FLIGHT TRANSPORTATION LABORATORY REPORT R86-4

ROUTING PROBLEM WITH SERVICE CHOICES

by

BOON CHAI LEE

June 1986

FLIGHT TRANSPORTATION LABORATORY
REPORT R86-4

ROUTING PROBLEM WITH SERVICE CHOICES

BY: BOON CHAI LEE

ROUTING PROBLEM WITH SERVICE CHOICES

by

BOON CHAI LEE

ABSTRACT

This thesis finds solutions to the routing problem with service choices which is formulated as a capacitated minimum cost flow circulation problem with GUB constraints. The routing problem with service choices is solved using a specialized GUB branch and bound algorithm. Methods for node and GUB set selection are presented. A heuristic for finding good feasible solutions to initiate the branch and bound using vehicle size cuts is also derived. Furthermore, a network reduction scheme is formalized to reduce the size of the problem. This reduction is applied between pairs of nodes whose ground arcs have infinite upper-bounds. Initial experiments using the GUB branch and bound on several medium scale test problems appear promising. A variable tracking scheme which updates the status of the branching variables is included, which can be used to fully automate the branch and bound. This work supports the use of LP based GUB branch and bound for solving combinatorial problems with GUB constraints. Extensions to several related problems are also given.

Contents

Chapter 1

Introduction

In recent years, there has been an abundance of research into vehicle routing [27],[8] that has successfully utilized the techniques of mathematical programming and algorithmic designs for computer implementation. However, most of the problems attempted ignored the temporal restrictions placed on them in a practical setting. In this sense, vehicle routing problems with temporal constraints have received only scattered attention, focusing on very specialized problems.

The inherent complexity of vehicle routing, and vehicle routing over time in particular, places great limitations on the methodologies used and the size of the problems considered. Very often, vehicle routing problems are treated as substitutes for vehicle routing over time, whereby vehicle routes are first obtained, and the schedule is constructed from the routes found. Schedules are constructed based upon an improvement that seeks to improve inital routings using a hybrid of savings-type (Clark-Wright) approach, coupled with insertions, and/or intercharge heuristics[8]. The quality of this approach can be penalized by initial bad solutions.

The vehicle routing problem addressed in this dissertation makes no provision that vehicles have to start and end at the same depots. Thus, it considers multiple depots in the routings. Moreover, it incorporates temporal constraints in the services that are routed. In addition, the problems restrict the performance of at most one service out of each group

of services. This added restraint is why the problem is termed the routing problem with service choices.

The routing problem with service choices can be formulated as a network flow problem with side constraints. The network used is a schedule map composed from a set of given services. In its general form, it can be used to solve a variety of problems that will be described subsequently. In particular, the routing problem with time windows can be modelled as a special instance of the routing problem with service choices. This application will be pursued in depth since it arises in many practical situations. In most routing problems, where departure times are allowed to span across a certain time window, a shift in departure time enables a better utilization of vehicles. The proposed problem will attempt to find departure times for services in an optimal set of markets. The routing problem with service choices can also be used to model data communication transmission problems with precedence relationships. These precedent constraints parallel the temporal considerations imposed in routings on the schedule map.

The side constraints in the network based formulation constrict flow on bundles of service arcs to at most one. As a result, they are often referred to as GUB constraints. GUB(generalized upperbounding) constraints are also referred to as bundle constraints, multiple choice constraints or specially ordered set(SOS) constraints. The term GUB constraints will be used. Flows along services contained in each GUB constraint are further restricted to be integer. Such constraints appear in many formulations of practical problems, for instanace, multicommodity flow problems, resource allocation problems given fixed budgets, and transportation problems where GUB constraints are used to model vehicle capacities, route length or choices of vehicle type. Although the knapsack problem with multiple choice constraints has received some attention[1],[33], the same is not true for the capacitated network flow problem with GUB constraints.

Thus, the routing problem with service choices is both of theoretical and practical interest. The author is unaware of any work done on solving the capacitated minimum cost circulation problem with GUB constraints, whose variables are restricted to take on integer values. This thesis is an attempt to probe deeply into the structure of the routing problem with service choices, and to recommend an effective approach to be used to solve the problem optimally. The next section describes the routing problem with service choices.

1.1 Routing Problem With Service Choices

To motivate the description of the routing problem with service choices, the problem is cast as a special aircraft routing problem. Given a cyclic schedule which consists of a potential set of services and their expected benefits, partitioned into mutually exclusive subsets, find the set of services to be offered that maximize total benefits such that, at most, one service from each subset is used. In addition, find the number of vehicles required to perform these services.

Each Service is defined by 3 attributes:

1. Time and place of departure.
2. Time and place of arrival.
3. Expected benefit for performing the service.(By convention, negative and positive values refer to profit and expenses, respectively).

Only non-stop services are considered. The inclusion of multiple stop service only increases the size complexity, but not the structure of the problem. In some instances, like bulk or convoy type deliveries, the non-stop assumption appears reasonable. The assumption of a homogeneous fleet is acceptable in cases where the market to be serviced is well defined. For example, short haul trips may require vehicles of a specific size or type, or, the nature of the goods being transported may place restrictions on the type of vehicle used.

The unique feature of this problem is that the same vehicle is not constrained to leave and return to the same depot during the completion of a cyclic schedule. The only provision is that the schedule be cyclic, which is typical in most delivery problems.

The inputs to the problem consist of a cyclic timetable of potential flight serivces between stations. These are used to construct a schedule map, as shown in figure 1.1. The schedule map consists of a vertical time axis which represents stations. Each axis contains event nodes which indicate the arrival or departure time at a particular station. Event nodes are connected by three kinds of directed arcs which model aircraft flows. These three kinds of arcs are:

- Ground arcs, G, which join event nodes at the same station.
- Overnight or cycle arcs, K, which join the last event node of each station to the earliest event node at the same station.
- Service arcs, S, which join event nodes between stations.

Service arcs represent flight services leaving one station at a specific time and arriving at a ready time, which is the earliest time at which an aircraft is ready to depart.

In addition, there are capabilities associated with each arc x_{ij}, with lower bound l_{ij}, and upper bound u_{ij}. Typically, for service arcs $(i,j) \in S$, $l_{ij} = 0$, $u_{ij} = 1$; for ground and cycle arcs $(i,j) \in (G \cup K)$, $l_{ij} = 0$ and $u_{ij} =$ {maximum allowable number of aircraft at a particular station}. Associated with each arc (i,j) is an expected per unit cost for sending an aircraft along arc (i,j). For $(i,j) \in S, c_{ij}$ represents the expected benefit of performing service (i,j) if $c_{ij} \leq 0$, and the expense incurred for deadheading if $c_{ij} > 0$. For $(i,j) \in G$, c_{ij} denotes the per unit cost of idling an aircraft at a particular station. Finally, for $(i,j) \in K, c_{ij}$ represents the ownership costs of using an aircraft for the cycle period. Suppose that, $n_1, n_2, \cdots, n_v$ represents mutually exclusive subsets of service choices

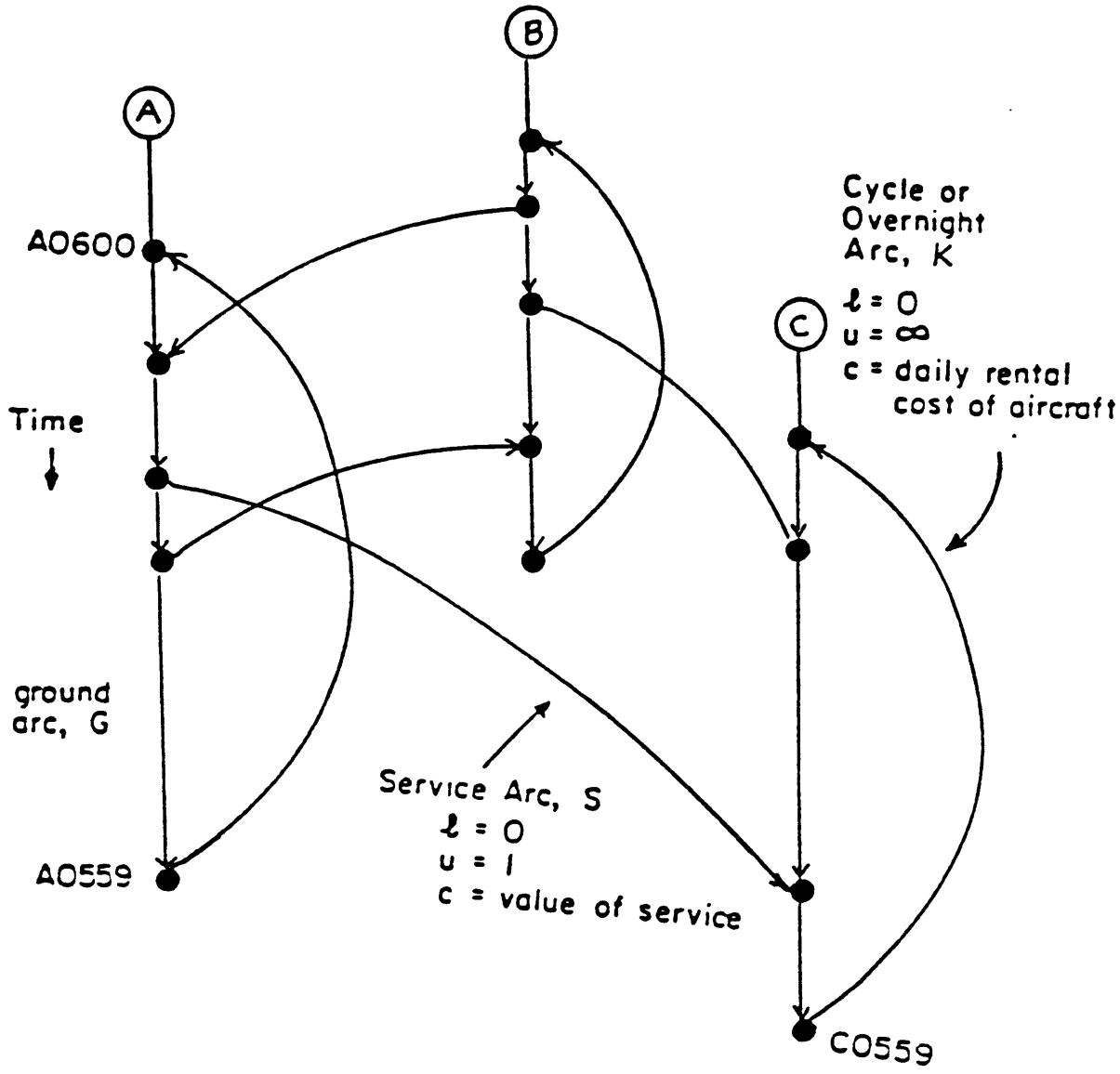


Figure 1.1 Schedule map

containing more than one element. Let $M = n_1 \cup n_2 \cdots \cup n_v$, where $M \subseteq S$. Now, if A and N are the arc set and node set, respectively, the routing problem with service choices can be formulated as follows:

$$
P \left\{
\begin{array}{lll}
\text{minimize} \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} & & \\
\displaystyle\sum_{i\in N} x_{ij} - \sum_{j\in N} x_{ji} = 0 & \forall\ j \in N & (1.1) \\
\displaystyle\sum_{(i,j)\in n_w} x_{ij} \leq 1 & \text{for } w = 1,\cdots,v & (1.2) \\
l_{ij} \leq x_{ij} \leq u_{ij} & \forall\ (i,j) \in K \cup G \cup (S-M) & (1.3) \\
x_{ij} \in \{0,1\} & \forall\ (i,j) \in n_w,\ w = 1,\cdots,v & (1.4)
\end{array}
\right.
$$

Constraint (1.1) is the conservation of flow constraints. (1.2) restricts the choices of services to, at most, one for each bundle. These constraints are called generalized upper bounding (GUB) constraints. Hereafter, the term GUB constraints is used. (1.3) are the capacity constraints on the arc not included in the GUB constraints (1.2). (1.4) ensures that a service is either chosen ($x_{ij} = 1$), or is not, ($x_{ij} = 0$). Note, that if (1.2) and (1.4) are satisfied, the remaining flows in A are integral since constraints (1.1) and (1.3) are totally unimodular. Moreover, (1.2) includes only subsets of service choices containing more than one element, that is $|n_w| > 1$, for $w = 1,2,\cdots,\ v$. For $(i,j) \in (S - M)$, constraint (1.3) takes the form $0 \leq x_{ij} \leq 1$. That is, for these arcs, the integrality restrictions (1.4) need not apply. Problem (P), when $M = \{0\}$, is known as the cyclic routing problem. This is a capacitated minimum cost circulation problem which can be solved using an out-of-kilter algorithm. Without loss of generality, assume that $l_{ij} = 0$ for $(i,j) \in (S - M)$. Observe that $x_{ij} = 0\ \forall\ (i,j) \in A$ is always a feasible solution to (P). When $l_{ij} \neq 0$ for some $(i,j) \in (S - M)$, there is no guarantee that (P) has a feasible solution. To ensure feasibility, the number of inbound service arcs should exceed outbound service arcs with

$l_{ij} \neq 0$, or the number of outbound service arcs should exceed inbound service arcs with $l_{ij} \neq 0$ at each city. If $x^*_{ij} \ \forall \ (i,j) \in A$ is the optimal solution to (P), then the number of aircraft used is given by

$$\sum_{(i,j)\in K} x^*_{ij}$$

Problem (P) can be extended to include the case where only Q aircraft are available by appending the constraint $\sum_{(i,j)\in K} x_{ij} \leq Q$. Note that the optimal solution to (P) consists of a set of profitable integral cycle flows. For each cycle flow, the sum of benefits along the service arcs must be greater than the costs of the ground and cycle arcs it includes. Since the ground arc costs are typically much smaller than other costs, this essentially means that a sequence of services is included only if the sum of its benefits exceed the operational cost of the aircraft used in that period. Moreover, in the optimal solution to (P), at least one ground arc from each city must contain a zero flow. Otherwise, a cycle flow consisting of cycle and ground arcs can be removed to improve the solution, since $c_{ij} \geq 0 \ \forall \ (i,j) \in K \cup G$.

The routing problem with service choices can be applied to many practical situations, a list of which are given below:

1. Modelling alternative pick-ups and deliveries based on availability coinciding at different times between pairs of cities.

2. Bundling arc flows can be used to enforce staffing or other budget requirements restricting the servicing of all arcs in a bundle.

3. Assignment of, at most, one item to any given activity or sets of activities.

4. Modelling time window constraints.

Modelling routing problems with time windows

The material that follows focuses on how the routing problem with time windows, can

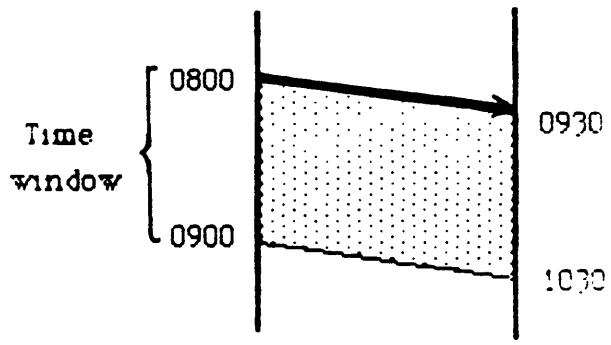


Figure 1.2 Service arc with time window

be modelled as a special instance of the routing problem with service choices. The time window for a service is the range of possible times for which the service can start. A given service and its associated time window is depicted in figure 1.2. The availability of flexible departure times for service results in better fleet utilization. For ease of exposition, the service arcs are labelled by integers. Consider the example given in figure 1.3. With no time windows for service, the schedule requires the use of two aircraft, assuming that all services result in profitable cycle flows. One cycle includes services 1 and 4, and the other 2 and 3. With the inclusion of time windows, only one aircraft is required. In general, it is not easy to identify such a profitable cycle. The problems created by the inclusion of the continuous time window are best illustrated by figure 1.4. The partial schedule shows that service 1 can connect to 2, and service 2 to service 3. However, this does not necessarily imply that service 1 can be connected with service 3 through services 1 and 2, because the earliest possible completion time for services 1 and 2 does not fall within the time window of service 3. This is referred to as the time window connectivity problem. To alleviate this problem, two formulations are presented.

Continuous time window model

The first includes the continuous time window given as follows. First, the original

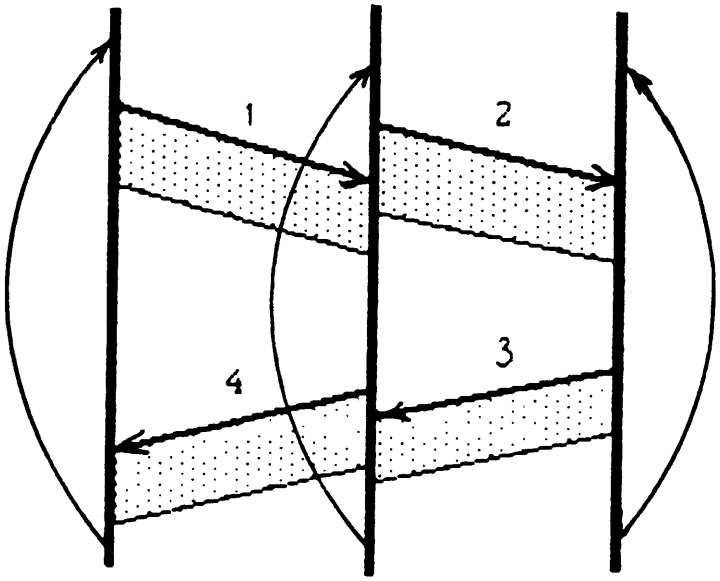


Figure 1.3 Routing with time window

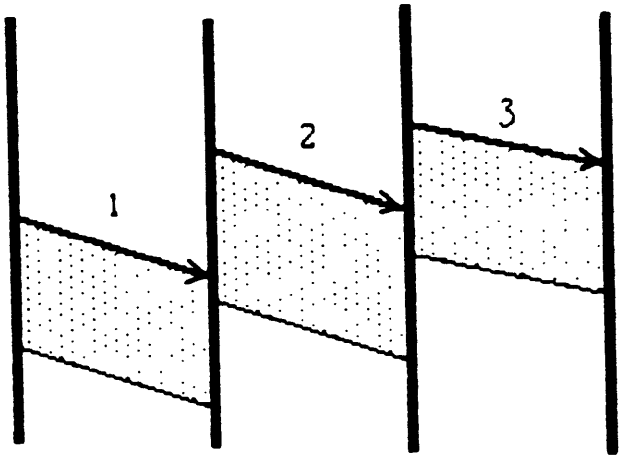


Figure 1.4 Time window connectivity problem

schedule map is transformed such that each service arc is replaced by a node. Denote this set of nodes by E. Node i refers to service i. Let t_i be the actual departure time for servcice i, and $[a_i, b_i]$ be its associated time window. Also let T_{ij} be the connecting time between service i and service j. T_{ij} is given by:

$$T_{ij} = \begin{cases} 0 & \text{if } a_j \leq a_i + BL_i \leq b_j \text{ or } a_j \leq b_i + BL_i \leq b_j \\ a_j - (b_i + BL_i) & \text{if } a_i + BL_i < a_j \\ \infty & \text{otherwise} \end{cases}$$

where BL_i = total time needed to perform service i.

BL_i includes the flying time and ground servicing time for service i. T_{ij} is zero if the completion time of sevice i falls within the time window of service j. The value of T_{ij} for the case where $a_i + BL_i < a_j$ is shown in figure 1.5.

