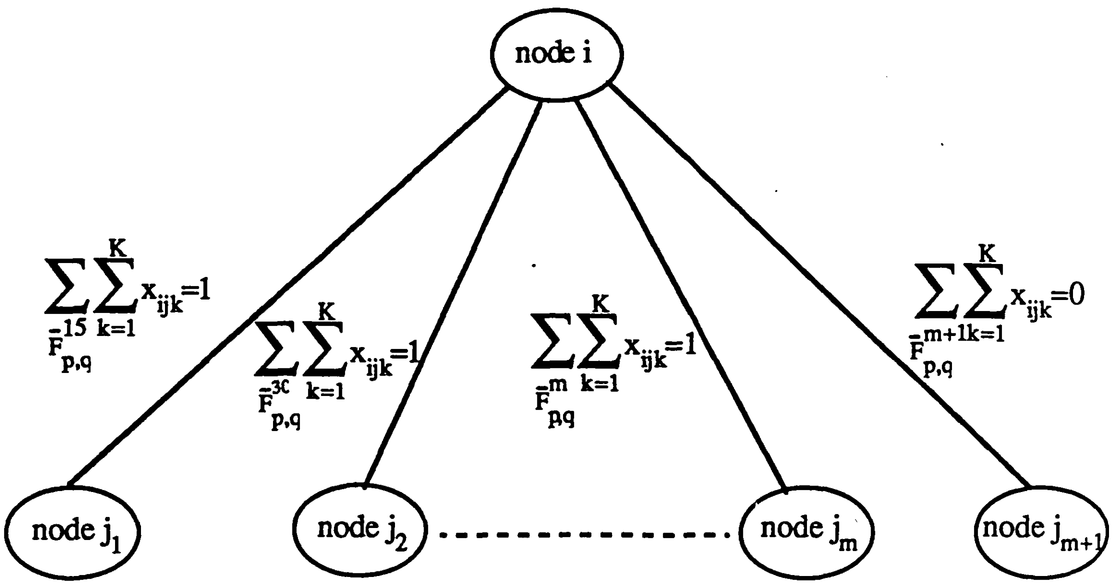
4-5 The Partitioning Process .

We saw in the reduction test that an arc of G_T can be eliminated if it has length greater than :

$$c_{max} = z_0 - (N\text{-}K\text{-}1)\, c_{min}$$

where z_0 is the value of the best feasible solution found so far , c_{min} is the minimum arc's cost . For our computational results we have noticed that for all problems considered the maximum length of an arc does not exceed 60 minutes ie c_{max}=60minutes . Hence with the assumption that the costs of the arcs are multiple of 15 minutes the only values it can take are 15,30,45 and 60 minutes . This encouraged us in choosing the following partitioning strategy :

Going back to the branching technique and considering constraint (5-56) , if we take into account the fact that the only values an arc of $\bar{F}_{p,q}$ can take are 15,...,M (where M can be equal to 15 or 30 or 45 or 60) then node i of the tree is partitioned as follows : **Figure 5-6** : Partitioning of node i.


node i
$\sum_{\bar{F}^{15}_{p,q}} \sum_{k=1}^{K} x_{ijk}=1$
$\sum_{\bar{F}^{30}_{p,q}} \sum_{k=1}^{K} x_{ijk}=1$
$\sum_{\bar{F}^{m}_{p,q}} \sum_{k=1}^{K} x_{ijk}=1$
$\sum_{\bar{F}^{m+1}_{p,q}} \sum_{k=1}^{K} x_{ijk}=0$
node j_1
node j_2
node j_m
node j_{m+1}

where m = M/15 and

for t=15,...,M ; $\bar{F}^{t}_{p,q}$ is the set of arcs of G_T with cost t that join vertices of G_T corresponding to task p to vertices of G_T corresponding to task q ;

or alternatively $\bar{F}^{t}_{p,q} = \{ (i,j) \in \bar{F}_{p,q} / c_{ij} = t \}$

4-6 Graph Reduction .

In the tree search procedure the size of network G_T can be reduced using one or some of the following reduction tests :

(i) By imposing constraints (5-57) to the problem all arcs of $\bar{F}_{p,q}$ can be eliminated from G_T ;

(ii) Consider the following constraint :

$$\sum_{(i,j) \in \bar{F}^{t}_{pq}} \sum_{k=1}^{K} x_{ijk} = 1 \qquad (5\text{-}58)$$

It means that tasks p and q (in this order) will be linked in the optimal solution by an arc of duration t (t=15,...,M) . Hence we can consider tasks p and q as forming a single task k with the following characteristics :

* the duration d_k of task k = duration of task p + duration of task q + t

* the starting time window WST_k of k is such that :

$$WST_k = \max(WST_p, WST_q - d_p - 15)$$

* the finishing time window of task k is given by :

$$WFT_k = \min(WFT_p, WFT_q - d_p - 15)$$

* task k can be covered by the type of vehicle that can cover both tasks p and q ;

As a consequence of this , and going back to figure 5.6 the problem at nodes $j_1, j_2, \ldots, j_m$ will be smaller than the problem at node i by one task . This means that the deeper we go down the tree the smaller the problem becomes .

(iii) As a consequence of (ii) , if in the tree's path from node 0 to node K a certain task schedule has been found which is full up (ie the addition of any other task will result in a schedule of length greater than the duty period) then all the corresponding tasks will be removed from the sons' nodes (and successors' nodes) and the number of crews will be reduced by one .

(iv) Whenever the best upper bound to the problem has been updated, the reduction test of section 4.3 is applied to remove more arcs from the graph .

4-7 Choice of Tasks p_0 and q_0 .

The branching strategy is a depth first search . Starting from node 0 of the tree , we go down the tree trying to build feasible schedules . Whenever a node is fathomed (see section 4-8) we backtrack (see section 4-9) and continue our descent down the tree .

Assume that at a certain node i of the tree a set of crew schedules has already been found . The choice of tasks p_0 and q_0 should be made in such a way :

* to complete an existing crew schedule ; or

* to start a new schedule in case all the existing schedules are full up .

It is whorthwhile to note that if we want to complete an existing schedule we must have one of the two following cases :

* p_0 is the last task of the schedule and q_0 is not in the schedule ; or

* q_0 is the first task of the schedule and p_0 is not in the schedule .

However if a new schedule is started then any 2 tasks not already assigned can be chosen . Also we have to make sure when choosing the tasks to branch on that their addition will not result in a schedule which violates the time restriction .

Finally assume that we are at node i of the tree (see figure 5-6) and that we have chosen the two tasks p_0 and q_0 to branch on , the next node that will be explored after node i is node j_1 ie the node corresponding to constraint (5-58) with $\bar{F}^{15}_{p_0,q_0}$.

4-8 Fathoming Tests .

A node i of the tree is fathomed if one of the following conditions is satisfied :

(i) the value of the lower bound (say, z_b) at that node is greater than or equal to the value of the current upper bound (say,z_u);

(ii) the solution at node i is feasible . In this case if the value of z_0 of this solution is such that $z_0 < z_u$ then update the value of the current upper bound by setting $z_u=z_0$;

(iii) the number of remaining crews will not be able to cover all the remaining tasks. By remaining we mean that if we consider in the tree the path from node 0 to node i , then all the tasks that have not been considered are called remaining tasks .

(iv) the solution is infeasible .