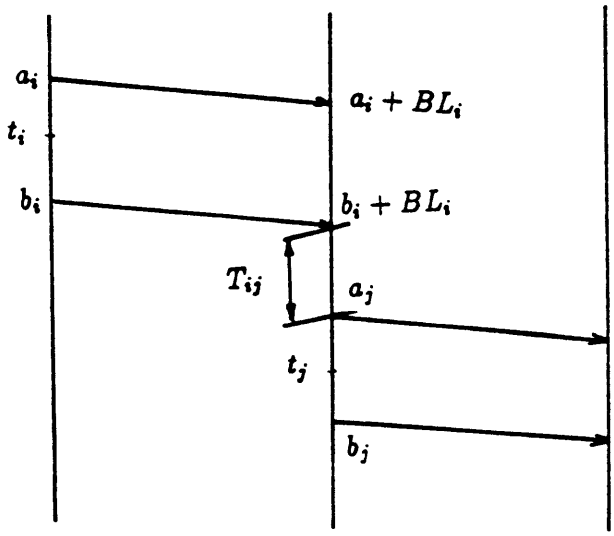


Figure 1.5 T_{ij} **for** $a_i + BL_i < a_j$

An arc connects node $i \in E$ to $j \in E$ if $a_i + T_{ij} \leq b_j$. Each cycle arc corresponding to a city A is replaced by a pair of nodes s_A, t_A; connected by an arc (t_A, s_A). Let $O = \{$ set of nodes $s_A \}$ and $D = \{$ set of nodes $t_A \}$. An arc (i, j), $i \in O$, $j \in E$ is included if service j departs from city i. Similarly, arc (i, j) for $j \in D, i \in E$ is included if service i terminates at city j. All arc upperbounds are set to 1 except those arcs connecting D to O, which corresponds

to the bounds on the cycle arcs in the schedule map. For all $j \in E$ c_{ij} = cost of performing service j. For $j \in D$, $c_{ij} = 0$, and c_{ij} = cycle arc costs for $i \in D$, $j \in O$. Figure 1.7 shows an example of a transformed network of the schedule map shown in figure 1.6, which contains time windows. If A and N denote the arc and node set of the transformed network, the routing problem with continuous time windows (PC), can be formulated as follows:

$$
(\text{PC}) \quad \begin{cases} \text{minimize} \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} & \\ \displaystyle\sum_{i\in N} x_{ij} - \sum_{j\in N} x_{ji} = 0 \qquad \forall\ j \in N & (1.5) \\ l_{ij} \le x_{ij} \le u_{ij} \quad \forall\ (i,j) \in A & (1.6) \\ x_{ij} > 0 \Rightarrow t_i + T_{ij} \le t_j \qquad \forall\ i,j \in E & (1.7) \\ a_i \le t_i \le b_i \qquad \forall\ i \in E & (1.8) \end{cases}
$$

Constraints (1.7) and (1.8) are scheduling or time related constraints which ensures that the time window connectivity problem is resolved. The decision variables are x_{ij}, $\forall\ (i,j) \in A$ and t_i, $\forall\ i \in E$. Notice that, t_i, $\forall\ i \in E$ are continuous variables. Constraint (1.7), as it stands, is non linear and can be replaced in linear form as follows:

Select L_{ij} such that $L_{ij} \ge b_i + T_{ij} - a_j$. Constraint (1.7) is then equivalent to:

$$
\forall\ (i,j) \in E \quad \begin{cases} t_i + T_{ij} - t_j \le (1 - x_{ij})L_{ij} \\ x_{ij} \in \{0,1\} \end{cases}
$$

To show this, assume that $x_{ij} > 0$, this implies that $x_{ij} = 1$ and the above is equivalent to (1.7). For $x_{ij} = 0$, the constraint above is trivially satisfied since $t_i + T_{ij} - t_j \le L_{ij}$.

Problem (PC) is thus a mixed integer programming problem. The simplified model presented assumes that ground arc costs are zero. In general, the magnitude of the idling or ground arc costs depend on the values of t_i for $i \in E$. Moreover, the value of T_{ij} is an optimistic approximation of the connect time between services i and j.

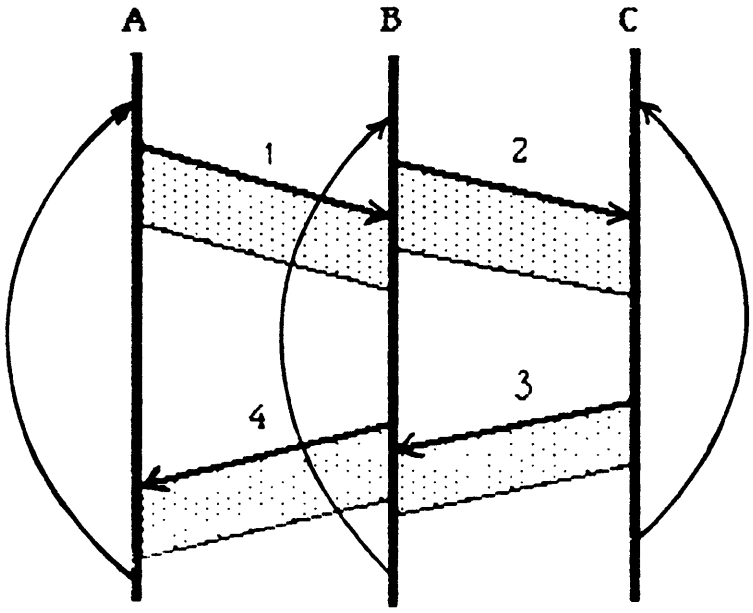


Figure 1.6 Network with time windows

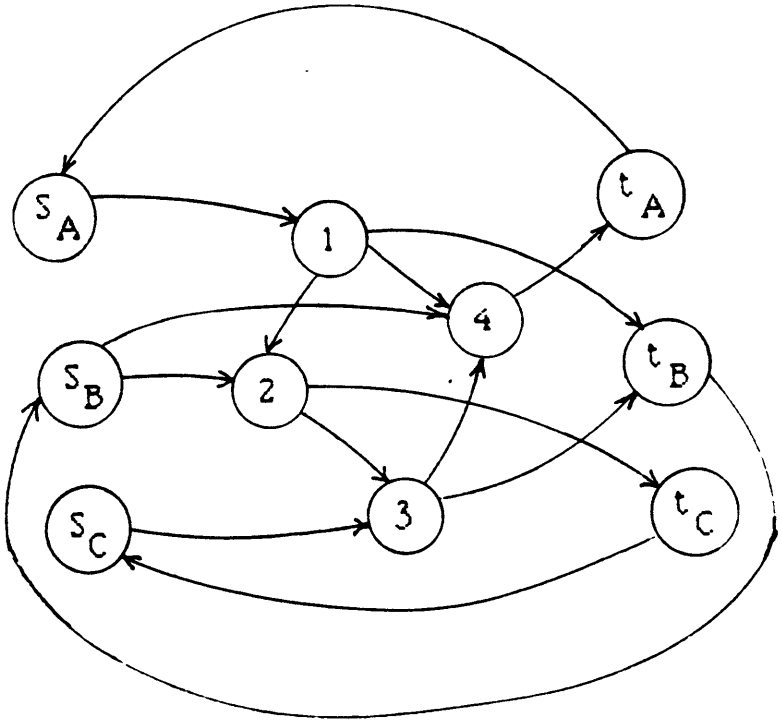


Figure 1.7 Transformed network of figure 1.6

Discrete time window model

Next, another model for the routing problem with time windows is presented. It turns out that in most practical situations, although the time windows given are continuous, there exists only a finite number of acceptable departure times within each time window. This can result, for instance, from slot allocations at airports. Moreover, the time required to perform a service includes the block time and ground servicing time which can vary depending on when the aircraft leaves and arrives. These problems, including others listed previously for routing with continuous time windows, are resolved by discretizing the departures within a given time window. This is shown in figure 1.8, where the time window is modelled by 3 departures. To ensure that at most one departure is used, a GUB constraint, $x_{12}+x_{34}+x_{56} \leq$

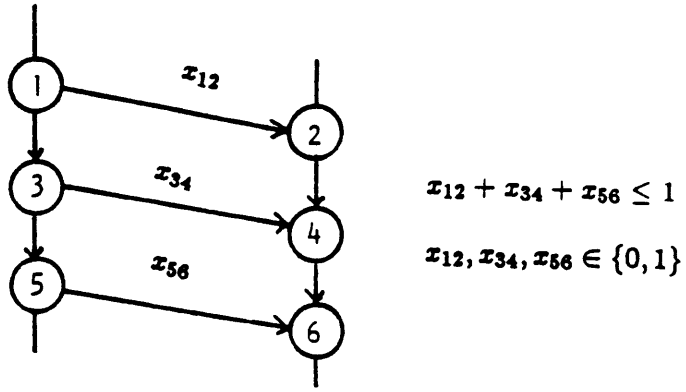


Figure 1.8 Discretizing the time window

1 is imposed, where $x_{12}, x_{34}, x_{56} \in \{0,1\}$. An additional feature of this model is that different benefits can be assigned to each arc in the bundle. For example, the later departures might offer slightly lower returns than the earlier ones.

This modelling of the routing problem with time window does not alter the formulation of (P). But, in this specific instance, the n_w for $w = 1,2,\cdots, v$ represents sets of service arcs corresponding to discrete departures within a time window for service i. Moreover, all arcs in a given set n_w for $w = 1,2,\cdots, v$ connect nodes belonging to similar city pairs.

The modelling of time windows by discrete departures increases the problem size of (P). At worst, the size of the problem is bounded as follows:

$$\text{Number of arcs} \leq 3 * \sum_{w=1}^{v} |n_w| + |K|$$

$$\text{Number of nodes} \leq 2 * \sum_{w=1}^{v} |n_w| + |K|$$

To get a flavor of the increase, assume that there are 500 services, modelled by 3 departures per service over 20 cities. The number of arcs generated is at most 4520 and the number of nodes is at least 3520. This means that the associated LP relaxation contains approximately 3520 rows and 4520 columns. This measures the worst case bound on the size of the problem (P). Fortunately, the temporal structure of the network allows for problem size reductions which are described in detail in Chapter 3. For the time being, it is adequate to note that if a service window corresponding to a service does not intercept any other service window, or is not intercepted itself, there is no reason to model it as a time bundle of service arcs. These and other observations related to the network topology allow for substantial reduction in the size of the problem.

1.2 Mixed integer programs with GUB constraints

Problem (P) belongs to the general class of network flow problems with side constraints. The side constraints are the GUB constraints which apply to arcs having bounds $[0, 1]$, and whose flows are restricted to be integral. Besides the routing problem with service choices, GUB constraints also appear in a wide variety of mixed integer programming problems. A list of problems that can be modelled within the framework of mixed integer programming problems with GUB constraints are shown next.

Consider the mixed integer programming problem with GUB constraints given as follows:

$$\text{MIPGUB} \begin{cases} \text{minimize} \quad \sum_{i=1}^{v}\sum_{j=1}^{n_i} c_{ij}x_{ij} & \\ \sum_{i=1}^{v}\sum_{j=1}^{n_i} a_{ij}x_{ij} \ (\leq\geq) \ b & (1.9) \\ \sum_{j=1}^{n_i} x_{ij} \leq 1 \qquad i = 1,2,\cdots,v & (1.10) \\ x_{ij} \in \{0,1\} \quad ; j = 1,2,\cdots,n_i; \ \text{for} \ i = 1,2,\cdots,v & (1.11) \end{cases}$$

Here, b and a_{ij} for all i,j are m component vectors $(b^1,b^2,\cdots,b^m)$ and $(a_{ij}^1,a_{ij}^2,\cdots,a_{ij}^m)$, respectively. The decision variables x_{ij} indicate which item j is selected for task i. Some problems that can be modelled as (MIPGUB) include:

A. Project Selection Problem.

The project selection problem can be stated as follows:

> Given some clusters of potential projects, and the expected returns and resource usages for each project, select at most one project from each cluster so as to maximize the total expected return.

For this case, the investment decision variables

$$x_{ij} = \begin{cases} 1 & \text{if project } j \text{ is chosen from cluster } i \\ 0 & \text{otherwise} \end{cases}$$

There are a total of $\sum_{i=1}^{v} |n_i|$ projects which are partitioned into v clusters $n_1,n_2,\cdots n_v$, from which at most one is to be selected. $b^k, k = 1,2,\cdots m$ refers to the amount of the k^{th} budget or resource. a_{ij}^k, $k = 1,2,\cdots m$ denotes the consumption of the k resource if project j is included in cluster i. c_{ij} estimates the potential gain from selecting project j in cluster i. Constraint (1.9) thus represents the resource or budget constraints, and (1.10) and (1.11) are the choice constraints. This problem is also referred to as the multidimensional knapsack problem with multiple choice constraints[1],[33]. If j refers to an item and i to a

schedule, the above problem is referred to as the multi-item scheduling problem[34]. It was also used to model menu-selections[3] and journal selection problem[18].

B. Spècial Crew Scheduling Problem

The special crew scheduling problem can be stated as follows:

> Given some set of rotations, each containing a specific home base, find the optimal assigmment of crew to rotation such that at most one crew can be assigned to each city base.

To model this problem, assume that $b^k = 1$, for $k = 1, 2, \cdots m$, corresponding to m flight segments. Let,

$$a_{ij}^k = \begin{cases} 1 & \text{if segment } k \text{ is included in rotation } j \\ & \text{containing city } i \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ij} = \begin{array}{l} \text{cost of assigning a crew to rotation } j \\ \text{staying at city } i. \end{array}$$

The decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if a crew is assigned to rotation } j \\ & \text{passing through city } i \\ 0 & \text{otherwise} \end{cases}$$

A total of v cities are considered and n_i for $i = 1, 2, \cdots v$, represents the set of rotations passing through city base i. Deadheading can be considered if constraint (1) takes on inequality $\geq$.

In addition to the above problems, GUB constraints appear in linearization of nonlinear functions[5], which convert an optimization problem with linear constraints and a nonlinear objective function into a mixed integer programming problem with linear objective functions and GUB constraints. Although these problems appear unrelated to the one addressed in this thesis, much insight can be gained from studying experiences in solving these specially structured problems, which closely relate to finding a solution strategy for (P). This thesis contributes, in part, to solving mixed integer programming problems with GUB constraints.

The rest of this thesis is organized as follows, Chaper 2 presents a brief survey of approaches to solving routing problems that most closely resemble those addressed in this dissertation. Since problem (P) falls within the class of mixed integer programming problems with GUB constraints, it is instructive to review some approaches used to solve such problems. In Chapter 3, initial experiments with decomposition techniques used to solve (P) are discussed, followed by a detailed description of a GUB branch and bound scheme to solve (P). Chapter 4, contains examples and results for several test problems applying the GUB branch and bound. It also includes a method of implementing the search aimed at fully automating the branch and bound. Finally, the last chapter summarizes the lessons gained from this exercise and recommends some extensions to certain related problems.

Chapter 2

Literature Survey

The previous chapter shows how the routing problem with service choices and time window constraints can be cast as a network flow problem with GUB constraints. Section 2.1 presents a brief review on the work done towards solving the routing problem with time window constraints. Since much insight can be drawn from experiences with solving mixed integer programming problems with GUB constraints, a survey of solution techniques dealing with GUB constraints is also included in Section 2.2. It is not known whether any literature exists which deals explicitly with the integral network flow problem with GUB constraints.

2.1 Routing problem with time windows

Solutions to the routing problem with time windows, as described in chapter 1, have proven elusive due to the integrality requirement. It has been modelled, in various forms, as large scale integer programs. In many cases, the LP relaxation was solved, and the solutions judiciously rounded to produce feasible solutions. In general, however, rounding off solutions to LP relaxation is unacceptable since they might not only be non-optimal, but infeasible as well.

A variant of this problem, known as the minimum fleet sizing problem was solved by Levin[26], where the routing problem with time windows is formulated as a bipartite maxi-

mum flow problem with GUB constraints. It was claimed, that for all large scale problems solved, an integer solution was always found from the LP relaxation. However, a Land and Doig branch and bound routine was included in case trouble arises. Levin[26] also showed how the minimum fleet sizing problem with time window constraints can be reformulated as a set covering problem. Unfortunately, this formulation incurs an explosion of integer variables. Benbasett[7], experimented further with different set covering formulations differing only in the choice of the decision variables. In particular, the columns can represent the set of service arcs, set of paths (which define sequences of connected services), or set of cycles (which define closed paths, given that the schedule is cyclic). For medium size problems, even the LP relaxation can contain thousands of rows, which makes it computationally prohibitive. For the routing problem with time windows as described in Chapter 1, the author has not always been fortunate in generating sufficiently large problems whose LP relaxation returns integral solutions, even though in most cases, the LP solutions are indeed integral.

Initial experiments were conducted by Simpson[32], Heldt[23], and Barkley[4], using the network flow formulation with GUB constraints to solve the routing problem with time window constraints. By ignoring the GUB constraints, or setting appropriate flow bounds on all variables in the GUB constraint, the problem can be effectively solved using an out-of-kilter algorithm. The strategy, then, was to search for the optimal solution by parametrically controlling the upper bounds on the arcs in each GUB set. Unfortunately, the rules for defining which arc in each GUB set to restrict in order to achieve good convergence to an optimal solution, remains difficult. Another approach is to use the Dantzig Wolfe decomposition, where the subproblem is a capacitated mimimum cost circulation problem. A price is attached to each GUB constraint until the adjoined constraints of the master problem are satisfied. If the optimal solution to the LP master problem is integer, then

the corresponding solution to the original problem is integer as well. Unfortunately, in most cases, decomposition results in an optimal mix of integer subproblem solutions whose combination need not be integer.

More recently, a new formulation of the minimum fleet sizing problem, with time window constraints, was proposed by Desrosier[12], whereby the departures within a time window are not discretized. The problem considers one depot, and minimizes the total travel time. A set of scheduling constraints are included to ensure that a sequence of services satisfies the time window constraints. The problem is solved using a column generation approach where the subproblem is a shortest path problem with time window constraints and a master set covering problem. At each iteration, a shortest path problem with time window is solved using dynamic programming, whose solution is used to generate a column for the master problem. The problem is applied to a school bus scheduling problem containing a single depot. In private communication, the subproblem solves quickly, since the number of services in each shortest path remains small. It is unclear, whether the dynamic programming algorithm for the special shortest path problem would be as efficient if the path selected at each iteration contains many services. Moreover, as in most column generation approaches, the solution is terminated upon reaching an acceptable bound.

The above presents a survey of problems most closely related to the routing problem with time window constraints that is addressed in this dissertation. To date, the author is not able to find any research done on the general routing problem with service choices. It is unclear that the structure of the routing problem is fully understood. It is, thus, the purpose of this dissertation to attempt to solve this problem, optimally, and in so doing, gain further insights into the nature of the problem. The lessons are then useful as the basis from which good heuristics can be derived.

Since the routing problem with service choices contains GUB constraints, it is instructive

to learn how other practitioners dealt with these structures, which appears in formulations of many practical problems. The following discussion reviews some approaches to a varied class of optimization problems with GUB constraints.

2.2 Survey of methods for integer programs with GUB constraints

Solutions of mixed integer programming problems with GUB constraints can be divided into 2 classes:

A) Cutting Plane Approach

B) Branch and Bound Approach

Cutting plane approach

The cutting plane approach was used by Healy[22], Glover[17], and Young[36]. Healy's approach solves a succession of modified LP problems, designed so that solutions of the corresponding LP converges to an optimal solution whereby only one variable is basic for each GUB set. The underlying idea is that there are, implicit in the updated LP matrix, a number of inequalities on the objective function, one for each GUB constraint, which bounds the value of the optimal solution. These objective cuts are obtained by estimating the bounds on the objective values, as the value of only one variable is increased to 1, and all others to zero, in each GUB constraint. By recursively tightening such inequalities, successive LP solutions can be driven to that choice of values for the 0-1 variables which is the least damaging to the objective function. It was shown that the objective cuts thus derived do not exclude any feasible solution. However, the convergence of this approach is not established, although it has worked well when implemented together with a prescribed terminating criteria.