4-9 Backtracking Process.

Consider figure 5-6 . Let us first present the following definitions :

(i) **Father node** : node i is said to be the father node of nodes $j_1,j_2,...,j_{m+1}$;

(ii) **Son node** : nodes $j_1,j_2,...,j_{m+1}$ are the son nodes of node i ;

(iii) **Brother node** : nodes $j_1,j_2,...,j_{m+1}$ are called brother nodes ;

(iv) **Uncle node** : the brothers of node i are the uncles of nodes $j_1,j_2,...,j_{m+1}$.

With these definitions in hand , the backtracking process can be described as follows :

If a node k is fathomed then the next node to be considered after it is its nearest brother node . In case all the brother nodes have been fathomed then try to explore the nearest uncle node to the father node and so on and so forth ...etc... If all nodes have been fathomed stop the tree search procedure : the optimal solution to the problem is the best integer solution found so far .

4-10 Task Ordering .

In the branching process , when choosing the next pair of tasks to branch on we have implicitly assumed that the tasks are ordered . So the natural question is how are they ordered ?

First , the tasks are divided into groups . Each group correspond to a certain type of vehicle ie they are as many groups as there are vehicle types . The first group corresponds to the first vehicle type , the second group corresponds to the second vehicle type ,.. etc ...

A task that can be covered by more than one vehicle type will be considered in all the corresponding groups . Within the group the tasks are ordered in ascending order of their window starting times . In case of ties the task with the shortest time window is first considered . Finally it is worthwhile noting that if in the branching process a task has been already assigned to a certain schedule , then this task will not be considered in any son node or descendent node .

4-11 Solving the Relaxed Problem in the Tree .

To obtain a lower bound to the problem at every node of the tree we considered, in the 2 previous sections 2 and 3 , several different formulations of the GCSP and compared the corresponding lower bounds obtained from the Lagrangian relaxation . The shortest weight constrained path formulation (SWCP) proved to be the best one in that the quality of the lower bound obtained ,over all 10 randomly generated GCSP's , was better than all the others .

So we first thought that this will be the formulation that will be embedded in the tree search procedure for obtaining lower bounds . However when we noticed that the computing time required to get such a lower bound was considerably large compared to the other formulations which are based on network G_T we decided to embed both this formulation and the shortest path formulation in the tree search procedure and then choose the one that performs better overall . The shortest path formulation was chosen among the other formulations because in addition of producing lower bounds as good as the others(except the SWCP) the corresponding algorithm was much faster .

4-12 Example .

Let us consider the 10 tasks-3 crew GCSP of figure 5-1. The optimal solution was obtained by the heuristic . But for the sake of explaining very briefly how the tree search procedure works , we assumed the upper bound to be equal to 120 and we run the program with this upper bound . The tree we obtained is represented in figure5-7 .

The tasks have been ordered as follows :

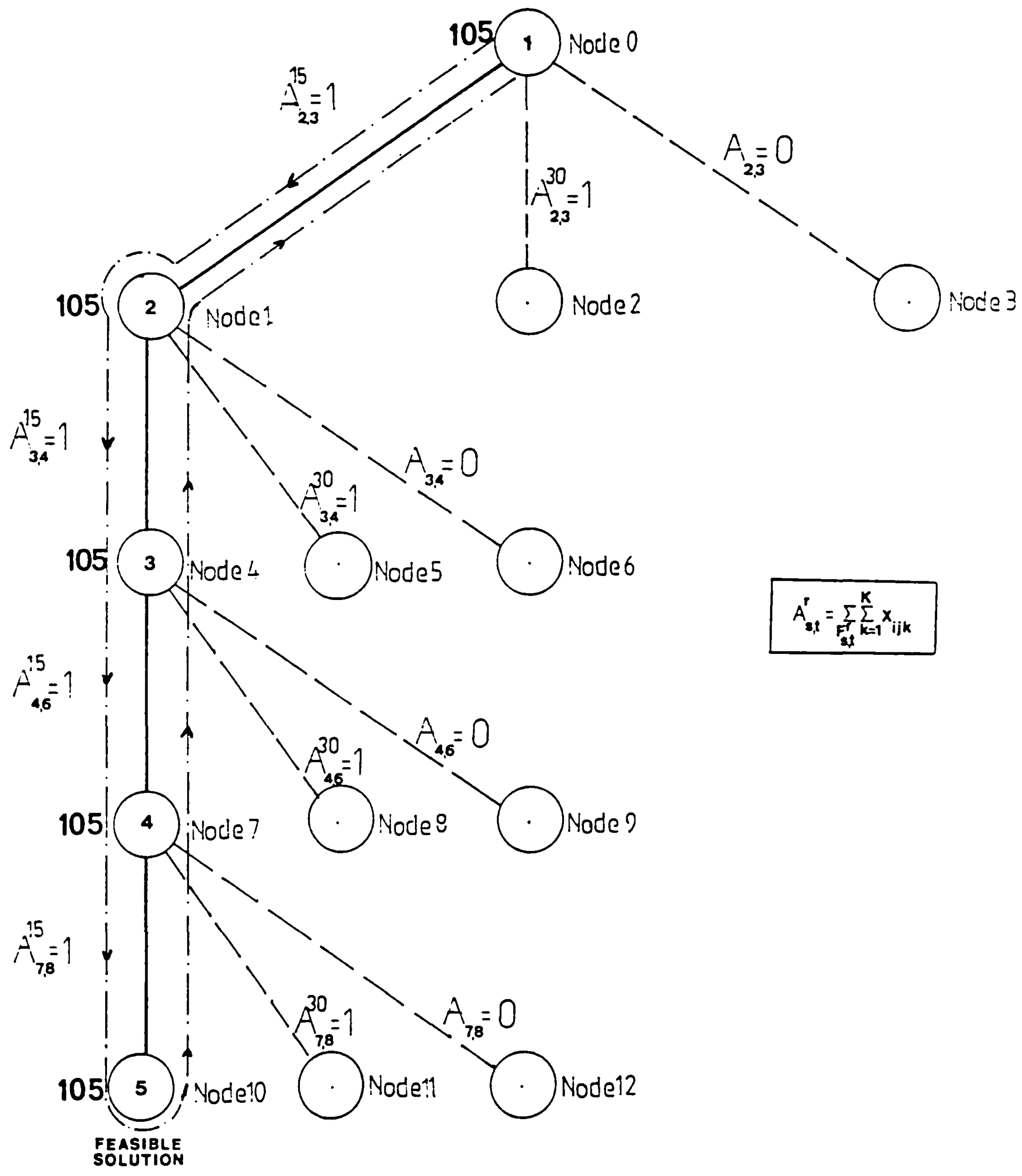
* Group 1 consists of tasks 2,3,4,6,7,8 and 9 ;

* Group 2 consists of tasks 1,3,4,5,9,10 .

Using a depth first search a feasible integer solution was found at node 10 . Backtracking to the father node 7 and comparing the values of the lower bounds of respectively nodes 7 and 10 we could fathom nodes 11 and 12 without investigating them . At this point it is worthwhile explaining the meaning of the different figures associated with the tree search of figure5-7 . The figures inside the circles (which represent the nodes) represent the order in which the nodes are visited . The terminal nodes at the extremities of the dotted branches mean that they did not need to be considered as they were fathomed before investigation . The dotted arrows show the direction in which the tree was explored . The figures beside each node give the value of the corresponding lower bound .