Glover[17], derived a procedure that successively reduces the LP feasible region that

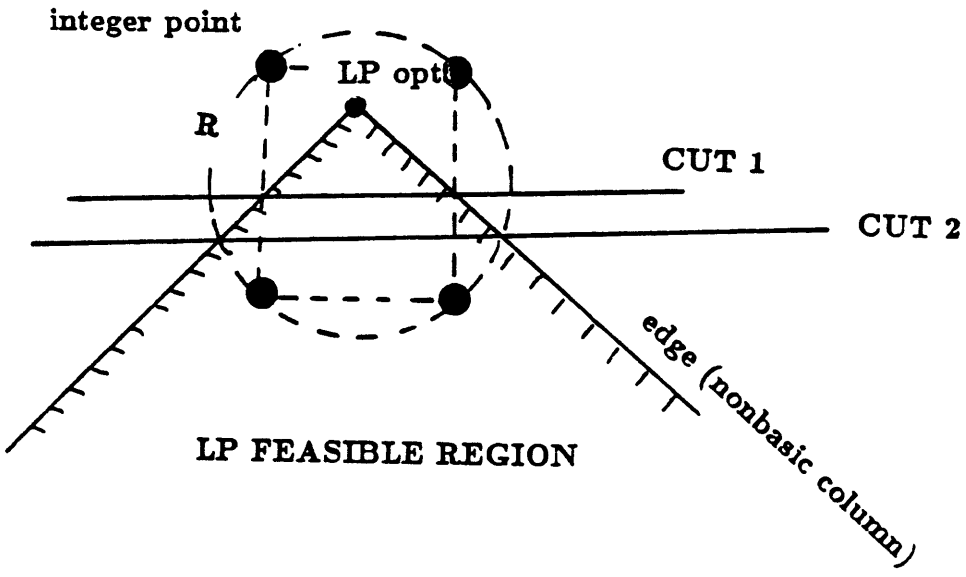


Figure 2.1 Convexity Cut

does not contain integral solutions in its vicinity. This method is an extension of the hyper-cylindrical cut procedure proposed by Young[36]. A convex region, R, is first constructed which intersects the LP feasible region, including the LP optimal point. A set of points is then derived from the intersection of R with the LP feasible region by moving along its edge from the LP optimal. These points form a plane that cuts off the LP optimal, and parts of the LP feasible region, without excluding any integral points. Such a plane is known as the convexity cut. Figure 2.1 illustrates the idea geometrically. In this example, R is a sphere that intersects all the integral points yielding Cut 2. Naturally, R could also be the convex hull of the integral points surrounding the LP optimal, yielding Cut 1. Note that Cut 2 is stronger than Cut 1.

The strength of the cut depends, therefore, on the region R. Glover describes how R is obtained from the GUB constraint.

As yet, there is little empirical evidence to suggest that the cutting plane approaches described converges satisfactorily for reasonably sized problems. Moreover, from an implementation viewpoint, the cut derivation can become computationally prohibitive.

Branch and Bound

Computationally, branch and bound approaches have fared much better in solving mixed integer programming problems. This is the approach adopted by Forrest[14], Gauthier[16], Mvert and Suhl[28], Bean[6], Sweeney and Murphy[34] and Glover and Mulvey[19], for solving various applications of mixed integer programming problems with GUB constraints.

Forrest performed extensive tests on solving mixed integer programs, and also provided a brief description on how to deal with GUB constraints based on ideas proposed by Beale and Tomlin[5]. These exercises were performed using UMPIRE, which has since been renamed SCIONIC. Gauthier[16] presents results for branch and bound using pseudo-costs, developed for the IBM MPSX system. Unfortunately, the problems dealt with, in both cases, are small and there is much indication that GUB branch and bound is still in its infant stages of development.

Mvert and Suhl[28] stress the need to look more closely at GUB branch and bound. The emphasis was to develop more efficient data handling of the problem, since it was observed that most of these problems either contain an enormous amount of constraints, but whose constraint matrix remains sparse, or relatively few constraints, with many integer variables and a dense constraint matrix. The data handling incorporates measures to perform logical testings during subproblem reoptimization.

Glover and Mulvey[19] presented an interesting solution technique aimed a solving network flow problems with GUB constraints. Each subproblem at a node corresponds to a transformed network flow problem. The solution of the transformed problem is optimal only if the arc flows are equal to their bounds. Since this is unlikely, a branch and bound

procedure is used to ensure that this condition is satisfied. Penalties for subproblem optimizations used in the branch and bound are also described. In general, the speed for solving each subproblem at a node is usually offset by the weaker bound obtained by solving a network flow relaxation. Moreover, the transformation, though clever, appears to require much pre and post processing. No computational experiences are provided.

Sweeney and Murphy[34], and Bean[6] use the idea of Lagrangian relaxation to derive and solve a restriction of the original problem, containing fewer GUB constraints and integer variables. For each GUB constraint, the minimum value of the reduced cost in a set is obtained from solving the LP relaxation which is used to guide the choices of variables to keep in the restricted problems. All variables in the same GUB set having reduced costs that do not fall within a prescribed bound of the minimum reduced cost value are then removed. The resultant problem remains a mixed integer program, though smaller. Experiments show that for large scale multi-item scheduling problem, solving the restricted problem is often sufficient to yield acceptable solutions. Each candidate subproblem corresponding to a node is solved as an LP.

Chang[9] presented a heuristic for solving this class of problem based on the pivot and complement idea of Balas and Martin[2]. The key idea of pivot and complement is to observe that the restriction that variables must take on values of 0 or 1 is equivalent to imposing that the slacks in the coupling constraints(which are constraints containing variables belonging to different GUB constraints) remain basic. If such is the case, the integer variables, being non basic, must have values equal to their integral bounds of 0 or 1, depending on their reduced costs. Chang[9] extended this idea to insist that in any feasible solution, only one variable in each GUB constraint of the equality type must be nonzero. Initial experiments indicate that such an emphasis is promising. However, the penalty on the optimality of such a heuristic solution remains uncertain.

In summary, the branch and bound method appears to offer the best potential for solving the routing problem with service choices. Although many previous attempts have been made to solve specific applications and generalizations of mixed integer programming problems, with GUB constraints, every indication suggests that the branch and bound approach used is still in its initial stages of development. Perhaps, this is because the major emphasis thus far has been on solving the LP relaxations to the optimization problems, coupled by judicious pertubation of the solution to integrality. In many problems this is acceptable since the coupling constraints are soft. Unfortunately, it is unclear that the routing problem with service choices exhibit such properties. In the next chapter, a GUB branch and bound scheme is presented to solve the problems described.

Chapter 3

Solution strategy for the routing problem with service choices

The formulation for the routing problem with service choices includes network structures with imbeded generalized upper bound constraints of the inequality type. The initial attempt is to exploit the network based structures through the use of Lagrangian and Benders decomposition. In both cases, the subproblem reduces to a capacitated minimum cost circulation problem which can be solved efficiently. Unfortunately, both procedures showed poor convergence. In section 3.1, an outine of experiences using these two approaches is summarized.

It turns out that the routing problem with service choices, (P) solved as an LP produces integer solution in most cases. A major difficulty is generating problems which fail to yield integer solutions to the LP relaxation. This observation prompted the use of a specialized branch and bound search for the optimal solution. The main difference in this branch and bound scheme is that branching is based on each GUB set as opposed to individual variables. A detail description of the procedure follows in section 3.2.

Section 3.3 proposes ways in which the branch and bound can be enhanced. Specifically, a network aggregation scheme is derived in order to reduce the size of the problem (P) and conceivably the number of GUB sets and integer variables. The reduction scheme is shown

to be optimal if the upper bounds on the non service arcs for each city, do not vary. The LP at each node can be solved more efficiently if the the GUB constraints are included implicitly. In appendix B, a method to include the GUB constraints implicitly in the LP using similar ideas proposed by Dantzig and Van Slyke[11] is included. Extensions based on similar ideas can be found in Lasdon[24] and Chen and Saigal[10]. To prune the search, a method for finding feasible solutions is described. This method resolves the LP parametrically by varying the vehicle size. At each node, an objective cut is imposed whenever the objective values of the remaining candidate problems, corresponding to unprocessed nodes, lies close to that of the best feasible solution obtained.

To simplify the discussions which follow, the routing problem with service choices is presented in a condensed matrix form (P) as follows:

$$
P \left\{ \begin{array}{rl}
\text{minimize } C_v X_v + C_s X_s & (3.1) \\
N_v X_v + N_s X_s = 0 & (3.2) \\
B X_s \leq 1 & (3.3) \\
0 \leq X_v \leq U_v & (3.4) \\
X_s \in \{0,1\} & (3.5)
\end{array} \right.
$$

where (N_v,N_s) is the node arc incidence matrix, and (3.2) is the conservation of flow constraints. X_s is the set of all bundle arcs. $(X_v \cup X_s)$ is the set of all arcs with $(X_v \cap X_s = 0)$. Constraint (3.3) denotes the set of GUB constraints of the ineaquality type. Note the B is a matrix with at least two ones in each row and a single one along each column. U_v refers to the upper bounds on X_v. 1 refers to a vector of ones, and $z(\bullet)$ equals to the objective value of solving problem $\bullet$. The LP relaxation of (P), (P^0), is the problem (P) with the constraints $X_s \in \{0,1\}$ replaced by $X_s \geq 0$. Constraint (3.3) ensures that $X_s \leq 1$. In this, and all subsequent chapters, x_i is used to refer to flows along arc $i \in (N_v, N_s)$.

3.1 Limitations of Lagrangian and Benders Decomposition

This section reviews work done on the routing problem with service choices using Lagrangian and Benders decomposition. These approaches were tried since they exploit the imbedded network structure of the problem. Although successful in solving some large scale problems[13], the convergence of both procedures is unsatisfactory when applied to the routing problem with service choices. In what follows, both procedures are briefly described[1], and problem areas identified.

Lagrangian Relaxation

Observe that (P) with constraint (3.3) relaxed, is a capacitated minimum cost circulation problem, since constraint (3.5) can be replaced by $0 \leq X_s \leq 1$ without affecting the solution. The method attempts to find solutions to (P) by solving the above relaxation. However, a solution to the relaxed problem need not be feasible, since the constraint (3.3) can be violated. A penalty, incorporated in the objective, is used to discourage violation of the relaxed constraints. A series of relaxations, using different penalties are solved, until a good feasible solution is found. This is the emphasis of Lagrangian relaxation, so called, since the subproblem is not strictly a relaxation of the original problem.

A price vector $W \geq 0$ corresponding to constraint (3.3) is selected to form the Lagrangian subproblem $(L(W))$ defined as follows:

$$L(W) \begin{cases} \text{minimize } C_v X_v + C_s X_s + W(BX_s - 1) \\ = -W\ 1 + \text{minimize } C_v X_v + (C_s + W\ B) X_s \\ \text{subject to:} \\ \text{Constraints (3.2), (3.4), and } 0 \leq X_s \leq 1 \end{cases} \tag{3.6}$$

Equations (3.6) are the network flow constraints. The term $W(BX_s - 1)$ measures the

[1]see appendices C and D for detail developments

penalty incurred for relaxing (3.3). Since $W(BX_s - 1) \leq 0$, $z(L(W)) \leq z(P)$ for all $W \geq 0$. Thus, it is preferable to select $W^* \geq 0$ such that

$$DL \left\{ \quad z(L(W^*)) = \max_{W \geq 0} z(L(W)) \right.$$

This is the Lagrangian master(dual) problem. Solution in $L(W)$ though integral, is not necessarily feasible in (P). A solution in $L(W^*)$ which is feasible in (P), is optimal only if $W^*(BX_s - 1) = 0$, where (X_v^*, X_s^*) is the solution of $L(W^*)$. In this case, the objective function in $L(W^*)$ corresponds to (P). This is sometimes called the global optimality condition[31]. If such is not the case, $(z(P) - z(DL)) > 0$, and this is known as the duality gap. This means that even if the dual problem (DL) is solved, there might be a gap which results from the discreteness of (P).

At iteration k, the procedure selects a $W^k \geq 0$ and solves $L(W^k)$. The solution (X_v^k, X_s^k) is used to form an objective cut for the reduced master problem (SDL), which is solved to update W^k. Problem (SDL) contains only a subset of constraints in (DL). This is then used to modify the objective of $L(W)$, which is resolved. The procedure iterates between solving $L(W)$ and (SDL) until $z(SDL) = z(L(W^k))$ for some k. For large problems, W^k is updated using subgradient optimization[13] in lieu of the method just described.

It turns out that W^* is equal to the optimal dual prices associated with the GUB constraints (3.3) when the LP relaxation of (P) is solved.[2] This is referred to as the integrality property and arises whenever the Lagrangian subproblem can be solved as an LP. Moreover, $z(P^0) = z(L(W^*)) = z(DL)$, even though the solution to (P^0), which is usually fractional, might be different from the integral solution of $L(W)$. In terms of objective values then, solving (P) via Lagrangian relaxation offers no advantage over solving (P^0).

Now, it is described how Lagrangian relaxation was implemented to solve the routing problem with service choices. The value of W was updated using subgradient optimization.

[2]See appendix C

Starting at an initial value W^0, a sequence $\{W^k\}$ is generated by the rule,

$$W^{k+1} = W^k + t_k(BX_s^k - 1)$$

where X_s^k is the optimal solution to $L(W^k)$, and t_k is a positive step size given by,

$$t_k = \frac{\lambda_k(z^{up} - z(W^k))}{\|BX_s^k - 1\|^2}$$

where λ_k is a scalar satisfying $0 \leq \lambda_k \leq 2$, and $z^{up} \geq z(DL)$.[3] A value of $\lambda_0 = 2$ is selected, and λ_k is halved whenever $z(DL)$ failed to improve after some fixed iterations.

Although it is convienient to select $W^0 = 0$, W^0 is chosen as follows: as follows:

Let C_s^i denote the cost vector associates with the variables of the i^{th} GUB constraint. Then, the i^{th} component of W_0 is

$$(W_0)^i = |\min_{\forall c_{ij} \in C_s^i} c_{ij}|$$

Next, let z^{up} be the value of a feasible solution in (P). Since $z^{up} \geq z(P^0)$, the implies that $z^{up} \geq z(DL)$ since $z(DL) = z(P^0)$. To obtain a feasible solution, select one arc from each GUB set having the greatest contribution, and all the service arcs not belonging to any bundle, and set the flows to one. To avoid infeasibilities, a check is made to ensure that the sum of flows entering each city equals the flow leaving the same city. This is done by resetting flows along some service arcs to zero. In most cases, this is not difficult to enforce, and a feasible solution can be obtained quite easily. However, it is not possible to determine the quality of the feasible solution so derived. Should the objective value of the feasible solution be greater or equals to zero, the arcs are then judiciously interchanged to obtain an improved solution.

Results from initial tests using Lagrangian relaxation, reveal that:

[3]The vector $(BX_s^k - 1)$ is known as the subgradient.

1. $z(L(W))$ does not converge satisfactorily, and appears erratic. Moreover, it is difficult to know whether the subgradient has converged correctly. This means that the hope of obtaining $z(DL)$ appears slim.

2. The price vector W^k either produces infeasible solution to (P) because it under penalizes the relaxation of (3.3) or, $(L(W^k))$ yields a flow of zero when W^k is strengthened. It is rather difficult to find the correct combination of weights that improve the solution to $L(W)$.

Very often subgradient optimization is terminated prematurely whenever the solutions corresponding to $L(W)$ and (DL) seem to tail off. It is conceivable that $z(DL)$ might never be obtained. Furthermore, when the optimal dual price, W^*, obtained from solving (P^0) is used to solve $L(W^*)$, a flow value of zero is usually obtained. This occurs because in most cases $z(P^0) = -1W^*$. Even if $z(DL)$ is found, the duality gap remains to be resolved. Resolution of duality gap forces the incorporation of methods that address the combinatorial property of the problem in order to find the optimal solution. This issue is deferred because it can be better addressed directly, as will be shown in section 3.2.

It is believed that Lagrangian relaxation cannot be effectly applied to problems which exhibit the integrality property. For problems without the integrality property, the value of the LP relaxation provides a lower bound for the Lagrangian subproblem(which is typically a combinatorial problem). The effectiveness of using Lagrangian relaxation lies heavily in the ability to find good solution to the combinatorial subproblem efficiently. It is only then, that a bound better than that of the LP relaxation can be found. A list of problems which confirm this observation is available in Fisher[13].

Benders Decomposition

Lagrangian relaxation decomposes the problem row-wise. It is sometimes referred to

as price directive decomposition. Consider next the decomposition of (P) column-wise, sometimes known as resource directive decomposition. Given an allocation of flows on some arcs (corresponding to allocating resources), which renders the problem (P) easy to solve, the remaining flow patterns are found by solving the resultant subproblem. Note that the subproblem need not be feasible. In any event, the result obtained from solving the subproblem provides an objective bounding cut, called Benders cut, on the solutions of (P), which is solved to obtain another set of flow allocations. The process continues until the solution yielding the best allocation is found. This is the focus of Benders decomposition[4], formalized next in the context of (P), followed by a discussion of some experiments relating to its use.

Let $Q = \{X_s | BX_s \leq 1 \text{ and } X_s \in \{0,1\}\}$. Select $X_p \in Q$ and determine the remaining flow in (P) by solving:

$$R(X_p) \left\{ \begin{array}{ll} & C_s X_p + \text{minimize } C_v X_v \quad (3.7) \\ \text{subject to:} & \\ & N_v X_v = -N_s X_p \quad (3.8) \\ & 0 \leq X_v \leq U_v \quad (3.9) \end{array} \right.$$

$R(X_p)$ is referred to as the Benders subproblem, and can be solved as a capacitated minimum cost flow problem by imposing a flow of X_p on X_s. However, $R(X_p)$ need not be feasible. This can occur when the flow selected in X_p does not satisfy continuity conditions at each city. Theoretically, as shown in appendix D, this possess no difficulty. Should the solution to $R(X_p)$ be feasible, it is also feasible in (P) since $R(X_p)$ is a restricted problem of (P). As such, $z(P) \leq z(R(X_p))$ for all $X_p \in Q$. Thus it is ideal to select X_q which solves the following problem:

$$RM \left\{ \quad z(R(X_q)) = \min_{X_p \in Q} z(R(X_p)) \right.$$

[4]A thourough description is given in appendix D

(RM) is known as the Benders master problem. It is an integer programming problem(with one continuous variable) because of the constraints $X_p \in Q$. Note that $z(P) = z(RM)$. The procedure begins by selecting $X_p \in Q$, and solving $R(X_p)$. The solution to $R(X_p)$, is used to derive a Benders cut, which is appended to (SRM), which is a relaxation of (RM) containing a subset of its constraints. Thus, $z(SRM) \leq z(RM)$ The enhanced problem (SRM) is then solved to obtain an updated X_p, which is used to resolve $R(X_p)$. This procedure iterates between solving $R(X_p)$ and (SRM). Unfortunately, the size of (SRM) grows rapidly and thus its solution time increases dramatically.

In this work, the LP relaxation of (SRM) is solved. Initially, the solution is rounded to satisfy the constraints $X_p \in Q$. The variable in each GUB constraint having the smallest integer infeasibility is set to 1. The integer of infeasibility for each variable is equal to its residual value as it is forced to zero or one. The reason for doing so is that the feasibility of (SRM) becomes increasingly difficult to verify as its size increases. In addition, some flows in X_p are reduced to zero to render R(X_p) feasible. Feasibility in R(X_p) is checked by ensuring that flow conservation at each city is preserved. Such an X_p is chosen, to provide feasible solutions as the method progresses. To prevent cycling, a different X_p is derived during each pair of optimizations.

Tests based on the above procedure yield the following observations:

1. The values of $R(X_p)$ converge poorly and, in many cases, either do not improve or return objective values greater than zero.

2. As the size of (SRM) increases it becomes increasingly time consuming to find a feasible solution for $R(X_p)$. Moreover, the number of simplex iterations required to reoptimized the restricted master problem increases.