Going back to where we stopped (ie fathoming nodes 11 and 12) the backtracking process was carried out from father to grand-father ,fathoming on our

Figure 5-7 : Tree Search of the 10 task-GCSP (table5-1).



way the brothers, till we finally arrived back to the head node 1 which is in its turn fathomed due to the fact that its lower bound was equal to the value of the feasible solution found at node 10 . This shows that this feasible solution is optimal for the whole problem .

4-13 Reducing the Size of the Tree.

When generating the input data we have made the following assumptions :

(a) The planning period of 24 hours is divided into small intervals of time of 15,or 30,...,or 60 minutes,...etc... . We will refer to these intervals as time unit. In all our computational results we assume a time unit of 15 minutes ;

(b) The window starting time , the window finishing time and the duration of each task are given as multiples of 15 .

As a direct consequence of this , the value of the optimal solution will be equal to a multiple of 15 . Hence the value of the lower bound can each time be rounded off to the first multiple of 15 greater than or equal to the value of this lower bound .

5- Computational Results of the Tree Search Procedure .

Table 5-11 compares the performance of the two heuristic algorithms presented in section 4-2 . It is clear that the heuristic based on the shortest weight constrained path formulation (SWCP) has performed much better , in all cases considered , than the greedy heuristic .

Using the reduction test of section 4-6 we show in tables 5-12 and 5-13 the effect of this test on respectively the size of network G_T and the computational times of the shortest path and SWCP formulations . It has been noticed that the larger the average time window is and the more efficient the test becomes . This will be discussed in section 6 .

Comparing , in table 5-14 , the tree search procedures based on respectively the shortest path and the SWCP formulations we noticed that although a smaller number of nodes needed to be explored in the latter case to obtain the optimal solution , the overall computing time required by the former formulation has proved to be much better . In table 5-15 we give more information about the tree search procedure based on the shortest path formulation .

In section 2-5.3 we made the reasonable assumption that the planning period of 24 hours was divided into smaller intervals of time of 15 minutes called time units . This means that since the cost coefficients represent the time durations of the corresponding arcs then they all are multiples of 15 . As a result of this the lower bound at each node can be rounded off to the nearest multiple of 15 . Table 5-16 presents the new values of the lower bounds of the shortest path problem formulation when the "modulo 15" consideration is taken into account . Finally the corresponding inprovements in time of the tree search procedure are shown in table 5-17.

Table 5-11 : Computational Results for the Two Heuristic Algorithms .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	[1] Running Time+	[1] Upper Bound	[1] Gap [%]	[2] Running Time+	[2] Upper Bound	[2] Gap [%]
1	5	2	2	45	0.01	45	0	0.13	45	0
2	10	2	3	105	0.01	105	0	0.46	105	0
3	15	2	4	165	0.01	165	0	1.31	165	0
4	20	3	5	225	0.01	255	13.33	2.65	225	0
5	25	3	6	285	*	*	*	4.11	315	10.52
6	30	3	7	345	0.02	405	17.39	6.95	360	4.34
7	35	4	8	405	0.02	420	3.70	9.28	405	0
8	40	4	10	450	*	*	*	12.55	480	6.67
9	45	4	12	495	0.02	585	18.18	14.01	525	6.06
10	50	5	12	570	0.03	630	10.53	18.97	585	2.62

+ Seconds of CYBER 855 (Fortran Compiler).
[1] Heuristic 1 ;
[2] Heuristic 2 ;

Table 5-12 : Size of Network G_T After the Reduction Process .

Problem	Number of Tasks	Network 1	Network 2	Network 3	Network4	Network5	Number of Vertices	Number of Arcs after Reduction	Number of Arcs before Reduction	Proportion of Arcs Eliminated
1	5	*	*	*	*	*	*	*	*	*
2	10	*	*	*	*	*	*	*	*	*
3	15	*	*	*	*	*	*	*	*	*
4	20	*	*	*	*	*	*	*	*	*
5	25	1255	1324	2076	150	160	582	6585	39496	83.3%
6	30	758	837	578	218	226	633	2173	44433	95.1%
7	35	*	*	*	*	*	*	*	*	*
8	40	621	1789	2006	4471	138	848	8887	61722	85.6%
9	45	1297	1405	1947	3475	149	956	8124	84011	90.3%
10	50	3869	1402	1285	1090	967	1220	8613	115924	92.6%

Table 5-13 : Computational Results for the Shortest Path and Shortest Weight Constrained Path Formulations after the Reduction of G_T .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	[1] Running Time+	[1] Running Time+	[1] Improvement [%]	[2] Running Time+	[2] Running Time+	[2] Improvement [%]
1	5	2	2	*	*	*	*	*	*
2	10	2	3	*	*	*	*	*	*
3	15	2	4	*	*	*	*	*	*
4	20	3	5	*	*	*	*	*	*
5	25	3	6	14.4	5.4	62.2	79.4	25.5	67.8
6	30	3	7	16.2	4.4	72.7	80.4	21.0	73.9
7	35	4	8	*	*	*	*	*	*
8	40	4	10	22.9	5.7	75.0	147.9	41.6	71.9
9	45	4	12	23.4	8.6	63.1	130.8	43.5	66.7
10	50	5	12	30.1	4.3	85.7	195.1	30.9	84.2

+ Seconds of Cyber 855 (Fortran Compiler) .
[1] Shortest Path Formulation .
[2] Shortest Weight Constrained Path Formulation

Table 5-14 : Computational Results for the Tree Search Procedure .
Comparison of the Shortest Path and the Shortest Weight Constrained Path Formulations .

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	[1] Number of Nodes Explored	[1] Running Time+	[1] Optimum Given by Heuristic	[2] Number of Nodes Explored	[2] Running Time+	[2] Optimum Given by Heuristic
1	5	2	2	45	*	*	yes	*	*	yes
2	10	2	3	105	*	*	yes	*	*	yes
3	15	2	4	165	*	*	yes	*	*	yes
4	20	3	5	225	*	*	yes	*	*	yes
5	25	3	6	285	972	432.7	no	440	920.4	no
6	30	3	7	345	139	60.8	no	206	388.4	no
7	35	4	8	405	*	*	yes	*	*	yes
8	40	4	10	450	625	402.3	no	340	1215.9	no
9	45	4	12	495	1036	800.2	no	364	1300.3	no
10	50	5	12	570	715	297.2	no	470	1056.4	no

+ Seconds of CYBER 855 (Fortran Compiler).
[1] Shortest Path Formulation .
[2] Shortest Weight Constrained Path Formulation .

Table 5-15 : Computational Results for the Tree Search Procedure (based on the shortest path).

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	Lower Bound	Upper Bound	Gap [%]	Number of Nodes	Total Time+	Time+ at Node 0
1	5	2	2	45	44.5	45	1.14	*	*	*
2	10	2	3	105	102.7	105	2.16	*	*	*
3	15	2	4	165	163.4	165	0.97	*	*	*
4	20	3	5	225	218.0	225	3.11	*	*	*
5	25	3	6	285	276.4	315	3.01	972	432.7	5.4
6	30	3	7	345	344.3	360	0.19	139	60.8	4.4
7	35	4	8	405	388.4	405	4.11	*	*	*
8	40	4	10	450	438.2	480	2.63	625	402.3	5.7
9	45	4	12	495	469.1	525	5.24	1036	800.2	8.6
10	50	5	12	570	556.8	585	2.32	715	197.2	4.3

+ Seconds of CYBER 855 (Fortran Compiler).