Since $z(SRM)$ is a lower bound of $z(P)$, a feasible solution obtained from $R(X_p)$ is

within ϵ of the optimal, where

$$\epsilon = \frac{z(R(X_p)) - z(SRM)}{z(SRM)}$$

Both experiments with Lagrangian and Benders decomposition, showed unsatisfactory convergence. In order to solve (P) optimally using Lagrangian relaxation, a method has to be incorporated to resolve the duality gap, which results from the discrete property of (P). With Benders decomposition, effective implementation depends heavily on finding dominating or strong cuts, perhaps from the structure of (P) or its solutions, that makes it possible to solve only a few master problems. It is believed however, that the basic issue pertaining to the efficient solution of (P) lies in how well the GUB constraints are handled. Fortunately, the LP relaxation, (P^0) of (P) yields integer solutions in many cases, and in the next section it is described how this observation and the structure of the GUB constraints can be combined to produce a special branch and bound that offers much potential for addressing the optimality of (P).

3.2 Branch and bound on problems with GUB constraints

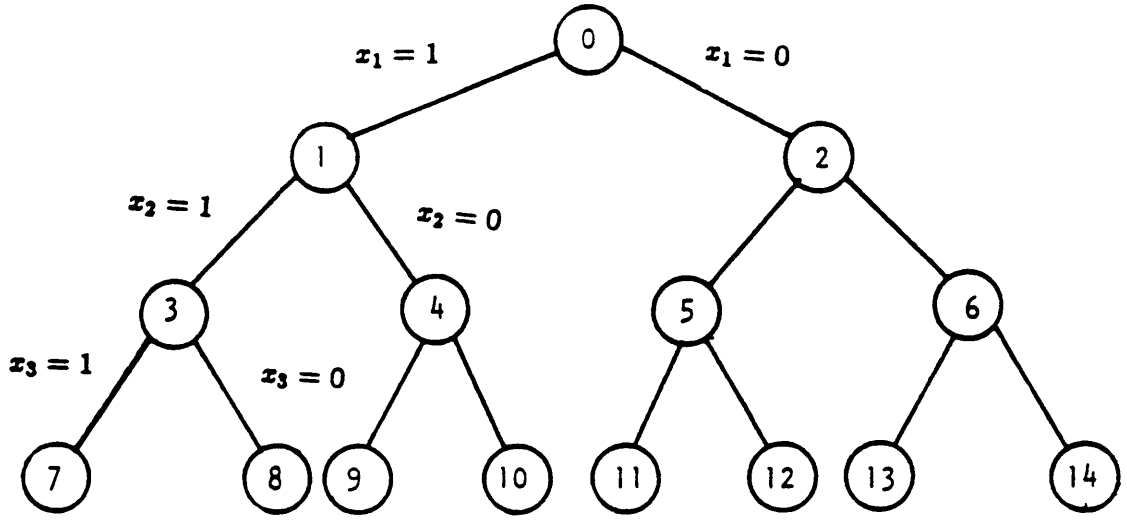
The previous section shows how Lagrangian and Benders decomposition is applied to the routing problem with service choices. In order to resolve the duality gap and solve the integer Benders master problem with GUB constraints, the discrete property of (P) needs to be addressed. This section shows how the integrality of (P) can be effectively dealt with. Specifically, a branch and bound scheme that utilizes the structure of the GUB constraints is presented. A comparison is made between three different tree structures. The choice to use branch and bound arises because, in most cases, (P^0) returns integral solutions and thus solves (P) optimally. In fact, generating solutions that are non integer when (P) is solved, remains a difficult problem. Although not handled implicitly in network flow programs, efficient solutions to LP exist which incorporate the GUB constraints implicitly.[5]

A review for the general branch and bound is given in Appendix A. Terms used in the descriptions that follow are also found in Appendix A. In this section, several choices of the branch and bound tree are presented. It is shown how the existence of the GUB constraint is instrumental in influencing the tree selection.

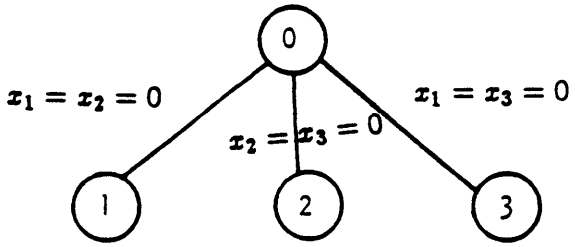
Consider three branch and bound trees, as shown in figure 3.1.

The trees are labelled as A, B, and C. For a given tree J, let $J(i)$ denote node i of tree J. For simplicity, assume that there is only one GUB set: $x_1 + x_2 + x_3 \leq 1$. Trees A and C are binary trees, whereas tree B is a variable tree. Tree A is used most commonly in single variable branching. Trees B and C consider implicitly the GUB constraints. They rely on the fact that, at most, one variable is basic at the optimal solution. For example, node B(1) corresponds to solving (P) with constraint $x_3 \leq 1$. Similarly, B(2), B(3) corresponds to imposing $x_1 \leq 1$ and $x_2 \leq 1$, respectively. Next, consider tree C. C(1) represents the subproblem with constraint $x_2 + x_3 \leq 1$ or $x_1 = 0$. C(2) considers the constraints $x_1 \leq 1$

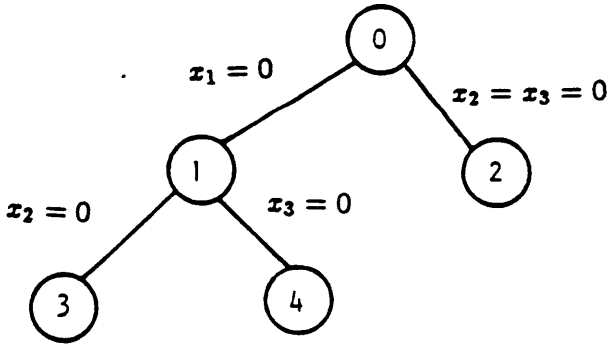
[5]see appendix B



Tree A



Tree B



Tree C

Figure 3.1 Three possible tree structures for GUB branch and bound

and C(3) and C(4) explore the cases $x_3 \leq 1$ and $x_2 \leq 1$, respectively. Observe that nodes C(2), C(3) and C(4) correspond to B(2), B(1) and B(3). These terminal nodes exhaustively consider all feasible solutions in (P). Although the constraints imposed at the terminal nodes are $x_i \leq 1$ for all i, and of the inequality type, the solution is guaranteed to be integer, since the optimization problem at each terminal node is a capacitated minimum cost network circulation problem.

Notice that tree A contains $2^3 = 8$ terminal nodes, whereas both tree B and C have 3 terminal nodes. In general, if there are n 0,1 variables, with v GUB sets each containing $n_1, n_2, \cdots, n_v$ variables where $\sum_{w=1}^{v} n_w = n$ then tree A contains 2^n leaf nodes, whereas trees B and C contain $\prod_{w=1}^{v} n_w$ nodes. For large n, $2^n \gg \prod_{w=1}^{v} n_w$ nodes. Take $n = 50$, and $|n_w| = 2$ for all i and $v = 25$, then $2^n \approx 1.3 * 10^{15}$, whereas $2^{25} \approx 3.6 * 10^7$. Now if $|n_w| = 5 \ \forall \ w$ and $v = 10$, the number of terminal nodes is $5^{10} \approx 9 * 10^6$. These analyses show the worst case consideration for using any of the trees and is tantamount to a complete enumeration of the solutions. It is not used as a yardstick to determine preference of one tree structure over another. If there is any reason to suppose that the complete tree needs to be processed, then it is inadvisable to use branch and bound. What is a more critical criteria for tree selection is the expected number of nodes that will be generated in the process of solving (P). Most general purpose codes use tree A, and a last-in-first-out (LIFO) scanning of nodes that probes deep into the tree in order to find a feasible solution, leaving many nodes to be fathomed even if a feasible solution found in the process is optimal.

However, what is more damaging, is that the single variable branch and bound fails to restrict the problem enough to force integral solutions, especially for degenerate problems. At the opposite extreme, tree B forces too many variables at a single branch to zero. This is done at the expense of incurring more optimization problems at each level corresponding to a GUB constraint. As a comparison between trees B and C, note that even though both

trees have 3 terminal nodes, C(1) might be fathomed and thus C(3) and C(4) need not be evaluated. The issue then is how likely can nodes in lower levels of tree C be fathomed. What appears promising for both trees B and C is that as the branch and bound progresses, the feasibility of the GUB constraints are enforced incrementally. The choice of tree C allows the determining of the partitions for the GUB variables into 2 mutually exclusive subsets at each pair of branching. The size of these subsets can be controlled so that the resultant subproblems produced by the branching provide sufficient improvements. Moreover, at each live node, there are sufficient flexibilities to select any infeasible GUB set to effect branching. Ideally, a particular GUB set is chosen whose partition penalizes the objective the least, but provides enough restrictions to return feasible solutions. This is, of course, an issue that needs to be addressed in more detail later. It suffices to note for the moment, that tree C affords more flexibility. Moreover, since the LP relaxation (P^0) has strong tendencies to solve to integrality, it is believed that the chances of getting feasible solutions at subproblem optimization remain favorable. Furthermore, from an implementation viewpoint, the binary branching structure of tree C, is much easier to implement. This is especially so when n_w are different sizes, so that tree B has a different number of branches at each level eminating from each node. All these observations encourage the use of tree C over B or A as the branch and bound tree.

Relations between different tree structures

Before describing the details of the GUB branch and bound using tree C, it is shown that tree C can be viewed as a generalization of trees A and B.

Consider first the relationship between trees C and A. First, rewrite the GUB constraint as an equality by including a slack to produce $x_1 + x_2 + x_3 + Slack = 1$, $Slack \geq 0$. Then, the first level of A corresponds to the first level of C if the variables are partitioned by setting first $x_1 = 0$, then $x_2 = x_3 = Slack = 0$. Note that, due to the special structure of

(P), the number of integer variables remains at 3.

The relationship between trees C and B is not as direct. Observe that $S^{C(2)} = S^{B(2)}$, and $S^{C(3)} = S^{B(1)}$ and $S^{C(4)} = S^{B(3)}$, where S^i denotes the optimization problem at node i.[6] That is, there is a one to one correspondence between the terminal nodes of C and B. Thus, when each GUB constraint contains 2 integer variables, tree C corresponds to B. In general, Tree C includes a series of more subproblems along the path from node 0.

GUB Branch and Bound

Having selected the tree structure C to use, the rules for node and GUB set selection are now described. The efficiency of any branch and bound depends on the proper selection of which node (subproblem) to evaluate next, and which GUB set to choose for branching. Unfortunately, there is no optimal rule for node or set selection. In most cases, the choices are guided by convergence considerations over when the branch and bound is applied to actual problems. Branching on a GUB set involves partitioning the variables in the set into 2 mutually exclusive and collectively exhaustive subsets, and setting the variables in each subset to zero consecutively to form two resulting subproblems. The rules to effect the partitioning of the variables in a given GUB set is described shortly.

Before presenting the algorithm, several terms need to be defined. z^{best} refers to the value of the best feasible solution available, or the incumbent solution. Note that $(X_v, X_s) = 0$ is always a feasible solution to (P), yielding $z^{best} = 0$. A feasible solution is one that satisfies the constraints in (P). Let L be the set of live nodes. The value of the LP optimization at node k is denoted by z_k. Thus, $z_0 = z(P^0)$ is the solution at node 0. Other terms will be defined as they arise. In what follows, the branch and bound is presented as Algorithm 1. This is followed by a detailed description of each step in the algorithm.

[6] See appendix B.

Algorithm 1: GUB Branch and Bound for (P)

Step (0): Initialization

Solve the LP relaxation (P^0) at node 0. If solution is feasible, stop. It is optimal. Otherwise, include 0 in the list of live nodes: $(L \leftarrow \{0\})$

Step (1): Node Selection

Choose a candidate subproblem corresponding to node q such tht $z_q < z^{best}$. If no such problem exists; (L is empty); stop. z^{best} is the optimal solution.

Step (2): GUB Set Selection

Select a GUB set p having more than one non-zero element in the candidate subproblem to branch on.

Step (3A): Branching

Create and solve the LP relaxation of 2 new subproblems corresponding to children of node q, denoted as nodes $q+1, q+2$. Partition the variables not fixed at zero in the GUB set p into 2 mutually exclusive subsets LL_p, LR_p. Subproblems $q+1, q+2$ are obtained by setting variables in LL_p, LR_p to zero consecutively. Remove q from the lists of live nodes. $(L \leftarrow L - \{q\})$

Step (3B): Bounding

Solving subproblems $i = q+1, q+2$ yields the following cases:

1. $z_i \geq z^{best}$: i is fathomed by bound, or feasibility. Go to 1.
2. $z_i < z^{best}$: either
   (a) Subproblem i is feasible.
   Update incumbent: $z^{best} \leftarrow z_i$. Fathom by feasibility. Go to 1.

(b) Subproblem i is infeasible.

Include i into list of live nodes: $L \leftarrow L \cup \{i\}$. Go to 1.

Step (2) parallels the variable selection phase in the single variable branch and bound. The rules adopted for node and GUB set selection for problem (P) are now shown. These are Steps (1) and (2) of Algorithm 1.

Node Selection

Two methods for node selection are presented. The first is a hybrid approach which combines the best objective criteria and the infeasibility set criteria. The second deals explicitly with the case when a strong incumbent is available at some point in the branch and bound, and the number of live nodes is large. Both methods are presented in what follows

A. Best objective and Infeasibility set criteria

Initially, nodes were chosen based on a weighted measure of its objective value and the number of infeasible GUB sets. Traditionally, the node with the greatest objective value is chosen so that further restrictions would encourage fathoming through feasibility or bounding. That is, if L is the set of live nodes, select node $q \in L$ such that

$$z_q \geq z_k \ \forall \ k \in L.$$

This is often referred to as the best objective criteria, and corresponds to a steepest descent criteria. However, experiments by Forrest[14] and Gauthier[16] concluded that this measure alone is unsatisfactory. It must be pointed out that some of the tests were done on pathologically contrived problems for general mixed integer programs. For problem, (P) however, the best objective criteria is used at the initial stages of node selection. Moreover, the number of infeasible GUB constraints resulting from the LP solution at a node, (those

that contain fractional solutions) reflect its potential of yielding feasible solutions through few subproblem optimizations. In rare cases, the number of infeasible GUB constraints might be expected to increase initially. Subsequent optimizations are expected to reduce the number of infeasibilities since GUB constraints are satisfied incrementally as the branch and bound progresses. Thus, another rule for node selection is to select nodes having the greatest number of infeasible GUB constraints. Specifically, if Θ_k is the number of infeasible GUB constraits corresponding to node k. Then node $q \in L$ is selected such that

$$\Theta_q \leq \Theta_k \ \forall \ k \in L$$

This is called the infeasibility set criteria.

A useful choice for node selection is then to combine the best objective criteria with the infeasibility set criteria. To this end, the best objective criteria is used initially until the objective values for the live node show little variability. At this stage, the number of infeasible GUB constraints differs only slightly. As the range of objective values for the live nodes increases, the range Θ_k shows much variation as well. It is not true, in general, that the node with the largest z_k has the smallest Θ_k. When the range of objective values reach a prescribed limit, node selection is based on the infeasibility set criteria. In cases of ties, the best objective criteria is reverted.

The above scheme can be summarized as a weighted measure of z_k and Θ_k, where z_k is weighted more heavily during the initial phase and subsequently Θ_k during the branch and bound.

The next topic deals directly with how to estimate the potential of each node in providing solutions that are better than the incumbent. This is useful for problems that have a large amount of live nodes. Processing all these nodes would be cost ineffective, if they are projected to yield solutions that are unlikely to be better than the incumbent. The procedure relies on the availability of a good incumbent solution. It is also used as the basis

for node selection when a good incumbent is available. This procedure is known as the Best Projection(BP) Criteria, which is described next.

B. Best Projection Criteria

Let x_i^k = value of x_i at node k.

Let G_j^k = set of variables in infeasible GUB constraint j at node k, and define the infeasibility of GUB constraint j by

$$\beta_j = \min\left\{ \sum_{x_i \in G_j^k} x_i^k \ , \ 1 - \max_{x_i \in G_j^k} x_i^k \right\}$$

Then the sum of integer infeasibility at node k is

$$\alpha_k = \sum \beta_j$$

for all infeasible GUB constraints j at node k. Since in any feasible solution at most one variable can be basic with value one, the left terms[7] measure the infeasibility in the case where all variables are zero, and the right term provides an optimistic measure by forcing the largest fractional variable to 1.

Consider the graph in figure 3.2, where horizontal axis, α, is the sum of integer infeasibility and the vertical axis, z, is the LP objective value. A live node $k \in L$, is represented in figure 3.2 as a point (α_k, z_k). Note that $z_k \leq 0 \ \forall \ k \in L$. Thus, (α_0, z_0) is the point corresponding to solutions of (P^0), at node 0. Any feasible solution corresponding to node v has $\alpha_v = 0$ and lies on the vertical axis. If $A = (0, z(P))$ corresponds to the optimal solution, (P), then,

$$\gamma = \frac{z_0 - z(P)}{\alpha_0}$$

is the expected degradation in z_0 per unit reduction in the sum of infeasibility from α_0 to 0. That is, γ is a measure of the rate of expected degradation from the LP solution

[7]Because of the inequality, the left term is not necessarily one

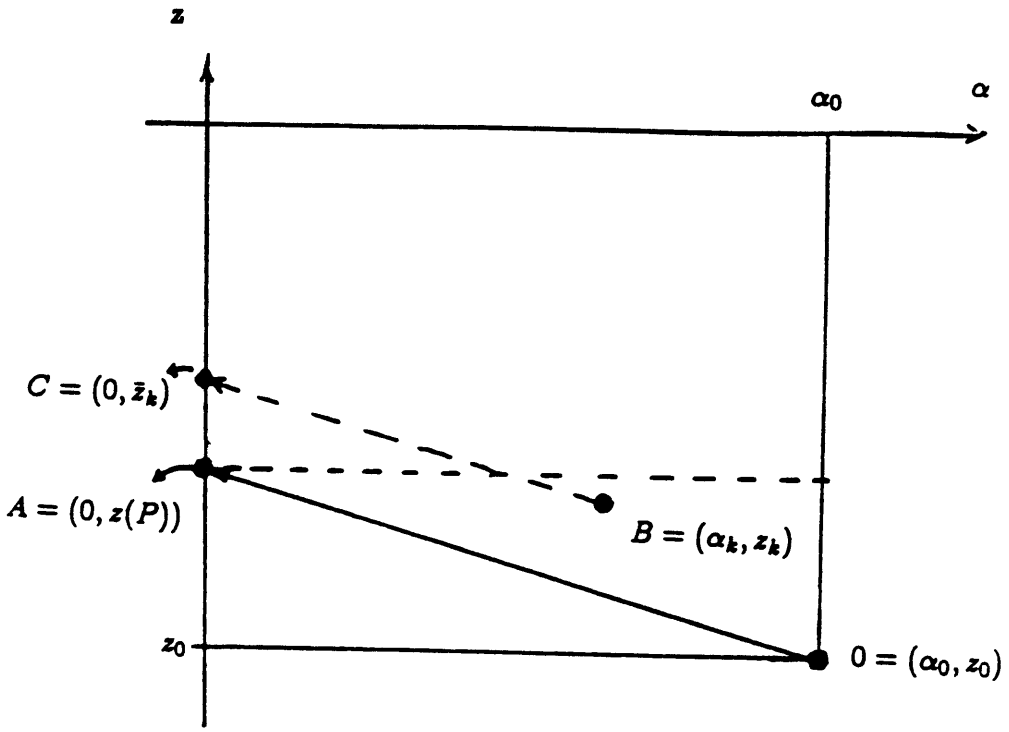


Figure 3.2 Best projection criteria

to the optimal solution. If the expected degradation of problems corresponding to live nodes is assumed to follow closely along the same gradient as OA, then, for a point $B = (\alpha_k, z_k)$, the estimated integer solution obtained by reducing its sum of integer infeasibilities is $\bar{z}_k$, given by

$$\bar{z}_k = z_k + \gamma * \alpha_k.$$

Geometrically, we project B to point C along the same gradient as OA. Thus, ideally, node $y \in L$ is chosen where

$$\bar{z}_q = \min_{k \in L} \{\bar{z}_k\}$$

That is, the node with the best potential of yielding the smallest objective value. Unfortunatley, the optimal solution to (P), and thus point A, is unknown. Thus, A is replaced by

the incumbent solution, that is, point $A = (0, z^{best})$. It is crucial that z^{best} is a good approximation to $z(P)$ or else, the gradient of OA will be too steep, thus incurring erroneous estimations. Each time A is updated with a feasible solution yielding a better value, γ is also changed. However, since the BP criteria is applied only when z^{best} is a good approximation of $z(P)$, few such changes are made.

Note that all points lie between $z = z^{best}$ and $z = z_0$. Successive optimization from node k follows a monotonically non-decreasing path from (α_k, z_k) to the vertical axis z. Since 0 is a feasible solution to any subproblem, each node traces a path toward the vertical axis. Points lying to the left of OA have greater potential of yielding a solution better than z^{best}. However, those points distributed around 0 should not be ruled out, since the BP scheme is an approximation. Points lying to the right of OA and away from 0 offer lesser potential of yielding better solutions than z^{best}. Thus, a heuristic, all nodes, $k \in L$ having values $\bar{z}_k \geq (1 + \epsilon) * z^{best}$ can be removed from L, where $\epsilon \geq 0$ is a prescribed value. Of the remaining nodes, the one offering the best value $\bar{z}_k$ is chosen as the next node to evaluate. However, if the remaining nodes in L do not show much variability in their $\bar{z}_k$ values, then the BP criteria is not useful in projecting the difference in the potentials of the live nodes. Under such conditions, the previous criteria is adopted.

Since it is imperative to know the quality of z^{best}, it can be estimated as follows:

Let

$$z_{low} = \min_{k \in L} \{z_k\}$$

then, z^{best} is within

$$\left(\frac{z^{best} - z_{low}}{|z_{low}|} * 100 \right)$$

of the optimal solution $z(P)$.

The best projection criteria assumes that γ, the cost of removing one unit of infeasibility remains constant irregardless of whether variables in the GUB set are set to take on values

of one or zero. The pseudocost criteria applied to the single variable branch and bound (see Forrest[14] and Gauthier[16]), differs only in that it treats the degradation of objective differently depending on whether a variable is scaled up to 1 or down to 0. Each variable selected for branching has a up and down pseudocost corresponding to increasing or decreasing its value to 1 or 0, respectively. The pseudocost of a variable is only determined when it is chosen for branching and only after the pair of subproblem optimizations restricting its value to one and zero are solved. Although the pseudocost criteria offers a better measure of the rate of degradation of the objective function empirically, the results reflect only testings using single variable branch and bound. Incorporation of the pseodocosts for GUB sets, (which is the analogy for single variable in the branch and bound), is less obvious and robust. The dichotomy that results from partitions for GUB sets further clouds an accurate estimation at each branching. As such, the pseudo cost criteria is not chosen.

In what follows, the GUB set selection is described. This relates to Step (3) of algorithm 1, and is analogous to choosing variables in the single variable branch and bound.

GUB Set Selection

Having chosen the live node $q \in L$, the GUB set is chosen next to effect the branchings. The GUB set is selected from the subset of infeasible GUB constraints containing more than one basic variable at node q. This is analogous to selecting fractional variables to branch in the single variable branch and bound. Branching on infeasible GUB constraints, in many cases, reduces the total number of infeasible GUB constraints for the resultant problem. This happens because the branching involves setting the flow of at least one variable to zero. Based on this idea, the infeasible set was chosen which has the largest number of basic variables. In cases where there is a tie, the GUB constraint having the largest range of reduced costs over its set of non-basic variables is chosen. Reduced cost criteria is chosen over the absolute cost criteria because it takes into account both the impact on the objective

function value and the feasibility. Unlike the single variable branch and bound, where the variable with the greatest reduced cost is typically chosen as the next branching variable, a measure must be adopted to reflect the reduced costs of all variables in the set. The following definitions are useful.

Let,

- r_i^q = reduced cost of variable x_i at node q.
- Ω_q = set of infeasible GUB sets with more than one nonzero element at node q.
- Q_j^q = set of variables not fixed at zero belonging to GUB constraint $i \in \Omega_q$.

Note, that if Ω_q is empty, then the solution at subproblem node q is feasible, since at most one variable is basic for each GUB constraint at node q.

Two modified reduced costs measure are proposed:

I. Average Absolute Reduced Cost Criteria

Let δ_j^q be the average reduced cost of variables in GUB constraint $j \in \Omega_q$. Evaluate:

$$\delta_j^q = \frac{\sum_{x_i \in Q_j^q} |r_i^q|}{|Q_j^q|} \quad \forall \; j \in \Omega_q$$

and select the GUB constraint $p \in \Omega_q$, such that

$$\delta_p^q \geq \delta_j^q \quad \forall \; j \in \Omega_q.$$

The reduced costs are averaged since at most one variable takes on non-zero value for any feasible solution.

II. Range of reduced cost criteria

Let Ψ_j^q be the range of reduced costs for GUB constraint $j \in \Omega_q$. Evaluate,

$$\Psi_j^q = \max_{x_i \in Q_j^q} \{r_i^q\} - \min_{x_i \in Q_j^q} \{r_i^q\} \quad \forall \; j \in \Omega_q$$

and select the GUB constraint $p \in \Omega_q$ such that

$$\Psi_p^q \geq \Psi_j^q \ \forall \ j \in \Omega_q$$

The reduced cost criteria attempts to measure the maximum differential effect that the GUB variables have on the objective function and and the coupling network flow constraints.

Extensive experiments by Sweeney[34] using the reduced cost range criteria for multi-item scheduling problems have worked very well. This is also the criteria chosen here.

In summary, the infeasible GUB set with the largest number of basic variables is chosen. If a tie exists, the GUB set with the greatest range of reduced costs is selected.

GUB Set Partition

After selecting the GUB set p to effect branching, the variables are partitioned to form two subsets LL_p, LR_p whereby nodes $q+1$ and $q+2$ are formed.

Let the variables in GUB set p not fixed to zero be ordered such that, $x_1 + x_2 + , \cdots , + x_m \leq 1$ where $x_1, x_2, \cdots, x_v$ are the basic (non-zero) variables corresponding to GUB constraints p. Note that $2 \leq v \leq m$.

Let $r = \lfloor \frac{v}{2} \rfloor$, so that $x_1, x_2, \cdots, x_v$ form 2 nonzero subsets $D_1 = \{x_1, x_2, \cdots, x_r\}$ and $D_2 = \{x_{r+1}, x_{r+2}, \cdots, x_v\}$. Also, let $y = \lfloor \frac{m-v}{2} \rfloor$ and form 2 subsets $D_3 = \{x_{r+1}, x_{r+2}, \cdots, x_{r+y}\}$ and $D_4 = \{x_{r+y+1}, \cdots, x_m\}$. Then set $LL_p = D_1 \cup D_4$ and $LR_p = D_2 \cup D_3$. What is done is to partition the GUB variables into 2 subsets such that each subset contains about equal numbers of basic and nonbasic variables. This particular rule of partition is selected to favour a balanced tree sturcture, and to distribute the restrictions evenly over subsequent subproblem optimizations. In step (3), then, setting variables in LL_p to zero results in reducing the GUB constraint p to $\sum_{x_j \in LR_p} x_j \leq 1$. To enforce this constraint to the problem at node $q+1$, the upperbounds of variables x_j in LL_p are set to 0. The resultant LP is labelled as node $q+1$. Similar arguments apply when variables in LR_p are

set to zero to produce node $q + 2$.

This section presents and proposes a GUB branch and bound algorithm to solve the routing problem with service choices. The basic issues relating to node and set partitions are described in detail. In the next section, it is shown how the basic GUB branch and bound can be enhanced, in order to improve its efficiency.

3.3 Improvements over the branch and bound

3.3.1 Solving LP with GUB constraints

In the previous section the GUB branch and bound algorithm was described in detail. The optimization at each node is a special LP with GUB constraints. The efficiency for solving the resultant LP at each node is an important one, and has received much attention. In large, multi-item scheduling problems, with GUB constraints the integrality restriction is usually relaxed and the continuous solution rounded to produce acceptable solutions based on some prescribed set of rules. For such problems, the coupling constraints are soft, and judicious rounding of fractional solutions will not cause unacceptable infeasiblities, see Lasdon[24]. Moreover, for many mixed integer programming problems with GUB constraints, the solution of the LP relaxation provides a very tight bound. Thus, it is often cost effective to solve the LP relaxation. Facilities to handle these constraints effectively have been studied by Dantzig and Van Slyke[11] and more recently by Grigoriadis[20], Hartman[21] and Lasdon and Terjung[24]. Essentially, the basis for these special LP have a special structure that allows the pivoting to be done exclusively with a reduced basis of size equals to the rank of the coupling constraint matrix. Appendix B describes how an LP with GUB constraints can be solved efficiently by maintaining a basis of size equal to the conservation of flow constraints, less one, since the node arc incidence matrix contains one redundant constraint.

Due to the existence of GUB constraints in many practical problems, most sophisticated computer codes now include an option to handle these constraints implicitly. In addition, these codes incorporate features that deal explicitly with the sparsity of the constraint matrix, which in our case is the node arc incidence matrix, in order to improve the overall computational efficiency. Initial tests by Dantzig[11] reveal that solutions of problems where the GUB constraints are included implicitly are of a magnitude 10 times faster than that

Code Name	Organization	Computer	Date
Apex III	Control	Cyber 70/72,73,74,76	1975
FMPS	Sperry-Univac	Univac 1100	1976
Haverly-MIP	Haverly Systems	IBM 360/370,Univac 90/30	1970
MIP/370	IBM	IBM 370	1974
MPS	Honeywell	Series 60	1973
Sciconic	Scicon	Univac 1100	1976
Tempo	Burroughs	B7000,B8000	1975
XDLA	ICL	1900 series	1970

Table 3.1 Codes which incorporate GUB constraints. Source: Garfinkel[15].

using conventional codes. Further improvements are realized by Hartman[21].

Not all LP codes contain facilities to handle GUB constraints implicity. Table 3.1 includes a list of the more popular codes available that handle GUB constraints.

In addition, appendix B, includes a Dantzig-Wolfe decomposition procedure which solves the LP relaxation (P^0) by maintaining a basis equal to the size of the number of GUB constraints. This is the general approach taken by Chen and Saigal[10]. However, a more general problem for solving LP relaxations of network flow problems with side constraints was solved. The Dantzig-Wolfe procedure iterates between solving a network flow subproblem and a resource sharing master problem. It is not clear how this procedure compares with the previous one. Details of the method were developed to provide an additional focus on solving (P^0). Based on the above observations, it is safe to infer that the LP with GUB constraints at each node can be solved efficiently. In the next section, it is shown how the problem size of (P) can be reduced based on its special structure. This leads to further improvements in the computational efforts required to solve the LP relaxations.

3.3.2 Network aggregation

The previous section shows how the LP computation at each node can be performed efficiently. In practice, the solution time for an LP increases approximately proportionally with the number of variables, and proportionally with the square of the number of constraints. Since the solution of (P) involves solving (P^0) and a series of related LP, it is advantageous to reduce the size of (P). This section recommends a network reduction scheme aimed at reducing the size of the (P). The reduction not only reduces the number of rows(nodes) and columns(variables) of (P), but also the number of integer variables and GUB constraints. This occurs since a resultant GUB constraint having only one variable can be removed, with its variable upperbound set to one.

Given a schedule map, rules for aggregating nodes and removing arcs that exploit the special structure of the network consist of 2 phases, node aggregation and arc removal. Since node aggregation precedes arc removal, it is explained first in what follows.

Node Aggregation:

Let g_i, g_{i+1} represent 2 nodes along the same city axis such that g_i directly precedes g_{i+1}. The following definitions are useful.

Departure Node: A node is a departure node if there is at least one service arc eminating from it.

Arrival Node: A node is an arrival node if there is at least one service arc arriving into it.

Note that a node can be both a departure and arrival node.

Principle of Node Aggregation(POA):

A pair of nodes g_i, g_{i+1} connected by ground arc (g_i, g_{i+1}) is aggregated if g_i and g_{i+1} do not represent departure and arrival nodes respectively, and the

upperbound on arc (g_i, g_{i+1}) is infinite.

Whenever a pair of nodes g_i, g_{i+1} is aggregated, the result is a single node g_{i+1} with the arc connecting i to j removed as shown in figure 3.3.

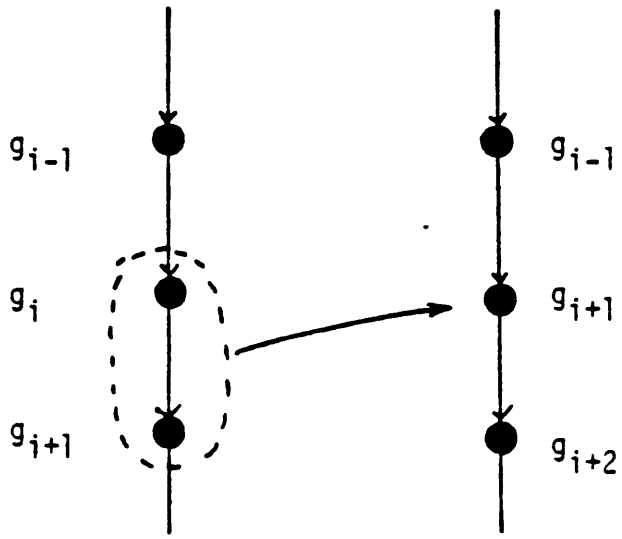


Figure 3.3 Node aggregation

If the upperbound along the ground arc is finite, the removal of the ground arc might cause more flow than is allowed to pass through. In such a case, the aggregation can result in infeasible solutions. This problem arises when the sum of the incoming flows into node g_i exceeds the upperbound on the ground arc (g_i, g_{i+1}). Conversely, if the nodes being aggregated are connected by a ground arc with an infinite upperbound, the above aggregation rule is optimal provided the ground arc cost removed is properly adjusted. Typically, only few ground arcs belonging to each city have finite upperbounds. In order to preserve feasibility, node aggregation is not performed between nodes connected by a ground arc with a finite upperbound.

Since the ground arc removed has a cost associated with it, costs on adjacent arcs to nodes g_i or g_{i+1} must be adjusted to reflect the exclusion. To show how arc costs are

adjusted, 2 cases are distinguished:

Case 1: g_i is not a departure node.

Then the cost along arc (g_i, g_{i+1}) is added to:

1. All incoming service arcs to g_i

2. Non service arc arriving at node g_i.

Case 2: g_i is a departure node.

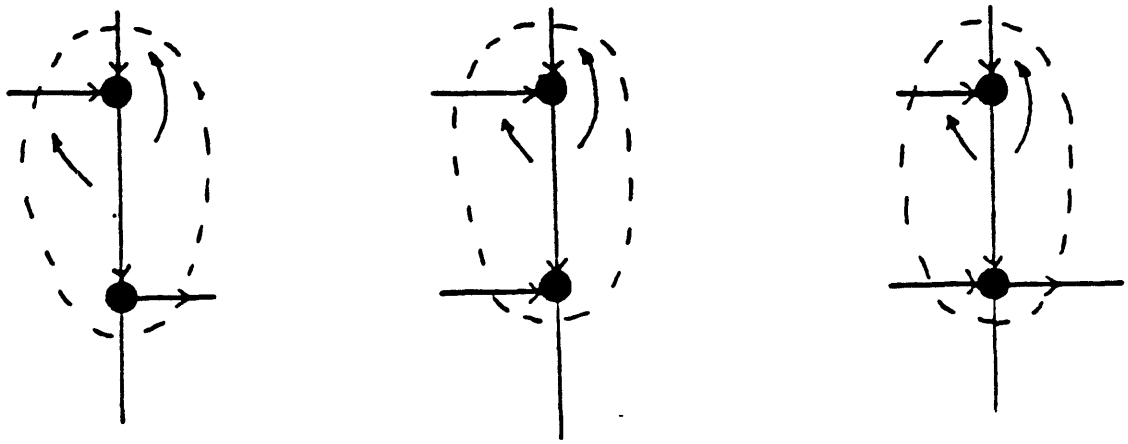
The ground cost along arc (g_i, g_{i+1}) is added to:

1. All outgoing service arcs from g_{i+1}.

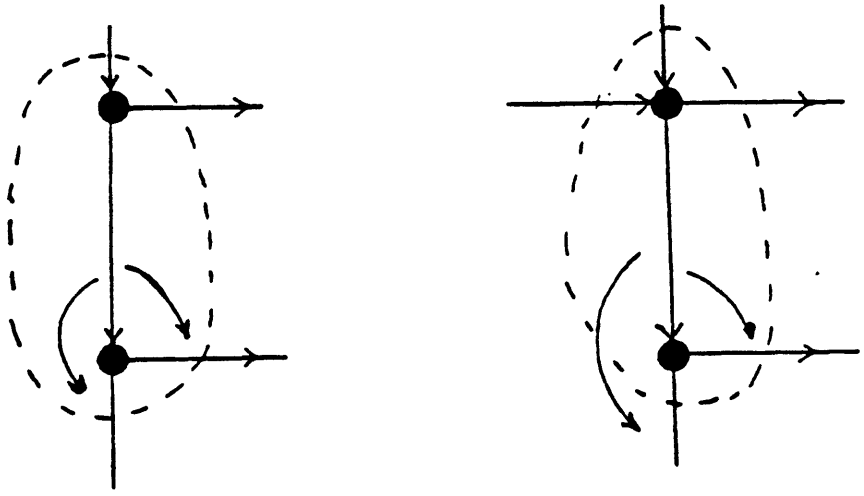
2. Outgoing non service arc from node g_{i+1}.

In case 1, all incoming services arcs to g_i are redirected to g_{i+1} and g_i is removed. In case 2, all outgoing arcs from node g_{i+1} is diverted to leave from g_i and g_{i+1} is removed. The resultant node is labelled g_{i+1}. The examples shown in figure 3.4, illustrate the aggregation.

Node aggregation for the entire network is described next. Cities are labelled $j = 1, 2, \cdots, |K|$, and the number of event nodes at city j is I_j. Furthermore, the nodes are ordered such that g_i lies above g_{i+1} for $1 = 1, 2, \cdots, I_j - 1$. For each city, begin with the first 2 nodes g_i, g_{i+1}, and determine if they can be aggregated based on the principle of node aggregation. If so, the rules of aggregation as described above are applied. The resulting node is relabelled g_{i+1}. Next, explore the node pair g_{i+1}, g_{i+2}, and so on until all nodes are considered. If node g_i and g_{i+1} cannot be aggregated, proceed on with g_{i+1} and g_{i+2}. The details of the node aggregation are shown in algorithm 2.



Case 1



Case 2

Arrows show how the ground arc cost is transferred or added.