Table 5-16 : Bounds Comparison for the Shortest Path Formulation .

Problem	Number of Tasks	Number of Vehicle Types	Number of Crews	Previous Lower Bound	New Lower Bound	Previous Gap [%]	New Gap [%]
1	5	2	2	44.5	45	1.1	0.
2	10	2	3	102.7	105	2.2	0.
3	15	2	4	163.4	165	1.0	0.
4	20	3	5	218.0	225	3.1	0.
5	25	3	6	276.4	285	3.0	0.
6	30	3	7	344.3	345	0.2	0.
7	35	4	8	388.4	390	4.1	3.7
8	40	4	10	438.2	450	2.6	0.
9	45	4	12	469.1	480	5.2	3.0
10	50	5	12	556.8	570	2.3	0.

Table 5-17 : Computational Results for the Tree Search Procedure Based on the New Lower Bounds (of table 5-16).

Problem	Number of Tasks	Number of Vehicles Types	Number of Crews	Value of Optimal Solution	[1] Lower Bound	[1] Upper Bound	[1] Gap [%]	[2] Number of Nodes	[2] Total Time+	[2] Time+ at Node 0
1	5	2	2	45	45.0	45	0.	*	*	*
2	10	2	3	105	105.0	105	0.	*	*	*
3	15	2	4	165	165.0	165	0.	*	*	*
4	20	3	5	225	225.0	225	0.	*	*	*
5	25	3	6	285	285.0	315	0.	118	76.7	5.4
6	30	3	7	345	345.0	360	0.	175	133.2	4.4
7	35	4	8	405	390.0	405	3.70	*	*	*
8	40	4	10	450	450.0	480	0.	269	181.7	5.7
9	45	4	12	495	480.0	525	3.03	528	428.4	8.6
10	50	5	12	570	570.0	585	0.	*	*	*

+ Seconds of CYBER 855 (Fortran Compiler).
[1] Node 0 of the Tree .
[2] Remaining Nodes of the Tree .

Table 5-18 : Efficiency of the Reduction Test when the Average Time-Window Varies.

Average Time-Window [hours]	Size of G40 [arcs]	Size of G40 After Reduction	Percentage of eliminated Arcs [%]
1	10540	8438	19.9
2	21505	17204	20.0
3	29920	15229	49.1
4	43732	23195	47.0
5	55887	17625	68.5
6	59160	20244	65.8
7	76033	8485	88.8
8	80877	11242	86.1
9	93713	14017	85.0
10	104932	10750	89.8

6- Conclusion .

A branch and bound procedure based on the Lagrangian relaxation of a shortest path formulation of the GCSP has been considered in this chapter . With this technique GCSP's that could not be tackled with the graph's expansion of the previous chapter , have been solved to optimality . The problems considered varied in size between 10 and 50 tasks and had a task time window allowed to vary between 0 and 24 hours .

Lagrangian relaxation was used to determine at each node of the tree a lower bound to the problem . The formulation of the problem as a shortest path problem plus additional time constraints was decided after 5 other integer programming formulations were considered and the corresponding computing times and lower bounds were compared . Although the Lagrangian lower bound of the shortest path based formulation was not the best one , the computing time that was required to get it was considerably better than all the others . To determine a good upper bound to the GCSP two heuristic algorithms were considered . The one that performed better was subsequently used to produce an upper bound to each one of the test problems considered . It is based on a polynomial algorithm used to solve the shortest weight constrained path relaxation of the GCSP .

Using a very simple but efficient reduction test the large size of the coresponding network G_T of the test problems was substantially reduced . This is one of the factors that improved the efficiency of the tree search procedure . It has been noticed that as the average task time window of the problem increases the reduction test becomes more and more efficient as it can be seen from the results of table 5-18 which show the proportion of arcs eliminated from network G_{40} by the reduction test when the average task time window varies between 1 and 10 hours . G_{40} is the network representative of a 40 task-GCSP with 3 types of vehicle. We can see from table 5-18 that when the average task time window converegs to 1 hour the reduction test becomes almost useless . However when this average time-window is small the size of the corresponding network is as a result also small . Hence we basically do not need at all to use any reduction test in this case . In fact the need of reducing the size of network G_{40} becomes necessary only when it gets larger ie when the average time window increases and in this case the test proved to be quite efficient . Hence we can say that for a given GCSP , whatever is the range in which the average time window varies , by using the reduction test when necessary a reasonable size network G_T can be obtained to work on .

CHAPTER 6

CONCLUSIONS

Two versions of the crew scheduling problem have been considered . The first version called simply crew scheduling problem (CSP) has been solved efficiently using just an LP package . A tree search procedure based on Lagrangian relaxation was used to solve the second version that was called general crew scheduling problem (GCSP) . The difference between the 2 versions is that in the first one the starting and finishing times of each task were specified and fixed beforehand , whereas in the 2nd version the starting time of each task is given within time-windows . Also while only one type of vehicle has been assumed in the case of CSP to service all tasks, several types of vehicle are considered to cover the tasks of the GCSP .

At first the CSP was formulated as a minimum cost network flow problem plus additional time constraints . The formulation proved to be so effective that out of all 101 randomly generated problems of size varying between 5 and 30 tasks , eighty one were solved to optimality by using just an LP package . For the remaining problems a cutting-plane algorithm has been devised . This algorithm consists of a combination of logical cuts and Gomory's cuts . When the logical cuts were first added only few Gomory's cuts were necessary to reach the optimal solution .

The drawback of this technique is that only small problems of up to 30 tasks could be tackled . This was due to the limitations imposed by the existing LP codes which cannot handle the very large size linear problems produced by the formulatiom as the number of tasks increases . To overcome this drawback another formulation and another integer programming solution technique were considered . In this formulation which is still based on the minimum cost network flow problem , the additional time constraints were generated , in a dynamic way , as needed and subsequently relaxed in a Lagrangian fashion . Our aim in using the Lagrangian relaxation was to derive good lower bounds for the problem that will be embedded in the tree search procedure . Unfortunately , the quality of the bound was so poor , (20% away from the optimum for certain problems), that the idea was dropped .

Using graph theoretic concepts , we then derived a graph expansion technique that proved to be very effective . The technique consists of expanding the network G representative of the CSP into another network G in which all paths are feasible for the CSP .As a result of this the problem was then formulated as a minimum cost network flow problem plus additional set-partitioning type constraints (instead of time constraints) which guarantee that each task must be covered once only by a single crew . With such a formulation in hand we first thought of relaxing the set partitioning type constraints in a Lagrangian fashion and solve the relaxed problem with a minimum cost network flow package to obtain a good lower bound that would be embedded in a tree search procedure . Fortunately , we did not need to go that far . Thus out of more than 250 randomly generated CSP's of size varying between 10 and 150 tasks , the optimal integer solution was found in all cases by just solving the linear relaxation of the problem with an LP package . These results are surprisingly good , especially when we know that the CSP is an NP complete problem . Then we tried to determine whether it is always the case that the solution of the linear relaxation of the CSP is optimal . Many attempts were made , unsuccessfully , to produce a single example for which this does not apply but each time an example was produce it was subsequently solved exactly with the LP package . To prove that the problem can always be solved by an LP package is itself a very hard problem and only partial proofs could be produced . It is worthwhile to mention at this point that the proof of the integrality of the extreme points of the CSP's polytope is of great importance in the area of algorithms' complexity . Thus this proof will solve a problem that in spite of the huge effort deployed to solve it , nobody up till now has been able to prove (or disprove) . This is to prove that problem complexity class P is equal to class NP .