Figure 3.4 Node aggregation and arc costs adjustments

Algorithm 2: Network Node aggregation

Procedure: node aggregate

Begin;

Do while ($j \leq |K|$)

$i \leftarrow 1$

do while ($i < I_j$)

if ({ g_i, g_{i+1} }) satisfies { POA } then

AGGREGATE;

relabel new node g_{i+1};

else

$i \leftarrow i + 1$

endif

end do;

$j \leftarrow j + 1$

end do;

End;

Arc removal

After node aggregation is completed, some service arcs in the transformed network can dominate over other service arcs belonging to the same GUB constraints. A process to reduce dominated service arcs is now explained. Let g_x and g_y be 2 nodes pertaining to different cities. Furthermore, let A_{xy} denote the service arc set between g_x and g_y. For every GUB constraint represented in A_{xy}, keep only one arc with similar orientation, belonging to each GUB constraint, having the highest contribution, (break ties arbitrarily), and remove all others belonging to the same set. This particular arc is said to dominate over subset of arcs in the same GUB set, between the given pair of vertices. Algebraically, arcs in A_{xy}

belonging to the GUB set have similar column vector in the constraint matrix, but with possibly different cost coefficients. Note that the constraint $X_s \leq 1$ is relaxed for the LP.

Arc removal is especially useful when applied to routing with time windows. For such problems, service arcs for each GUB constraint are more closely clustered, and the chance for extensive arc removal increases. In general routing problems with service choices, arc removal might not be instrumental in reducing the size of the problem.

Worst case consideration for aggregation

Thus far, very little is said regarding how many operations the node aggregation and arc removal scheme requires. In terms of nodes, the worst-case requirements for node aggregation is:

$$\sum_{i=1}^{|K|}(J_i - 1), \text{ where } \sum_{i=1}^{|K|} J_i$$

is the total number of nodes in the network. If $|N|$ is the total number of nodes in the network, the worst-case of the node aggregation requires $|N|+|K|$, or $O(|N|)$ computations. Arc removal requires comparisions for a total of $|N|(|N|-1) = |N|^2 - |N|$, node pairs, and thus an order of $|N|^2$ operations. The network aggregation scheme described is optimal. In most cases, the number of ground arcs having infinite upperbounds far exceeds those having finite upperbounds. Thus, the benefits of network aggregation far outweigh the penalty incurred through pre and post processing, which results in a reduced problem size.

Network aggregation is applied on the network only once prior to branch and bound. Even though reduction rules hold at each node, problem size reduction during branch and bound is not advisable since the information pertaining to the LP at that node would be lost.

Thus far, the improvements suggested deal explicitly with the efficiency of solving LP. Measures for pruning the branch and bound search are considered next. Appendix A, shows how a good feasible solution can be used to eliminate or fathom live nodes whose

objective value is greater. Therefore, it is desirable to find a good feasible solution in order to initiate the search. However, the method for finding a feasible solution should not require excessive computational time, and will, hopefully, utilizes the structure of the problem and any information available. This is the path taken, which is described in what follows.

3.3.3 Parametric search for feasible solutions

This section describes a procedure for finding a good feasible solution by parametrically altering the sum of the flows along the cycle arcs. The sum of cycle flows is equal to the fleet size of (P).

The special structure of (P) restricts all flows to begin and end at the cycle arcs. Any infeasible solution to (P) is decomposible into a set of cycle flows, where at least one such cycle flow carries fractional values. These are consequences of the conservation of flow and GUB constraints. Altering the bounds on cycle arcs can often result in drastic changes in the flow patterns because these arcs behave like bottlenecks on the flows in the network. Consequently, it is desirable to identify the appropriate cycle arc (arcs), and impose bounds in such a manner as to bias a feasible flow. In so doing, the objective is penalized, and the proper choice would be to alter bounds on those arcs that incur the least penalty. It is difficult for any problem to identify these cycle arcs. Indiscriminate alteration of bounds on some arbitrary cycle arcs penalizes the objective excessively. To circumvent this problem, the sum of the cycle flows is bounded, instead, based on the solution provided by (P^0). In what follows, the method for finding feasible solutions is explained.

Let $(x_1, x_2, \cdots, x_b) \subset X_v$ be the set of cycle arcs in (P) and $(u_1, u_2, \cdots, u_b)$ be its corresponding upper bounds. Moreover, let $(x_1^0, x_2^0, \cdots, x_b^0)$ be the flows along the cycle arc corresponding to solutions of (P^0). The sum of cycle flows of (P^0) is $F^0 = x_1^0 + x_2^0 + \cdots + x_b^0$. A graph of objective value z versus fleet size F (sum of cycle flows) is shown in figure 3.5, where $F^{max} = \min(u_1 + u_2 + ..., +u_b$, sum of the number of GUB constraints and service arcs not belonging to any GUB constraints). The LP solution to (P^0) is represented as point D. Let i be integer value lying between, and including 0 and F^{max}. Then, the optimal solution to (P) is a point lying on one of the vertical axis $F = i$, $0 \leq i \leq F^{max}$, and bounded by the region AEFB. Note that the sum of cycle flows cannot exceed F^{max}. In

general, F^0 need not be integral. The heuristic for finding a feasible solution comprises 2 phases, H1 and H2. H1 tries to find a feasible solution bounded by the region ACDB by bounding the sum of cycle flows between F^0 to 0. H2 tries to find a feasible solution bounded by region CEFD by bounding the sum of cycle flows between F^0 to F^{max}. The feasible solution yielding the best objective from H1 or H2 is chosen as the best feasible solution.

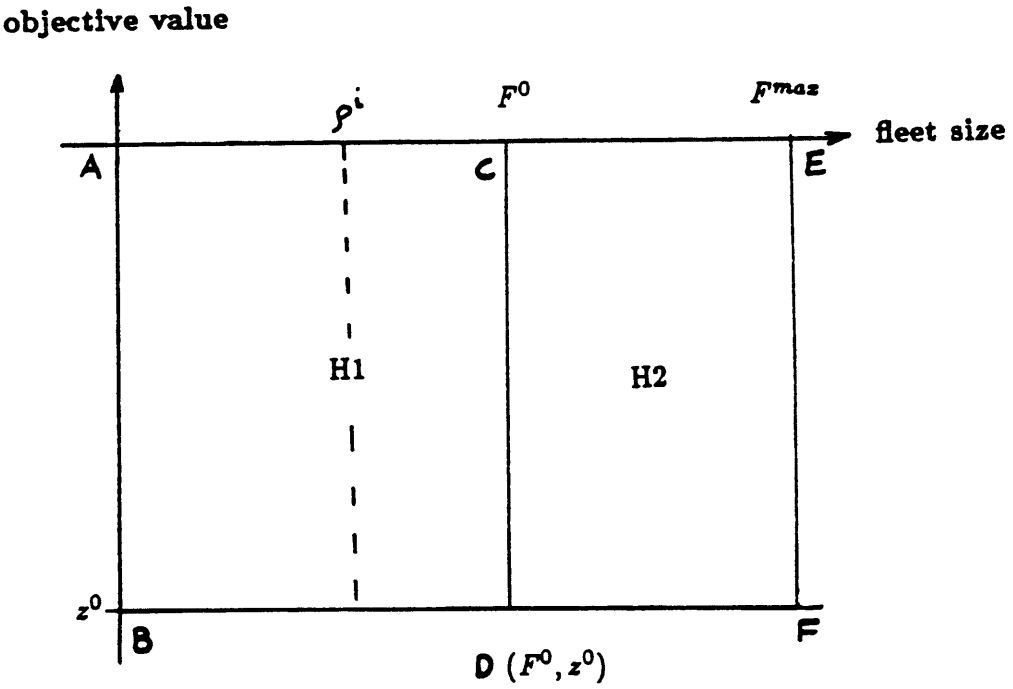


Figure 3.5 Graph of vehicle fleet size vs. objective value

For phases H1 and H2, let $(x_1^i, x_2^i, \cdots, x_b^i)$ be the cycle flows when (P^0) is solved with the constraints $x_1 + x_2 + \cdots + x_b \leq \rho^i$. Beginning with $i = 0$, $z^{stop} = 0$, ρ^i is obtained progressively as shown in algorithms H1 and H2. In addition, let $F^i = x_1^i + x_2^i + \cdots + x_b^i$

be the sum of cycle flows at iteration i.

Algorithm H1: Finding a feasible solution yielding fleet size $F \leq F^0$.

Step (1): Determine ρ^i

$$\rho^i = \begin{cases} \lfloor F^{i-1} \rfloor & \text{if } F^{i-1} \text{ is fractional} \\ F^{i-1} - 1 & \text{otherwise} \end{cases}$$

If $\rho^i = 0$, then the best feasible solution is given by $(F, z) = (0,0)$, stop. Otherwise, go to (2).

Step(2): Reoptimize (P^0)

Append the constraint $x_1 + x_2 + \cdots + x_b \leq \rho^i$ to (P^0) and resolve.[1] If solution is feasible; or its objective value greater than or equal to z^{stop} then stop. Otherwise, $i \leftarrow i + 1$. Go to (1).

Observe that as ρ^i decreases, the objective values obtained in Step (2) increase monotonically. Thus, the process is stopped whenever a feasible solution is found.

Finding a feasible solution whose sum of cycle flows exceeds F^0, follows similar arguments as those presented in Algorithm H1 with some slight modifications. $F^i \; \forall \; i$ are defined as before, starting with $i = 1$.

Algorithm H2: Finding a feasible solutions yielding fleet sizes $F > F^0$

Step (1): Determine ρ^i

$$\rho^i = \begin{cases} \lceil F^{i-1} \rceil & \text{if } F^{i-1} \text{ is fractional} \\ F^{i-1} + 1 & \text{otherwise} \end{cases}$$

if $\rho^i > F^{max}$, stop. No feasible solution if found.

Step (2): Reoptimize (P^0)

Append the constraint $\rho^i \leq x_1 + x_2 + \cdots + x_b$ to (P^0) and resolve if solution is feasible, or its objective value greater than or equal to z^{stop}, stop. A feasible solution is found. Otherwise, $i \leftarrow i + 1$. Go to (1).

[1]Note that for $i > 0$, this involves changing the right hand side of ρ^i parametrically.

In this case, as ρ^i increases, the objective values obtained in solving the modified (P^0) increases monotonically. Thus the method is terminated when a feasible solution is found. Unlike phase H1, there is no guarantee that a feasible solution can be found.

The method for finding a feasible solution alternates between phases H1 and H2, where the objective value, z^{stop}, of a feasible solution in one phase is used as an additional terminating criteria for the other. To illustrate this, suppose at some point z^{stop} is the objective value of a feasible solution found in phase H2. Then, H2 is terminated, and H1 continued until a feasible solution is found, the bounds on ρ^i is violated in step (1), or an objective value greater than z^{stop} is realized. Pursuing with H1 is futile since it cannot yield a better objective value than the current one. If a feasible solution is found, the one having the smaller objective value is retained. The best value obtained is labelled z_h.

In both phases, ρ^i need not change by one unit per iteration. It is conceivable that the solution at Step (2) results in $|\rho^i - F^i| > 1$. The value of F^{max} provides the worst case bound on the number of iterations required. In some cases, however, F^{max} can be better estimated. The method described explores both phases since it is not possible, in general, to know whether the best objective is served by using more or less vehicles. However, the method is not expected to require excessive iterations, and it is often adequate to implement phase H1 and H2 for ρ^i values in the vicinity of F^0. The next paragraph explains why. Typically, since the cycle arc costs are much larger, in general, than ground or service arc costs, the optimal solution to (P) is satisfiable more often by a smaller set of integral cycle flows than a larger one. As such, the solution is more likely to lie on a vertical axis on or to the left of $\lceil F^0 \rceil$. Moreover, as ρ^i increases in phase H2, many unprofitable cycle flows having contributions greater than zero are formed. This means, that an objective value greater than zero is quickly obtained, and phase H2 terminated. By comparison, the penalty on the objective for flow boundings on phase H1 is less severe. It is likely too, that a

better feasible solution can be found using H1. Thus, this phase deserves to be implemented in its entirety.

A good incumbent solution is extremely useful to drive the GUB branch and bound. The computational efforts are more often justifiable. To alleviate extensive computations, the above method uses the informations from the LP solution to (P^0) and solves a series of related problems parametrically. As such, it can be conveniently married to the GUB branch and bound.

The objective in this last section is to implement a procedure to search for a feasible solution effectively to initiate the GUB branch and bound. Varying flows along service arcs in each GUB constraint is unlikely to work well since it places insufficient restrictions. Moreover, enforcing feasibility on a GUB constraint by manipulating flows of its variables is likely to result in infeasiblities on some other GUB constraint. Conversely, restrictions placed on the sum of cycle flows, penalizes the objective enough to drastically alter the flow patterns. Such restrictions and penalty discourages the formation of optimistic fractional flow patterns.

This section relates to upper bounding on (P) prior to branch and bound. In the next section, a method of lower bounding related optimization problem at each node is described.

3.3.4 Objective cut enhancement

This section deals explicitly with lower bounding the optimization problem at each node of the branch and bound tree. A good lower bounding technique finds a better bound than that obtained from solving the LP. Using such a bound can help fathom nodes whose objective values are lesser than those of the incmbent solution. However, there is clearly a tradeoff between computing a better bound at each node, and finding a good feasible solution, so that the search can be effectively pruned.

A simple bounding at each node relates to the problem data, (C_v, C_s) of (P). Given that (C_v, C_s) are integral, the objective value of (P) must also be integral. Thus, a fractional LP objective value, z^k, obtained at node k can be replaced by $\lceil z^k \rceil$. A modified solution can then be obtained by appending the objective cut $C_v X_v + C_v X_s \geq \lceil z^k \rceil$ at node k, and resolving. However, it is often unlikely that such a reoptimization is ever useful in improving the bound significantly. Based on the special structure of (P), a method to obtain a better lower bound at each node is presented next.

Let V^k be the sum of cycle flows $x_1^k + x_2^k + \cdots + x_b^k$ at node k, where x_i^k = flow value of cycle arc x_i, at node k. If V^k is fractional, then 2 subproblems are solved by appending constraints:

$$x_1 + x_2 + \cdots + x_b \leq \lfloor V^k \rfloor$$

and

$$x_1 + x_2 + \cdots + x_b \geq \lfloor V^k \rfloor + 1$$

to (P^k), respectively. The minimum objective value, $\hat{z}^k$, obtained from these pairs of subproblems optimization is used as a lower bound at node k. It means that the best feasible solution at node k can yield objective values no smaller than $\hat{z}^k$. Therefore if the solution yielding $\hat{z}^k$ is also feasible, then, node k is fathomed. the incumbent solution is updated only if $\hat{z}^k$ is smaller. Consequently, any problem that are descendents of k cannot yield a feasible

solution whose objective value is better than $\hat{z}^k$. Thus, all subproblems corresponding to descendants of k are solved with an additional constraint, $C_v X_v + C_s X_s \geq \hat{z}^k$. The right hand side value of this objective cut can be modified whenever a new bound is obtained during subsequent optimizations. Such a cut is not valid for subproblems optimation at nodes j where node j precedes node k or whenever node j does not lie in the same path from node 0 to k. Incorporating such a lower bounding procedure means that the objective cut has to be carefully monitored. Moreover, subproblem optimizations at each node can be time consuming. Thus, this procedure is used discriminately. It is recommended that this procedure be used only when the objective values of nearly all live nodes are close to that of the incumbent.

The objective cut described is obtained through solving subproblems generated by the sum of cycle flows. In some cases, V^k is integral, although, x_i^k, for $i = 1, 2, \cdots, b$, need not. Let $J \subset \{1, 2, \cdots, b\}$ be the index of fractional x_i^k. Then, for each $i \in J$, a pair of subproblems are solved by appending the constraints $x_i \leq \lfloor x_i^k \rfloor$ and $\lceil x_i^k \rceil \leq x_i$ to (P^k), respectively. Let d_i denote the minimum objective value obtained from solving these pair of subproblems. Then, the lower bound of node k, is equal to the $\max_{i \in J}\{d_i\}$. The properties of this solution are similar to those described earlier. In general, only a subset of arcs in J are used since otherwise, the lower bounding method can become computationally costly.

The above procedures describe means for obtaining a lower bound at node k no worse than that of z_k. Such a bound is used to fathom live node. In some cases, fathoming occurs because the best bound obtained corresponds to a feasible solution. Moreover, a better bound $\hat{z}^k > z_k$ can be used as an objective bounds for subproblems corresponding to nodes that are descendants of k.

Chapter summary

In this chapter, it is shown that Lagrangian and Benders decomposition methods which have worked well in some large scale problems failed to perform satisfactorily. To solve (P) optimally, the integrality properties need to be addressed. To this end, a specialized branch and bound procedure is suggested which incorporates the structure of the GUB constraints. An LP relaxation is solved corresponding to each subproblem optimization at a node. This is chosen since the LP relaxation is believed to provide a tight bound on the subproblem optimization at each node. In addition, several schemes are presented which help in reducing the problem size and pruning the search.

However, the success of any branch and bound scheme depends on its performance in practical problems. Its efficiency also depends on how it is implemented. In the next chapter, experiments on several test problems are described, and issues relating to implementation are also presented.

Chapter 4

Empirical Results and Implementation Issues

This chapter presents issues related to testing and implementation of the GUB branch and bound algorithm described in the previous chapter.

In Section 4.1, two examples are given to provide insights into the characteristics of the problem (P) that gives rise to fractional solutions. They reveal properties that helped explain why most LP relaxations of (P) solve to integrality. Furthermore, they provide clues on how to generate interesting problems for which the GUB branch and bound can be tested. A small example is then solved completely to illustrate the procedure described. Thereafter, results from solutions to several medium scale problems related to routing with service choices and routing time window constraints, are presented and discussed. These problems were generated to conform as much as possible to medium scale real life problems, and are solved interactively with LINDO as the LP support. The LP was solved using LINDO because it provides sufficient flexibility and interactive capabilities.

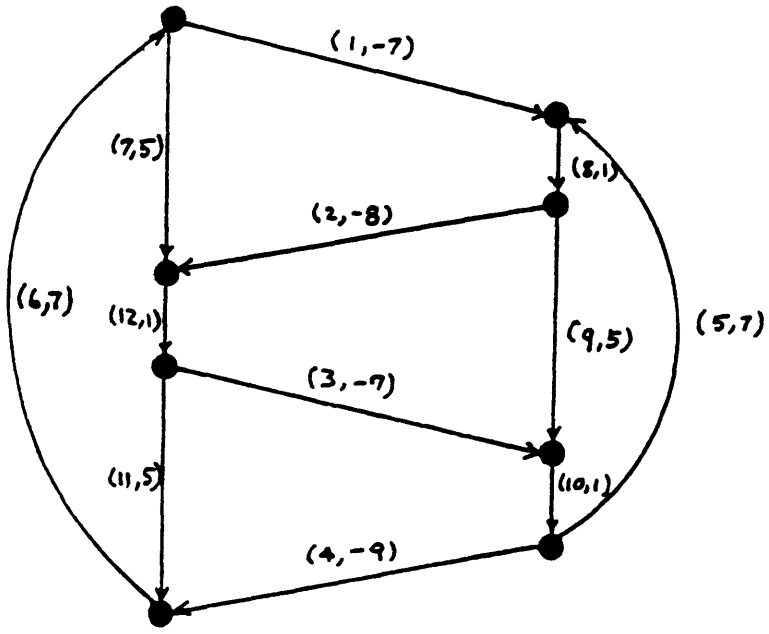
Since the GUB branch and bound differs from conventional single variable branch and bound, a method for implementing the branch and bound search is described in Section 4.2. Such a method is used to fully automate the GUB branch and bound.

Two small examples

Before discussing results for tests on some medium size problems, two small examples are given to provide flavor to the solutions of the LP relaxation, (P^0). They reveal interesting properties that provide clues as to when the LP relaxation is likely to return fractional solutions. Both examples are special instances of routing problems with time window constraints. The first example relates to the case where the time window for each service is large, and the second example considers the case where the time windows are tight.