Decomposition algorithms [53,104] have been devised to solve large scale linear programming problems which involve thousands of rows and hundreds of variables . It is worthwhile noting that taking advantage of the fact the the solution of the linear relaxation of the problem is optimal , large size CSP's which can involve hundreds of tasks can be solved efficiently with one such decomposition method.

Using an adapted version of the graph expansion technique (GET) two different extensions of the CSP were subsequently considered and solved in exactly the same manner as the CSP ie just using an LP package . These versions are :

(i) the multiple depot CSP in which several depots of known capacity are assumed . Each crew (or vehicle) must return at the end of the day to its original depot;

(ii) the CSP with rest periods : in this problem encountered mainly in the airline industry the planning period is supposed to be one week of 7 days . Between any 2 consecutive trips each crew is to have a rest period of at least T_1 hours and at most T_2 hours ($T_1 < T_2$) where T_1 and T_2 are given ;

The GCSP was first tackled with the graph expansion technique. For the GCSP's that have small task time windows the GET has proved to be as effective as for the CSP .Thus , GCSP's of up to 50 tasks could be optimally solved using just an LP package . However as the time window increases it becomes impossible to use GET . this is due to the enormously large size of the expanded network that no existing LP package could handle . Thus when the task average time window is allowed to vary between 00.00 and 24.00 hours only GCSP's of up to 10 tasks could be solved .

In our attempt to solve larger size GCSP's a branch and bound procedure based on the Lagrangian relaxation of a shortest path problem formulation of the GCSP was devised . Using the Lagrangian relaxation to obtain lower bounds for the tree search procedure six different integer programming formulations were considered . After comparing the corresponding running times and lower bounds produced two formulations were chosen as possible candidates to be embedded in the tree search procedure . And finally it appeared that the shortest path based formulation , although the corresponding bound was not the best found , was the most suitable and the most performant to be embedded in the tree search procedure.

The success of any branch and bound procedure depends entirely on the quality of both the upper and lower bounds of the problem . To find a good upper bound to the GCSP two heuristics algorithms have been considered and compared. The one that performed better was subsequently used to determine upper bounds to the problem. It is based on a polynomial algorithm that has been devised to solve the shortest weight constrained problem (applied to our case) .

The results obtained with this tree search procedure , the efficiency of which was improved by some reduction tests , were satisfactory . Thus all randomly generated GCSP's of size varying between 10 and 50 tasks and with a task time window allowed to vary in the range [00.00-24.00hours] were solved to optimality .

REFERENCES

1. *Agmon S. [1954]*, " The Relaxation for Linear Inequalities", Canadian Journal of Math., Vol 6, pp382-392.

2. *Arabeyre J.P., Fearnley J., Steiger F.C. and Teather W. [1969]*, "The Airline Crew Scheduling Problem : a survey", Transportation Science, Vol 3, pp140-163.

3. *Assad A. [1980]*, "Models for Rail Transportation," Transportation Research, Vol 14A, pp205-220.

4. *Baker E. [1979]*, "Efficient Heuristic Solutions for the Airline Crew Scheduling Problem", D.B.A thesis, University of Maryland, College Park, Maryland.

5. *Baker E. , Bodin L.D., Finnegan W.F. and Ponder R.J. [1979]*, "Efficient Heuristic Solutions to an Airline Crew Scheduling Problem", AIEE Transactions, Vol 11, pp79-84.

6. *Baker E., Bodin L., and Fisher M. [1980]*, "The Development and Implementation of a Heuristic Set Covering Based System for Air Crew Scheduling," Working Paper No.80-015, University of Maryland.

7. *Balas E. [1965]*, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", Operations Research, Vol 13, pp517-546.

8. *Balas E. [1971]*, "Intersection Cuts-A New Type of Cutting-Plane for Integer Programming," Operations Research, Vol 19, pp19-30.

9. *Balas E. [1974]*, "Intersection Cuts from Disjunctive Constraints," Man. Res. Rep. No. 330, Carnegie-Mellon University, February, 1974.

10. *Balas E. [1975]*, "Disjunctive Programming : Cutting-Planes from Logical Conditions," Talk Given at SIGMAP-UW Conference, April 1974, Published in O.L. Mangasarian, R.R. Meyer, and S.M. Robinson (Eds).,Nonlinear Programming, Vol 2, Academic Press, New York.

11. *Balas E. and Christofides N. [1981]*, "A Restricted Lagrangian Approach for the

Travelling Salesman Problem," Mathematical Programming, Vol 21, p19.

12. *Balas E. and Ho A.[1980]*, "Set Covering Algorithms Using Cutting-Planes,Heuristics and Subgradient Optimization : A Computational Study", Mathematical Programming, Study 12, Combinatorial Optimization, pp37-60.

13. *Balas E. and Jeroslow R. [1975]*, "Strengthening Cuts for Mixed Integer Programs," MSSR no. 359, GSIA, Carnegie-Mellon University, February.

14. *Balas E. and Padberg M. [1979]*, "Set Partitioning - a Survey," In Combinatorial Optimization, N. Christofides, A. Mingozzi, P. Toth and C. Sandi (Editors).

15. *Ball M. [1980]*, "A Comparison of Relaxation and Heuristics for certain Crew and Vehicle Scheduling Problems", presented at the National ORSA/TIMS Meeting, Washington D.C..

16. *Ball M., Bodin L. and Dial R. [1980]*, "Experimentation with a Computerized System for Scheduling Mass Transit Vehicles and Crews," Presented at the International Workshop on Urban Passenger,Vehicle,and Crew Scheduling, the University of Leeds, England.

17. *Bazaraa M.S. and Jarvis J.J. [1977]*, "Linear Programming and Network Flows", John Wiley & Sons, New York.

18. *Bazaraa M.S. and Goode J.J. [1979]*, "A Survey of Various Tactics for Generating Lagrangean Multipliers in the Context of Lagrangian Duality", European Journal of Operational Research, Vol 3, pp 322-338.

19. *Bazaraa M.S. and Shetty C.M. [1979]*, "Nonlinear Programming : Theory and Algorithms", John Wiley & Sons, New York.

20. *Beale E.M.L. [1965]*, "Survey of Integer Programming", Operational Research Quartely, Vol 16, pp219-228.

21. *Beale E.M.L. [1979]*, "Branch and Bound Methods for Mathematical Programming Systems", Annals of Discrete Mathematics, Vol 5: Discrete Optimization II, pp 201-219.

22. *Beasley J.[1983]*, "Route First-Cluster Second Methods for Vehicle Routing," Omega, Vol 11, pp403-408.

23. *Bellman R. [1957]*, "Dynamic Programming", Princeton University Press, Princeton, N.J.

24. *Bellman R. and Dreyfus S.E. [1957]*, "Applied Dynamic Programming", Princeton University Press, Princeton, N.J.

25. *Bellmore M., Bennington G. and Lubore S. [1971]*, "A Multivehicle Tanker Scheduling Problem," Transportation Science, Vol 5, No 1, pp36-74.

26. *Benders J. [1962]*, "Partitioning Procedures for Solving Mixed Variable Programming Problems," Numerische Mathematic, Vol 4.

27. *Benichou M. et Al [1971]*, "Experiments in Mixed Integer Linear Programming," Mathematical Programming, Vol 1, pp76-94.

28. *Bennington G.E. and Rebibo K.K. [1975]*, "Overview of the RUCUS Vehicle and Scheduling Program (BLOCKS)", in Workshop on Automated Techniques for Scheduling of Vehicle Operations for Urban Public Transportation Systems, held in Chicago, Bergmann D. and Bodin L. editors.

29. *Berge C.[1970]*, "Graphes et Hypergraphes," Dunod,Paris.

30. *Berge C.[1972]*, "Balanced Matrices", Mathematical Programming, Vol 2, pp19-31.

31. *Bodin L.and Berman L. [1979]*, "Routing and Scheduling of School Buses by Computer," Transportation Science, Vol 13, pp113-129.

32. *Bodin L.and Golden B. [1981]*, "Classification in Vehicle Routing and Scheduling," Networks,Vol 11,pp97-108.

33. *Bodin L., Golden B., Assad A. and Ball M. [1982]*, "The State of the Art in the Routing and Scheduling of Vehicles and Crews", Computers and Oper. Res., Vol

10, pp63-212.

34. *Bodin L.and Kursh S. [1978]*, "A Computer-Assisted System for the Routing and Scheduling of Street Sweepers," Operations Research, Vol 26, pp525-537.

35. *Bodin L., Rosenfield D. and Kydes A. [1978]*, "UCOST, A Micro Approach to a Transit Planning Problem", Journal of Urban Analysis, Vol 5, No 1, pp 47-69.

36. *Camion P.[1965]*, "Characterization of Totally Unimodular Matrices", Proc. Am. Math. Soc., Vol 16, pp1068-1073.

37. *Chalmet L.G. and Gelders L.F.[1976]*, "Lagrangian Relaxations for a Generalised Assignment-Type Problem", Katholieke Universiteit Leuven , Working Paper, No 76-121.

38. *Cheddad H. and Christofides N. [1986]*, "Algorithms for Crew Scheduling Problems", Working Paper, Department of Management Science, Imperial College, University of London.

39. *Christofides N. [1975a]*, "Graph Theory, an Algorithmic Approach", Academic Press.

40. *Christofides N. [1975b]*, "Vehicle Routing," RAIRO Rech. Oper., Vol 10, pp55-70.

41. *Christofides N. [1979]*, "The Travelling Salesman Problem," in Combinatorial Optimization (edited by N. Christofides,A. Mingozzi,P.Toth and C. Sandi), John Wiley & Sons.

42. *Christofides N. [1985]*, "Vehicle Routing," In E.L Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (eds.), The Travelling Salesman Problem, Wiley, Chichester, Ch.12.

43. *Christofides N., Mingozzi A., and Toth P.[1981]*, "State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems," Networks, Vol 11, pp145-164.

44. *Christofides N. and Paixao J. [1982]*, "State-Space Relaxation Algorithms for

the Set-Covering Problem," Imperial College Report No IC,OR,81/12.

45. *Chvatal V. [1979]*, "A Greedy Heuristic for the Set-Covering Problem", Mathematics of Operations Research, Vol 4, pp233-235.

46. *Cook, S.A. [1971]*, "The Complexity of Theorem Proving Procedures," Proc.3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, 151-158.

47. *Cook, S.A. [1973]*, "A Hierarchy for Nondeterministic Time Coomplexity," J. Comput. System Sci., Vol 7, pp 343-353.

48. *Crowder H.[1974]*, "Computational Improvement of Subgradient Optimization," IBM Technical Report RC 4907, Yorktown Heights, New York.

49. *Crowder H., Johnson E.L. and Padberg M. [1983]*, "Solving Large-Scale Zero-One Linear Programming Problems", Operations Research, Vol 31, No 5, pp 803-834.

50. *Dakin R.J.[1965]*, "A Tree Search Algorithm for Mixed-Integer Programming Problems," Computer Journal, Vol 8, pp250-255.

51. *Dantzig G. [1963]*, Linear Programming and Extensions, Princeton University Press, Princeton, N.J.

52. *Dantzig G. and Fulkerson D. [1954]*, "Minimizing the Number of Tankers to Meet a Fixed Schedule", Naval Research Logistics Quarterly, Vol 1, pp 217-222.

53. *Dantzig G. and Wolfe P. [1961]*, "The Decomposition Algorithm for Linear Programming," Econometrica, Vol 9, No. 4.

54. *Deo N. [1974]*, "Graph Theory with Applications to Engineering and Computer Science", Prentice-Hall.

55. *Desrosiers J., and Desrochers M. , Soumis F.[1984]*, "Routing with Time-Window by Column Generation," Networks, Vol 14, pp545-565.

56. *Desrosiers J., Pelletier P., and Soumis F. [1983]*, "Plus Court Chemin avec

Contraintes d'Horaires," R.A.I.R.O. Rech. Oper., Vol 17, pp357-377.

57. *Dreyfus S.E. and Law A.M. [1977]*, "The Art and Theory of Dynamic Programming", Academic Press, New York.

58. *Finnegan W. [1977]*, "A Network Model for Bidline Generation", in FEC Technical Report, Federal Express Corporation, Memphis, TN.

59. *Fisher M.J. and M.O. Rabin [1974]*, "Super-Exponential Complexity of Presburger Arithmetic," in R.M.Karp (ed), Complexity of Computation, American Mathematical Society, Provedence, RI, 27-41.

60. *Fisher M.L. [1981]*, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", Management Science, Vol 27, No 1, pp 1-18.

61. *Fisher M.L. [1985]*, "An Applications Oriented Guide to Lagrangian Relaxation", Interfaces, Vol 15, No 2, pp 10-21.

62. *Florian M., Guerin G., and Bushel G. [1976]*, "The Engine Scheduling Problem in a Railway Network," INFOR Journal, Vol 14, pp121-138.

63. *Ford L. and Fulkerson D. [1962]*, "Flows in Networks", Princeton University Press, Princeton, New Jersey.

64. *Forrest J.J.H., Hirst J.P.H., and Tomlin,J.A. [1974]*,"Practical Solution of Large Mixed Integer Programming Problems with UMPIRE," Management Science, Vol 20, pp736-773.

65. *Gallo G. and Pallottino S. [1986]*, "Shortest Path Methods, a Unifying Approach", Mathematical Programming Study, Vol 26, pp 38-64.

66. *Garey M.R. and Johnson D.S. [1979]*, "Computers and Intractability: a Guide to the Theory of NP-Completeness", Freeman & Co, San Francisco.

67. *Garfinkel R.S. [1979]*, "Branch And Bound Methods for Integer Programming", in Combinational Optimization, Christofides N., Mingozzi A., Toth P. and Sandi C. editors, John Wiley & Sons, pp 1-20.

68. *Garfinkel R.S. and Nemhauser G.L. [1972]*, "Integer Programming", John Wiley & Sons.