Figure 4.1 shows a simple example of the routing problem with time windows containing two potential services, modelled by choices (x_1, x_3) and (x_2, x_4). The GUB constraints are $x_1 + x_3 \leq 1$ and $x_2 + x_4 \leq 1$. Note that the time window for x_1 is so large that it admits another service arc x_2 to fall within it with opposite orientation. The LP solution is -10.5, given by a flow of 0.5 along cycle C1. Observe that C1 contains at least one cycle arc, and need not admit integral flow. The optimal solution is -5.0 with a flow of one along cycle C2. This is verified by branching along arc $x_3 = 0$ and $x_1 = 0$ for GUB set $x_1 + x_3 \leq 1$. Similar results can be achieved by single variable branching along arc $x_3 = 0$ and $x_3 = 1$. However, if the fractional flows occur in large cycles containing many arcs belonging to similar GUB constraints, then restricting flow along a single variable might result in another cycle flow that is fractional. GUB branch and bound limits flow along a subset of variables in a GUB constraint and thus has a better chance of breaking infeasible cycle flows.

The next example, shown in figure 4.2, has small time windows for every service but, nevertheless, has fractional LP solutions. The LP solution is given by flows along 2 cycles, C3, and C4. Each cycle contains a flow of 0.5. The objective value is -7.5. However, note that C3 has 2 cycle arcs x_9 and x_{21}. This gives a sum of cycle flows equal to 1.5. To search for a feasible solution, two problems are solved parametrically by first appending the sum of cycle flow cut, $x_9 + x_{17} + x_{21} \leq 1$, then $x_9 + x_{17} + x_{21} \geq 2$, to the original LP.



(a, c_a)

a = arc x_a

c_a = contribution of arc x_a

$C1 = \{x_6, x_1, x_8, x_2, x_{12}, x_3, x_{10}, x_4\}$

$C2 = \{x_5, x_8, x_2, x_{12}, x_3, x_{10}\}$

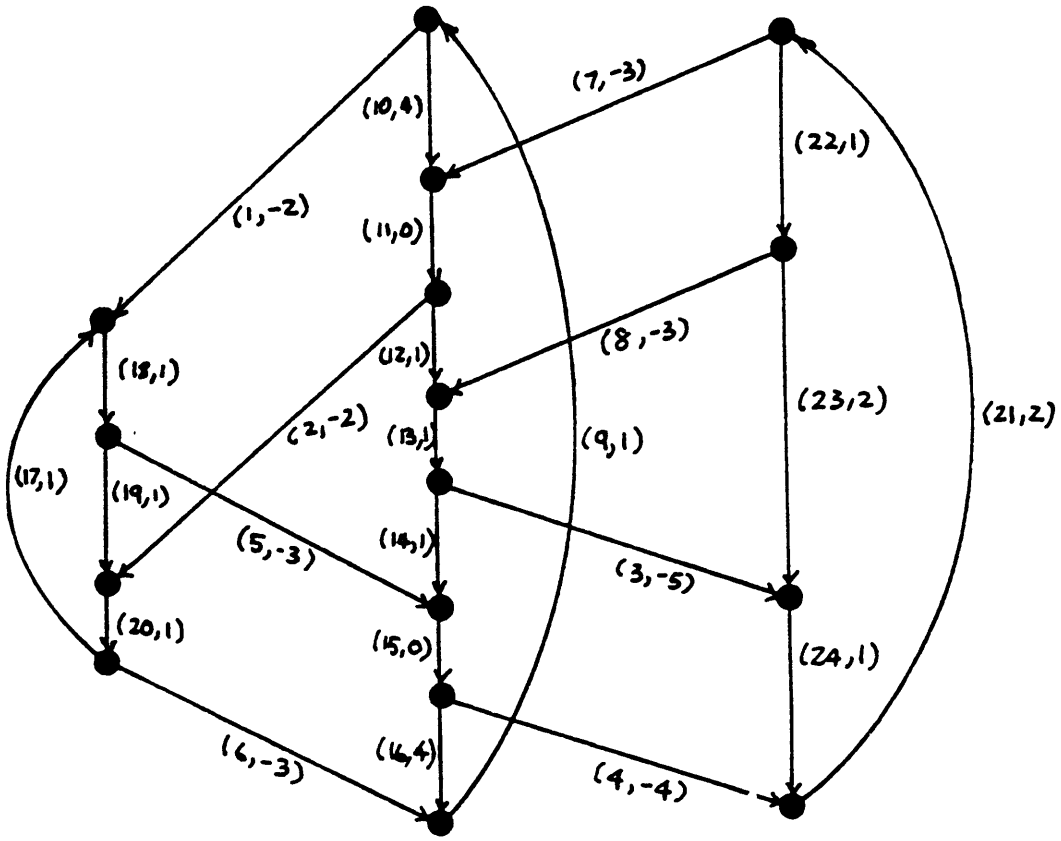
GUB constraints:

$x_1 + x_3 \leq 1$

$x_2 + x_4 \leq 1.$

Figure 4.1 Example of routing problem with large time windows.

As described previously, this is done because the sum of cycle flows must be integral. In this case, the optimal solution is given by the addition of the latter cut. Moreover, it is possible to conclude that such a solution is indeed optimal because the solution obtained by adding the latter cut produces a feasible solution whose objective value is smaller than that obtained by adding the first cut. To illustrate, the GUB branch and bound, however, it will be assumed that no feasible solution is known. First, select the infeasible GUB set for branching. Note, from figure 4.3a, that the infeasible GUB constraints are $x_1 + x_2 \leq 1$,



(a, c_a)

a = arc x_a

c_a = contribution of arc x_a

GUB constraints:

$x_1 + x_2 \leq 1$

$x_3 + x_4 \leq 1$

$x_5 + x_6 \leq 1$

$x_7 + x_8 \leq 1$

$C3 = \{x_9, x_1, x_{18}, x_5, x_{15}, x_4, x_{21}, x_7, x_{11}, x_2, x_{20}, x_6\}$

$C4 = \{x_{21}, x_7, x_{11}, x_{12}, x_{13}, x_3, x_{24}\}$

Figure 4.2 Example of routing problem with tight time window

$x_3 + x_4 \leq 1$, $x_5 + x_6 \leq 1$, with two nonzero elements in each set. Note that all infeasible GUB sets contain the same number of nonzero elements. Next, consider the range of reduced costs for all infeasible GUB sets with more than one nonzero element to select that which has the largest range of reduced costs. As it turns out, since all variables in the infeasible GUB sets are basic, they have zero reduced costs. See figure 4.3a. This means that the range of reduced cost is zero for every infeasible GUB set. To make it interesting, the set $x_3 + x_4 \leq 1$, is selected and 2 LP corresponding to setting x_3, then x_4 equal to zero are solved consecutively. These two subproblems are represented as nodes 1 and 2, in figure 4.4, with objective values -7.0 and -5.0, respectively. Node 2 is feasible and thus fathomed. Referring to figure 4.3b, the infeasible GUB sets are $x_1+x_2 \leq 1$ and $x_5+x_6 \leq 1$. Arbitrarily, select $x_1 + x_2 \leq 1$ to effect to next branching. Two subproblems are solved by enforcing x_1, then x_2 to zero, on the subproblem corresponding to node 1. Nodes 3 and 4 both return feasible solutions with objective values of -7.0 and -4.0, respectively, and are thus fathomed. The optimal solution is shown in figure 4.3c corresponding to node 3 in figure 4.4.

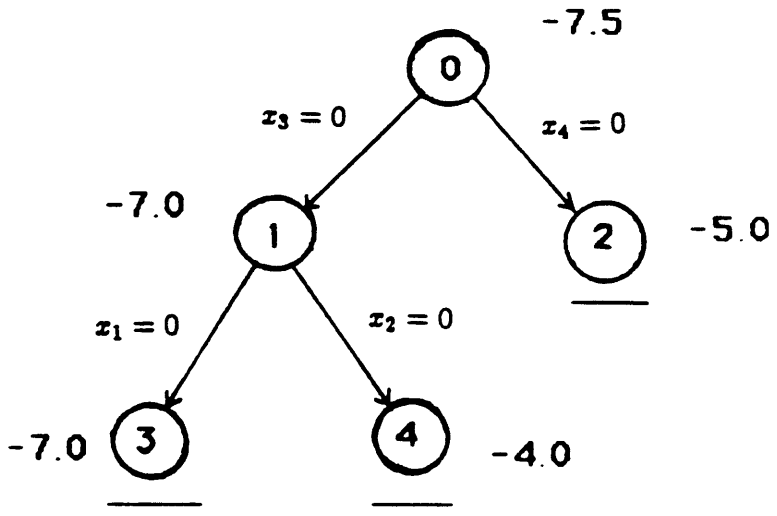


Figure 4.4 Branch and bound tree for example of figure 4.2

LP OPTIMUM FOUND AT STEP 19

OBJECTIVE FUNCTION VALUE

1) -7.50000000

VARIABLE	VALUE	REDUCED COST
ARC0001	0.500000	0.000000
ARC0002	0.500000	0.000000
ARC0003	0.500000	0.000000
ARC0004	0.500000	0.000000
ARC0005	0.500000	0.000000
ARC0006	0.500000	0.000000
ARC0007	1.000000	0.000000
ARC0008	0.000000	0.000000
ARC0009	0.500000	0.000000
ARC0010	0.000000	5.500000
ARC0012	0.500000	0.000000
ARC0013	0.500000	0.000000
ARC0014	0.000000	1.000000
ARC0016	0.000000	5.500000
ARC0017	0.000000	0.500000
ARC0018	0.500000	0.000000
ARC0019	0.000000	3.500000
ARC0020	0.500000	0.000000
ARC0021	1.000000	0.000000
ARC0022	0.000000	0.000000
ARC0023	0.000000	6.000000
ARC0024	0.500000	0.000000
ARC0011	1.000000	0.000000
ARC0015	0.500000	0.000000

Figure 4.3a Results of optimization at node 0

LP OPTIMUM FOUND AT STEP 2

OBJECTIVE FUNCTION VALUE

1) -7.00000000

VARIABLE	VALUE	REDUCED COST
ARC0001	0.500000	0.000000
ARC0002	0.500000	0.000000
ARC0003	0.000000	-1.000000
ARC0004	1.000000	0.000000
ARC0005	0.500000	0.000000
ARC0006	0.500000	0.000000
ARC0007	1.000000	0.000000
ARC0008	0.000000	0.000000
ARC0009	0.500000	0.000000
ARC0010	0.000000	6.000000
ARC0012	0.500000	0.000000
ARC0013	0.500000	0.000000
ARC0014	0.500000	0.000000
ARC0016	0.000000	6.000000
ARC0017	0.000000	0.000000
ARC0018	0.500000	0.000000
ARC0019	0.000000	4.000000
ARC0020	0.500000	0.000000
ARC0021	1.000000	0.000000
ARC0022	0.000000	0.000000
ARC0023	0.000000	6.000000
ARC0024	0.000000	0.000000
ARC0011	1.000000	0.000000
ARC0015	1.000000	0.000000

Figure 4.3b Results of optimization at node 1

LP OPTIMUM FOUND AT STEP 1

OBJECTIVE FUNCTION VALUE

1) -7.00000000

VARIABLE	VALUE	REDUCED COST
ARC0001	0.000000	0.000000
ARC0002	1.000000	0.000000
ARC0003	0.000000	-1.000000
ARC0004	1.000000	0.000000
ARC0005	1.000000	0.000000
ARC0006	0.000000	0.000000
ARC0007	1.000000	0.000000
ARC0008	0.000000	0.000000
ARC0009	0.000000	0.000000
ARC0010	0.000000	6.000000
ARC0012	0.000000	0.000^00
ARC0013	0.000000	0.000000
ARC0014	0.000000	0.000000
ARC0016	0.000000	6.000000
ARC0017	1.000000	0.000000
ARC0018	1.000000	0.000000
ARC0019	0.000000	4.000000
ARC0020	1.000000	0.000000
ARC0021	1.000000	0.000000
ARC0022	0.000000	0.000000
ARC0023	0.000000	6.000000
ARC0024	0.000000	0.000000
ARC0011	1.000000	0.000000
ARC0015	1.000000	0.000000

Figure 4.3c Results of optimization at node 3

Empirically, structures of the type in example 2 pose less problems than those of example 1. This is especially so as the problem gets large. However, the above examples reveal some interesting characteristics of the problem. The reason that most problems solves to integrality is due to the large disparity in cycle arc costs as compared to both ground and service arc costs. That is, having a set of cycle flows with at least one profitable cycle, including more than one arc from each GUB constraint, becomes rare when the cycle arc costs are large. This would explain why Levin[26] in solving the minimum fleet sizing problem (where the ground and service arc costs are zero, and GUB constraints are of the equality type), failed to generate any large problem with non integer LP solutions. The GUB branch and bound addresses this issue directly by selecting infeasible GUB sets to branch, and attempts at reducing alternative fractional cycle flows.

The next observation relates to conditions under which the routing problem with service choices is likely to return fractional LP solutions. This generalizes the properties of examples 1 and 2 and helps explain why, in general, routing problems with tight windows do not pose severe problems. Moreover, it has helped tremendously in our ability to generate interesting problems on which the GUB branch and bound can be tested. Any fractional solution is decomposable into an overlay of fractional flows along a set of cycles. Moreover, at least one of these cycles contain more than one arc belonging to the same GUB set. These are consequences of the network flow and GUB constraints. Thus, an infeasible solution is more likely to occur when the network is distributed in such a way as to permit an overlay of fractional cycle flows. This occurs when the GUB variables are distributed in a manner that makes it possible to form cycles containing more than one arc from the same GUB constraint. This observation can be extended to include the case where each GUB constraint consists of service arc variables distributed across the network, which is the general routing problem with service choices. As suspected, this problem has a greater chance of yielding

fractional solutions since more cycles can be constructed to include more than one variable from the same GUB constraint.

4.1 Results for solutions to routing problems with service choices

LINDO[30] was chosen to solve the LP at each node because it provides sufficient flexibility in a interactive environment. Specifically, it allows for parametrically:

- Adding/deleting rows/ from the current optimization.
- Setting upper bound on variables.
- Changing data on current formulation, for example, altering right hand side values and signs of inequality constraints, and replacing direction of inequallity.

The disadvantages are that LINDO does not provide facilities to include GUB constraints implicitly. This means that the LP basis is larger, and consequently the computational time required will be longer. Moreover, it does not allow the current basis to be stored easily. Therefore, the basis cannot be reinvoked when nodes are evaluated which do not lie along the same path from 0. However, the purpose is to use the LP capabilities of LINDO to test the convergence of the GUB branch and bound in terms of the number of nodes generated. The total number of simplex iterations are also tabulated to provide an approximate measure of the computational requirements. The process can be accelerated using a more sophisticated LP code, if using LINDO shows good convvergence.

Before describing and discussing results on several test problems, the nature of the input data is described. The input to (P) is a series of potential service arcs, where each service contains the following information:

OC	OT	AC	AT	CIJ	NG

where

OC: City of origin

OT: Time of origin

AC: Arrival city

AT: Time of Arrival

CIJ: Contribution (negative value for profit)

NG: GUB set which this service belongs[1]

The inputs are first transformed into a schedule map[2], and then preprocessed in MPSX format for input into LINDO. The MPSX input format is chosen to allow for compatibility since most efficient LP code accept this format. The preprocessing is not described since little insight is gained by doing so. It is stressed, however, that for variables with $NG = 0$, the upperbounds of that variable are included in the formulation, and set to 1. Otherwise, the upperbounds need not be included.

Two classes of problems are generated and tested. They are:

A Routing with time windows (RTW)

where the service choices represent discrete departures over a given time window. No restrictions are placed over the number of arcs posted for each time window

B Routing with service choices (RSC)

where the service choices form mutually exclusive subsets of arcs distributed among the serivce arcs in the network

Three Class A problems are generated, RTWA, RTWB, RTWC. RTWA discretizes each time window to contain 3 departures, where the first and last departures coincides with the bounds of the time window and the third lies in the middle of the bound. RTWB differs with RTWA only in size. For both problems, the time window is kept small. RTWC considers the case where the size time window varies, and the number of departures posted varies

[1]For convenience, $NG = 0$ implies that the given arc does not belong to any GUB constraint.

[2]See Chapter 1

from 2 to 6. In general, more arcs are distributed across a much wider time window. In class A problems, GUB variables from the same constraint are distributed between similiar city pairs. Class B problems are a generalization of class A problems where GUB variables from the same constraint need not include services between similiar city pairs. RSC belong to Class B, where the number of arcs per GUB constraint varies from 2 to 6. The characteristics of the four problems generated are summarized in table 4.1.

Every attempt has been made to ensure that the four problems generated reflect actual medium scale problems. The schedule map for all 4 problems contain 3 cities, where the service arcs lie between 0700 hr to 2200 hr in a 2400 hour period. This correspond to normal hours of operations. Between this time period, the departures are dense and average more than 4 per hour except between 2000 hr to 2200 hr. The ratio of cost coefficients between cycle to service arcs ranges from 2 to 3. This is slightly unrealistic, but the intention here is to underplay the dominance of the cycle arc contribution in order to force fractional LP solutions. The ground arc cost C_i^Q corresponding to ground arc x_i along city axis Q, is calculated as

$$C_i^Q = \left(\frac{t_i^Q}{TR_Q} * OQ \right)$$

where

OQ = cycle arc cost corresponding to city Q.

TR_Q = range of time betwen the first and last time node along any given city axis Q.

t_i^Q = range of time between the incident nodes connected to ground arc x_i along city Q.

That is, the ground arc cost is measured as a proportion of cycle arc cost, depending on the time range it encompasses.

Table 4.1 contains the list of problems tested, with results shown in table 4.2. Many other problems were generated whereby the LP relaxation solves to integrality. These are uninteresting cases which will not be dealt with here.

Problem A	Rows	Columns	(0,1) Variables	Numbers of GUB Set	Variables Per GUB
RTWA	196	291	96	32	3(fixed)
RTWB	227	340	111	38	3(fixed)
RTWC	190	285	96	26	2-6
RSC	191	286	95	26	2-6

Table 4.1 Problem Summary

Problem	z_{LP}	z^*	z_h	E_1	E_2	T_1	T_2	N_1	N_2	I
RTWA	-615	-609 (347)	-606 (9)	0.01	0.005	370	356	2	0	0.50
RTWB	-498	-493 (378)	-493 (73)	0.01	0.000	527	451	2	0	0.50
RTWC	-430	-410 (369)	-410 (26)	0.05	0.000	1400	664	54	10	0.40
RSC	-588	-562 (329)	-502 (505)	0.15	0.110	3227	1526	52	20	0.38

Table 4.2 Results for GUB Branch and Bound

Note:

1. $z_{LP} = z(P^0), z^* = z(p)$
2. $E_1 = |\frac{z_h - z_{LP}}{z_{LP}}|$, $E_2 = |\frac{z^* - z_h}{z_h}|$
3. Values enclosed in parenthesis indicate the number of simplex iterations required.
4. z_h = Value of feasible solutions obtained from using the sum of vehicle size constraints.
5. T_1 = Total number of simplex iterations to find an optimal solution.
6. T_2 = Total number of simplex iterations to find a solution with 2 % of optimality.
7. N_1 = Total number of nodes generated to find optimal solutions.
8. N_2 = Total number of nodes generated to find solutions within 2 % of optimality.
9. I = Fraction of leaf nodes that are integer.