69. *Gavish B. [1978]*, "On Obtaining the "Best" Multipliers for a Lagrangean Relaxation for Integer Programming", Computers and Operations Research, Vol 3, pp 322-338.

70. *Geoffrion A.M. [1967]*, "Integer Programming by Implicit Enumeration and Balas' Method", SIAM Review, Vol 9, pp 178-190.

71. *Geoffrion A.M. [1974]*, "Lagrangean Relaxation for Integer Programming", Mathematical Programming Study, Vol 2, pp 82-114.

72. *Geoffrion A.M. and Graves G. [1974]*, "Multicommodity Distribution System Design by Benders Decomposition," Management Science, Vol 20, pp822-844.

73. *Geoffrion A.M. and Mac Bride R. [1978]*, "Lagrangian Relaxation Applied to Capacitated Facility Location Problems", AIIE transactions, Vol 10, pp40-47.

74. *Geoffrion A.M. and Marsten R.E. [1972]*, "Integer Programming Algorithms: A Framework and State-of-the-Art Survey", Management Science, Vol 18, No 9, pp 465-491.

75. *Giannessi F. and Nicoletti B. [1979]*, "The Crew Scheduling Problem: A Travelling Salesman Approach," in Combinational Optimization, Christofides N., Mingozzi A., Toth P. and Sandi C. editors, John Wiley & Sons.

76. *Gillet B. and Miller L.[1974]*, "A Heuristic Algorithm for the Vehicle Dispatch Problem," Operations Research, Vol 22, pp340-349.

77. *Glover F. [1968]*, "Surrogate Constraints", Operations Research, Vol 16, No 4, pp 741-749.

78. *Glover F. [1973]*, "Convexity Cuts and Cut Search," Operations Research, Vol 21, pp123-134.

79. *Gomory R.E. [1958]*, "An Algorithm for the Mixed Integer Problem,"

RM-2597, RAND Corporation.

80. *Gomory R.E. [1963],* "An Algorithm for Integer Solutions to Linear Programs," In Graves and Wolfe (Eds), Recent Advances in Mathematical Programming, pp269-302.

81. *Graham R., Lawler E., Lenstra J., and Rinnoy Kan A. [1970],* "Optimization and Approximization in Deterministic Sequential and Scheduling : a survey," Annals of Discrete Mathematics, Vol 5, pp287-326.

82. *Harary F. [1971],* "Graph Theory", Addison-Wesley.

83. *Hartmanis J. and Stearns R.E. [1965],* "On the Computational Complexity of Algorithm," Trans. Amer. Math. Soc., Vol 117, pp285-306.

84. *Held M. and Karp R.M. [1970],* "The Travelling Salesman Problem and Minimum Spanning Trees," Operations Research, Vol 18, pp1138-1162.

85. *Held M. and Karp R.M. [1971],* "The Traveling Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming, Vol 1, pp6-25.

86. *Held M., Wolfe P. and Crowder H.P. [1974],* "Validation of Subgradient Optimization", Mathematical Programming, Vol 6, pp 62-88.

87. *Hey A.[1980],* "Algorithms for the Set Covering Problem", Ph.D. Thesis, Department of Management Science, Imperial College, London.

88. *Hinson J. and S. Mulherkar [1975],* "Improvements to the Clarke and Wright Algorithm as Applied to an Airline Scheduling Problem," Technical Report, Federal Express Corporation.

89. *Hoffman A.J. and Kruskal J.B. [1956],* "Integral Boundary Points of Convex Polyhedra", in H.W.Kuhn and A.W. Tucker (eds), Linear Inequalities and Related Systems, Annals of Mathematics Studies, No 38, Princeton, N.J.

90. *Horowitz E. and Sahni S. [1978],* "Fundamentals of Computer Algorithms", Pitman.

91. *Iri M.[1966]*, "A Criterion for the Reducibility of a Linear Programming Problem to a Linear Network-flow Problem", RAAG Research Notes, Third Series, No 98.

92. *Jeroslow R.E. [1977]*, "Cutting-Plane Theory:Disjunctive Methods", Annals of Discrete Mathematicacs, Vol 1, pp293-339.

93. *Jeroslow R.E. [1979]*, "An Introduction to the Theory of Cutting-Planes", Annals of Discrete Mathematics, Vol 5: Discrete Optimization II, pp 71-95.

94. *Karp, R.M. [1972]*, "Reducibility among Combinatorial Problems," in R.E. Millet and J.W. Thatcher (eds), Complexity of Computer Computations, Plenum Press, New York, pp85-103.

95. *Karp, R.M. [1975]*, "On the Complexity of Combinatorial Problems," Networks , Vol 5, pp45-68.

96. *Klee V. and Minty G.J. [1972]*, "How Good is the Simplex Algorithm ?", in Inequalities III, O. Shisha Editor, Academic Press, pp 159-175.

97. *Klingman D. [1977]*, "Finding Equivalent Network Formulations for Constrained Network Formulations," Management Science, Vol 23, No. 7.

98. *Klingman D. and Russel R. [1975]*, "Solving Constrained Transportation Problems," Operations Research, Vol 23, No. 1.

99. *Krolak P. and Nelson J. [1972]*, "A Family of Truck Load Clustering Heuristics for Vehicle Routing Problems," Technical Report 78-2, Department of Computer Science, Vanderbilt University, Nashville, TN.

100. *Krolak P. and Williams M. [1978]*, "Computerized School Bus Routing," Technical Report 78-1, Department of Computer Science, Vanderbilt University, Nashville, TN.

101. *Land A. and Doig A.G. [1960]*, "An Automatic Method for Solving Discrete Programming Problems," Econometrica, Vol 28, pp497-520.

102. *Land A. and Powell S. [1960]*, "Computer Codes for Problems of Integer

Programming," Presented at the Conference on Discrete Optimization, Vancouver, B.C.

103. *Lawler E.L. and Wood D.E. [1960],* "Branch-and-Bound Methods: A Survey," Operations Research, Vol 14, pp699-719.

104. *Lasdon L.S. [1970],* "Optimization Theory for Large Systems", The Macmillan Company.

105. *Lenstra J.K. and Rinnoy Kan A.H.G. [1979],* "Computational Complexity of Discrete Optimization Problems", Annals of Discrete Mathematics, Vol 4: Discrete Optimization I, pp 121-140.

106. *Lenstra J.K. and Rinnoy Kan A.H.G. [1981],* "Complexity of Vehicle Routing and Scheduling Problems", in Proceedings of the International Workshop on Current and Future Directions in the Routing and Scheduling of Vehicles and Crews, Golden B.L. and Bodin L.D. editors, Networks, Vol 11, No 2, pp 221-228.

107. *Lin S.[1965],* "Computer Solution of the TSP," Bell System Tech. J., Vol 44, pp2245.

108. *Lin S. and Kernighan B.W. [1973],* "An Effective Heuristic Algorithm for the Travelling Salesman Problem," Operations Research, Vol 21, p498.

109. *Lucena A. [1986],* "Algorithms for the Vehicle Routing Problem," Ph.D Thesis, Department of Management Science, Imperial College, London.

110. *Maffioli F. [1979],* "The Complexity of Combinatorial Algorithms and the Challenge of Heuristics", Combinatorial Optimization, Christofides N., Mingozzi A., Toth P. and Sandi C. editors, John Wiley & Sons, pp 107-129.

111. *Marsten R.E. [1974],* "An Algorithm for Large Set-Partitioning Problems," Management Science , Vol 20, pp774-787.

112. *Marsten R.E. [1981],* "The Design of the XMP Linear Programming Library", ACM Transactions on Mathematical Software, Vol 7, No 4, pp 481-497.

113. *Marsten R.E., Muller M., and Killion C.[1978]*, "Crew Planning at Flying Tiger : A Successful Application of Integer Programming," Technical Report No. 553, Management Information Systems Dept., University of Arizona , Tucson, AZ.

114. *Marsten R.E. and Sheparsdon F. [1981]*, "Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications", in Proceedings of the International Workshop on Current and Future Directions in the Routing and Scheduling of Vehicles and Crews, Golden B.L. and Bodin L.D. editors, Networks, Vol 11, No 2, pp 165-177.

115. *Martin G.E. [1981]*, "Aircraft Scheduling Considered as an N-Task,M-Parallel Machine Problem with Start-Times and Deadlines," INFOR, Vol 19, No 2.

116. *Martin G.T. [1963]*, "An Accelerated Euclidean Algorithm for Integer Linear Programming," In Graves and Wolfe (Eds), Recent Advances in Mathematical Programming.

117. *Martin-Lof A.[1970]*, "A Branch and Bound Algorithm for Determining the Minimal Fleet size of a Transportation System," Transportation Science, Vol 4, pp159-163.

118. *Megiddo N. [1977]*, Private Communication . (A2.3)

119. *Meyer A.R., and L.J. Stockmeyer [1972]*, "The Equivalence problem for Regular Expressions with Squaring Requires Exponential Time," Proc. 13th Ann. Symp. on Switching and Automata Theory, IEEE Computer Society, Long Beach, CA, pp125-129.

120. *Mole R. and Johnson D. and Wells K. [1983]*, "Combinatorial Analysis of Route First-Cluster Second Vehicle Routing," Omega, Vol 11, pp507-512.

121. *Motzkin T. and Schoenberg I.J. [1954]*, "The Relaxation Method for Linear Inequalities," Canadian Journal of Math., Vol 6, pp393-404.

122. *Nemhauser G.L. [1966]*, "Introduction to Dynamic Programming", John Wiley & Sons , New York.

123. *Orlin J. [1982]*, "Minimizing the Number of Vehicles to Meet a Fixed Periodic Schedule : An Application of Periodic Posets," Operations Research, Vol 30, pp760-776.

124. *Owen G. [1973]*, "Cutting-Planes for Programs with Disjunctive Constraints," Journal of Optimization Theory and its Applications, Vol 11, pp49-55.

125. *Orloff S.C. [1976]*, "Route Constrained Fleet Scheduling", Transportation Science, Vol.10, No. 2, pp149-168.

126. *Orloff S.C. [1973]*, "Routing and Scheduling a Fleet of Vehicles:The school Bus Problem," unpublished dissertation, Cornell University, Ithaca, New York.

127. *Orloff S.C. [1974]*, "A Fundamental Problem in Vehicle Routing," Networks 4, pp35-64.

128. *Padberg M.W.[1975]*, "Characterisations of Totally Unimodular , Balanced and Perfect Matrices", in Roy B.(ed),Combinatorial Programming,Methods and Applications, pp275-284, D. Reidel Publishing Company, Dowrecht-Holland.

129. *Paixao J. [1983]*, "Algorithms for Large Scale Set Covering Problems", PhD Thesis, Department of Management Science, Imperial College, University of London.

130. *Roy B., Benayoun R., and Tergny J. [1970]*, "From S.E.P Procedure to the Mixed Ophelie Program," in J.Abadie (ed), Integer and Nonlinear Programming, American Elsevier, pp419-436.

131. *Rardin R.L. and Lin B.W. [1978]*, "What Makes Integer Programming Problems Hard to Solve: An Empirical Study", Opsearch, Vol 15, Nos 2 & 3, pp 65-77.

132. *Rubin D.[1973]*, "A Hybrid Cutting-Planes Enumeration Algorithm for Integer Programming", 43rd National Meeting of ORSA, Milwaukee, May 11th.

133. *Sandi C. [1976]*, "A Direct Method for Linear Inequalities," Presented at Euro II, Stockholm, Sweden, Nov.29-Dec. 1.

134. *Shapiro J. [1968]*, "Group Theoretic Algorithms for the Integer Programming Problem : Extension to a General Algorithm," Operations Research, Vol 16, pp928-947.

135. *Shapiro J.F. [1979b]*, "A Survey of Lagrangean Techniques for Discrete Optimization", Annals of Discrete Mathematics, Vol 5: Discrete Optimization II, pp113-138.

136. *Shepardson F. and Marsten R.E. [1980]*, "A Lagrangian Relaxation Algorithm for the Two Duty Period Scheduling Problem", Management Science, Vol 26,pp274-281.

137. *Smith B. and Wren A. [1981]*, "VAMPIRES and TASC: Two Successfully Applied Bus Scheduling Programs", in Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling, Wren A. editor, North Holland Publishing Company, pp 97-124.

138. *Spitzer M. [1961]*, "Solution to the Crew Scheduling Problem",AGIFORS Symposium.

139. *Taha H.A. [1975]*, "Integer Programming: Theory, Applications and Computations", Academic Press.

140. *Thiriez H.M. [1969]*, "Airline Crew Scheduling : a Group Theoretic Approach",
Flight Transportation Laboratory, Report R69-I, Department of Aeronautics and Astronautics, M.I.T., Cambridge, Mass.

141. *Thiriez H.M. [1971]*, "The Set Covering Problem : a Group Theoretic Approach", Revue Francaise d'Automatique,Informatique et Recherche Operationelle, Vol 3, pp83-104.

142. *Toregas C. and C. ReVelle [1972]*, "Location under Time or Distance Constraints," Papers of the Regional Science Association , Vol 28, pp133-143.

143. *Turing A.[1936]*, "On Computable Numbers,with an Application to the Entscheidungsproblem," Proc. London Math. Soc. Ser. 2 , Vol 42, pp230-265.

144. *Ward R., Durant P. and Hallman A. [1981]*, "A Problem Decomposition Approach to Scheduling the Drivers and Crews of Mass Transit Systems", in Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling, Wren A. editor, North Holland Publishing Company, pp 297-312.

145. *Werra D. [1981]*, "On Some Characterisations of Totally Unimodular Matrices", Mathematical Programming, Vol 20, pp14-21.

146. *Wolters J.[1979]*, "Minimizing the Number of Aircraft for a Transportation Network," European Journal of Operational Research, Vol 3, pp394-402.

147. *Wren A. [1981]*, "General Review of the Use of Computers in Scheduling Buses and their Crews," Computer Scheduling of Public Transport : Urban Passenger Vehicle and Crew Scheduling, A. Wren,ed,North-Holland Publishing Co., pp3-16.

148. *Young R.D. [1971]*, "Hypercylindrically-Deduced Cuts in Zero-One Integer Programs," Operations Research, Vol 19, pp1393-1405.

149. *Zadeh, N. [1973]*, "A Bad Network Problem for the Simplex Method and Other Minimum Cost Flow Algorithms," Math. Programming , Vol 5, pp255-266.

150. *Zwart P. [1972]*, "Intersection Cuts for Separable Programming," Washington University, St. Louis, January.