In all 4 problems, the heuristic solution obtained by using vehicle size cuts produce good solutions. In two problems, optimal solutions were obtained, though not verified. However, in one case, RSC, the heuristic solution required more simplex iterations than those required to solve to original LP. In general, the addition of new constraints, requires the extension of the basis, and many more simplex iterations. Thus, it was chosen to bound the variable to zero by setting a variable to zero rather than to include a constraint setting the variable to zero. From the inititial tests, it appears that for problems with tight time windows, like RTWA and RTWB, the GUB branch and bound is extremely effective. This is expected, since branching reduces the formation of alternative fractional cycle flows since variables belonging to similar GUB constraints are effectively removed. Moreover, the number of cycles in the network containing more than one variable for such problems are restricted. As expected, RTWC and RSC are more cumbersome, although the number of nodes generated for problems of this size are not considered large. It is also key to emphasize that a solution within 2% of the optimal requires the generation of only a fraction of the total nodes. For large problems, it might be advisable to seek near optimal solutions using the ideas developed in chapter 3. It is most interesting that for all the problems tested, the number of leaf nodes which have integer solutions are at least 38% of total nodes. This supports experience that these problems have strong tendencies to solve to integrality and the GUB branch and bound is effective in finding feasible solutions. Although more than 50 nodes are generated for RTWC and RSC, the total number of simplex iterations, T_1 spent (which includes the initial LP solution and feasible search), is small. This results from the fact that the number of iterations required to reoptimize the modified LP problem at each node are small, that is on the order of less than 10% of that to solve the LP, on the average. It is not surprising that of the four problems tested, RSC requires the most iterations. This is because variables belonging to the same GUB constraints are distributed

across all service arcs. Breaking fractional cycle flows via GUB branch and bound has a lesser effect on the potential of other cycles being formed with variables belonging to the same GUB constraints.

The cost structures of the 4 problems tested are intentionally construed so that the LP yields fractional solutions. This is achieved by enforcing less variation between the cycle arc costs and the service arc costs, assuming that the ground arc costs are minimal in a dense network. For most practical problems, the cycle arc cost is much more substantial in comparison to other arc costs. In that event, the occurrance of fractional solutions can usually be traced to some small subset of GUB constraints whose variables are responsible for encouraging fractional solutions. For this case, it is recommended that those set of GUB constraints are selected first to effect the branching at each node should they become infeasible. This is sometimes referred to as priority based set selection. And in effect, GUB set selections are modified to start with those having high priorities.

Initial tests using GUB branch and bound on problems with service choices appears promising. More tests are required to tailor the algorithm to meet real life applications. In the next section, we describe the mechanics for automating the GUB branch and bound search by keeping track of the variables that are being processed during branching and backtracking are described.

4.2 Implementation Related Issues

Thus far, the GUB branch and bound is applied to test problems in a semi interactive environment. Although is is useful to maintain some level of user interaction in a branch and bound scheme, the basic search structure must be automated. The GUB branch and bound differs from single variable branch and bound in that at least one variable is fixed or freed during a branch or backtrack. As such, the usual tree tracking schemes cannot be directly applied. In what follows, a procedure for tracking the branch and bound is described to update the status of the variables that are fixed or free at each node. The discussion assumes the use only one working basis.

Tracking the status of variables at each node

Balas introduced a clever method for the single variable branch and bound that stores the branching information in a vector, whose size equals to the number of $0-1$ variables. This can be done because only one variable is processed along an arc. For the GUB branch and bound, at least one variable must be processed along each arc. That is, at least one variable might be set to zero or one during each branch or backtrack respectively. Thus, the original scheme must be modified to accomodate this difference.

To highlight the issues involved in the tracking, consider the following example. For simplicity, assume that only one GUB constraint, $x_1+x_2+x_3+x_4 \leq 1$, is imposed. Initially, the LP at node 0 is solved. Assume that node 0 is not fathomed. Two subproblems are created corresponding to nodes 1 and 2 by setting $x_1 = x_2 = 0$ and $x_3 = x_4 = 0$ respectively. Suppose node 1 is evaluated next. This is done by fixing the variables $x_1 = x_2 = 0$, and reoptimizing. Fixing a variable means setting its upperbound to zero. To evaluate node 2, x_3 and x_4 are fixed next, but x_1, and x_2 must be freed. Freeing a variable means setting its upperbound to 1. Thus, it is important to know which variables are fixed and freed at

each node prior to optimizing. The other issue relates to GUB set partition, to form two subproblems. At node 1, for instance, the GUB set selection phase selects the same GUB set for partitioning. It is important to partition only those variables in the set that are not fixed. Otherwise, redundant partitions can occur. Thus, knowing that $x_1 = x_2 = 0$ are imposed at node 1, only x_3 and x_4 are left for partitioning. The following discussions describe a method for resolving the problems.

At node k, an LP is solved. Node k is included in the list of live nodes if it is not fathomed. The following information, derived at node k are stored:

Node k:	Z_k	LL_k	LR_k	L_k

where

- z_k = integer part of optimal LP value at node k.
- L_k = list of variables fixed at zero at node k.
- LL_k = List of additional variables to be fixed at zero when the left child of node k is evaluated.
- LR_k = List of additional variables to be fixed at zero when node the right child of node k is evaluated.

L_k is used to ensure that LL_k and LR_k are created correctly as follows:
Let n_i be the set of variables belonging to the GUB set i chosen at node k. Let $Q_i^k = n_i/L_K$ be the variables in the selected GUB set i, not fixed.[3] Partition Q_i^k into 2 subsets using the methods described in the last chapter. One set is stored as LL_k and the other as LR_k, where $LL_k \cup LR_k = Q_k^i$, $LL_k \cap LR_k = 0$.

Note that subproblems corresponding to children of k differ from that at node k only in the additional variables that are required to be fixed. These variables are found after

[3]For sets A, B, $A/B = A - (A \cap B)$.

node k is solved, and included in sets LL_k and LR_k. LL_k, LR_k, and L_k are stored in order to avoid recomputing node k later when it is selected as the live node to effect branching. Thus, the variables fixed at left child and right child of node k, are obtained from $L_k \cup LL_k$ and $L_k \cup LR_k$, respectively. Having evaluated the child of node k, all information pertaining to node k is purged to better manage the storage requirements.

The basic idea of the tracking scheme is simple. Consider the branch and bound subtree in figure 4.5, where the arcs connecting nodes represent paths containing at least one node. Assume that node j is the last node processed, and the next node evaluated is k. To solve node k from j, the variables along path i to k are fixed; and those along path i to j are freed.

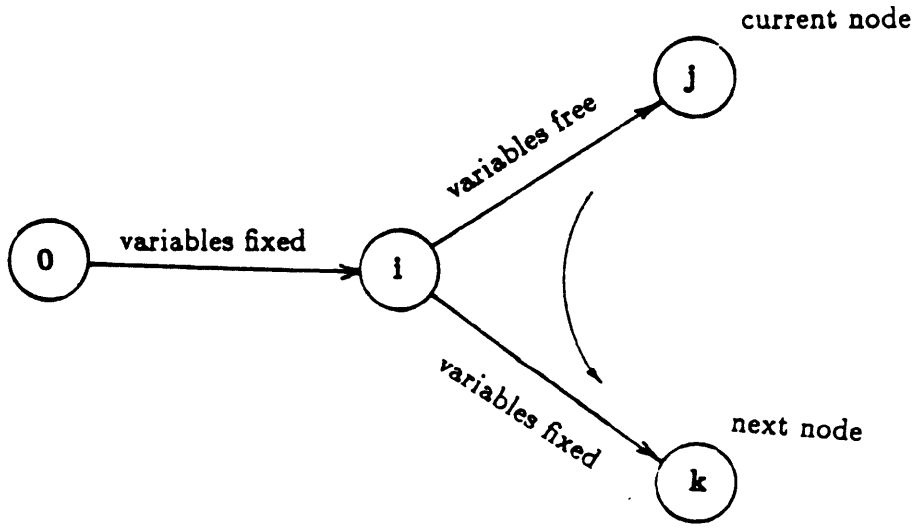


Figure 4.5 Variable tracking

In what follows, a method is described to effect this process without explicitly identifying node i. Suppose L_j represents the variables fixed at node j, which is the last node processed. Let k be the next node solved. L_k includes variables that are fixed at node k. The variables that must be freed at k (corresponding to those lying along path i to j) are included in $E_k = L_j/L_k = L_j - (L_j \cap L_k)$. Assume that $E_0 = \{0\}$.

To illustrate how these ideas are implemented, return to the previous example. Results for 4 nodes only are shown. Let nodes 1 and 2 be the children of 0, and nodes 3 and 4 be the children of node 1. The nodes are processed in the order 0, 1, 2, 3, 4. Since there is only one GUB constraint $n_1 = \{x_1, x_2, x_3, x_4\}$. The results are shown below.

Node 0: Variables fixed = $L_0 = \{0\}$

Variables freed = $E_0 = \{0\}$

Solve. $Q_1^0 = n_1/L_0 = \{x_1, x_2, x_3, x_4\}$

$LL_0 = \{x_1, x_2\}, LR_0 = \{x_3, x_4\}$

Node 1 : Variables fixed = $L_1 \cup LL_0 = \{x_1, x_2\}$ since node 1 is child of 0.

Variables freed = $E_1 = L_0/L_1 = \{0\}$

Solve. $Q_1^1 = n_1/L_1 = \{x_3, x_4\}$

$LL_1 = \{x_3\}, LR_1 = \{x_4\}$

Node 2 : Variables fixed = $L_2 = L_0 \cup LR_0 = \{x_3, x_4\}$ since node 2 is a child of 0.

Variables freed = $E_2 = L_1/L_2 = \{x_1, x_2\}$

Solve, $Q_1^2 = n_1/L_2 = \{x_1, x_2\}$

$LL_2 = \{x_1\}, LR_2 = \{x_2\}$

Node 3 : Variables fixed = $L_3 = L_1 \cup LL_1 = \{x_1, x_2, x_3\}$ since node 3 is a child of node 1.

Variables freed = $E_3 = L_2/L_3 = \{x_4\}$

Solve. Fathomed.

Node 4 : Variables fixed = $L_4 = L_1 \cup LR_1 = \{x_1, x_2, x_4\}$

Variables freed = $E_4 = L_3/L_4 = \{x_3\}$

Solve. Fathom.

The above discussion shows how the tracking scheme is accomplished by keeping a list of variables that are fixed at each node. This list is derived by finding the partitions at each

subproblem optimization. The tracking can thus be easily incorporated into any branch and bound procedure.

Chapter summary

In this chapter, the GUB branch and bound described in Chapter 3 is implemented on a set of test problems. For problems where the service arcs are distributed in such a way that few cycles can be traced containing more than one arc from the same GUB set, the method appears to work very well. This is the case for the routing problem with small time windows. The service arcs having fractional solutions more closely identify those bundle arcs that cause the infeasibilities. In the more general case, the results are also encouraging, since many nodes are fathomed through being feasible. As such, the GUB branch and bound appears successful in breaking fractional cycle flows causing integral cycle flows to be found. Because many feasible nodes are found in the branch and bound, the method can be terminated prematurely to obtain good solutions. The efficiency of the GUB branch and bound is aided by the good solutions obtained through the heuristic for generating feasible solutions using fleet sizing cuts. However, it is felt though that the lower bounding objective cut enhancement idea is not as effective in the preliminary stages of the branch and bound, and only becomes effective when the objective values of the live nodes lie close to that of the incumbent. In general, as the cycle arc costs dominate over other arc costs, the LP solutions for these problems have strong tendencies to yield integer solutions. The results obtained are far better than the ones obtained using decomposition methods described in Section 3.1.

This thesis focuses on a specific routing problem with service choices formulated as a network flow problem with GUB constraints. The next chapter considers extensions of the basic routing problem with service choices and points to related areas of interest.

Chapter 5

Conclusion

The previous chapters show how the routing problem with service choices formulated as network flow problems with GUB constraints can be effectively solved using the GUB branch and bound. In this chapter, it is shown how the model can be extended to consider routing problems with varying fleet types. Furthermore, several interesting extensions are also included. This chapter concludes by summarizing the lessons and experiences gained through this exercise.

5.1 Extensions

The routing problem considered in this thesis assumes a homogeneous fleet. In some situations, this assumption is not adequate. It is shown here how the multi-fleet routing problem can be formulated as a multicommodity flow problem with GUB constraints. Only two types of vehicles are considered. The extension to the multi-fleet case can be easily deduced. A new set of decision variables are now defined as follows:

$$X_{ij}^{k} = \begin{cases} 1 & \text{if aircraft type } k \text{ is used on arc } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

Moreover, the constraints of the models are defined as:

U_{ij}^{k} = Capacity of aircraft type k along arc (i,j)
C_{ij}^{k} = Per unit cost/benefit associated with sending aircraft k on arc (i,j)

Letting S be the set of service arcs, and N, A be the node and arc set, respectively. The multi-fleet routing problem (MFP) is formulated as follows:

$$
MFP \left\{
\begin{array}{rll}
\text{minimize} \sum_{(i,j)\in A} C^1_{ij} X^2_{ij} + \sum_{(i,j)\in A} C^2_{ij} X^2_{ij} & & \\
\sum_{i\in N} X^1_{ij} - \sum_{i\in N} X^1_{ji} = 0 & \forall\ j \in N & (5.1) \\
\sum_{i\in N} X^2_{ij} - \sum_{i\in N} X^2_{ji} = 0 & \forall\ j \in N & (5.2) \\
\sum_{k=1}^{2} X^k_{ij} \le 1 & \forall\ (i,j) \in S & (5.3) \\
0 \le X^k_{ij} \le U^k_{ij} & \forall\ (i,j) \in (A-S),\ k = 1,2 & (5.4) \\
X^k_{ij} \in \{0,1\} & \forall\ (i,j) \in S,\ k = 1,2 & (5.5)
\end{array}
\right.
$$

Constraints (5.1) and (5.2) are the conservation of flow constraints. (5.3) enforces arc (i,j) to be served by only 1 vehicle type, and (5.5) ensures the use of integral vehicles. The arc capacities are represented by constraint (5.4). This problem is a simplification that does not include service choice constraints. More appropriately, it considers the special case where each service arc represents a service choice. Initial attempts to solve this problem are given by Sadiq[29] using decomposition methods. It is interesting to see how a GUB branch and bound can be embedded in the decomposition to obtain good solutions.

Thus far, the arcs being bundled have upperbounds of 1. The above analysis is easily extended to include bundling arcs having bounds greater than one. Since the bound on the GUB set is 1, all variables with an upperbound greater than 1 are replaced by an upperbound of one, because its flow is restricted to at most one by the GUB constraint.

The GUB constraints dealt with in this thesis belong to a broader class of constraints of the form $x_1 + x_2 + \cdots, + x_m \le k$, where $k < m$. It is interesting to develop an efficient branching scheme for the case where $k > 1$. In some cases, like the special crew scheduling problem described in Chapter 1, the number of crews stationed at each base might be bounded by 5, for which case $k = 5$.

In what follows, a simple branch and bound scheme is described for handling constraints

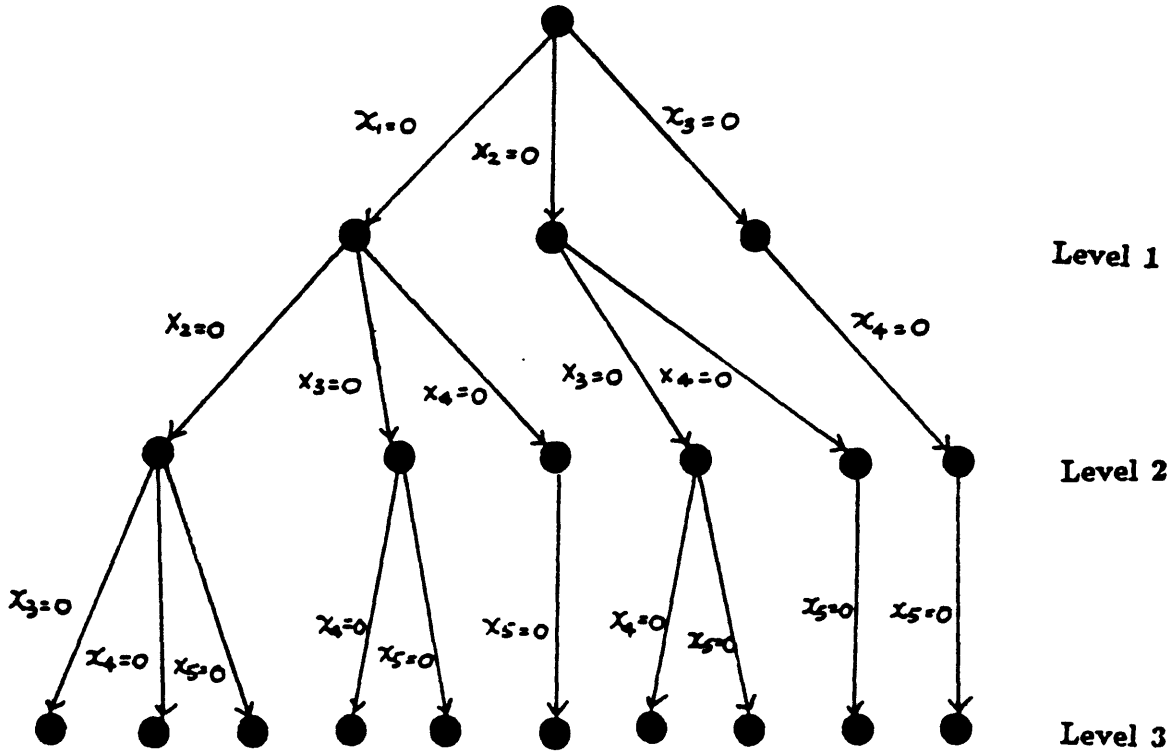


Figure 5.1 Alternative Tree For $x_1 + x_2 + x_3 + x_4 + x_5 \leq 2$

of the type $x_1 + x_2 + \cdots, + x_m \leq k$, where $k > 1$. For simplicity, assume that only one such constraint exists in problem (P). The method is described via a simple example given as follows.

Suppose the given constraint is $x_1 + x_2 + x_3 + x_4 + x_5 \leq 2$. Then, all possible solutions can be found by successively setting sets of three variables to zero. Since there are a total of ${}^5C_3$ such sets, a tree can be constructed that has ${}^5C_3$ arcs where each arc restricts flow on three variables to zero.[1] The terminal nodes are feasible since the above constraint, with 3 variables set to zero, is redundant because all the variables have bounds [0,1] and the network flow constraint is totally unimodular. Such a tree is somewhat restrictive, containing nodes with a large number of adjacent arcs. A better tree representation is given below in figure 5.1.

[1] ${}^nC_r$ reads n choose r.

The characteristics of the tree above are listed as follows:

1. There are $m - k$ levels.

2. At level j, the index of the last arc eminating from each node is $m - k + j$.

3. The index of the leftmost arc from each node is greater than the previous arc in the same path by one.

Such a tree structure imposes less restrictions at each subproblem optimization and can conceivably yield faster convergence. Note, however, that the arcs are ordered in a specific manner so that the left subtrees are denser than the right ones. It is thus imperative to select the appropriate variable as x_1 so that setting its value to zero results in a good chance of fathoming.

The analysis above considers integer variables whose flow is bounded by $[0, 1]$. It is interesting to find out how the routing problem with service choices, or the network flow problem with GUB constraints bounded by k, can be solved if the integer variables they contain are bounded by $[0, u]$ for $u > 1$. It is unclear, at this point, how such problems can be solved efficiently. There are reasons to believe, in some cases, that if u is large, rounding off solutions to LP relaxations would not result in unacceptable infeasibilities.

5.2 Summary

The routing problem with service choices is modelled as a network flow problem with side constraints. These constraints restrict flows on bundles of arcs having bounds $[0, 1]$ to 1. As such, they are often referred to as GUB constraints. The variables in these constraints are further restricted to take on integral values.

Initial attempts to solve these problems utilized decomposition principles aimed at exploiting the network flow substructure. To this end, Lagrangian and Bender's decomposition

techniques were used. Unfortunately, the convergence results using these procedures were far from satisfactory. Since the Lagrangian subproblem has the integrality property, the best solution obtianed from the Lagrangian relaxation is, at best, as good as that provided by the LP relaxation. Resolving the duality gap involves addressing the combinatorial properties of the problem. In the case of Benders decomposition, the master problem is an integer programming problem which contains GUB constraints. Thus, the computational savings derived from solving the network flow subproblem is offset by the time required to solve the master problem. Moreover, the time required to obtain approximate solutions from the LP relaxation to the master problem increases drastically as the decomposition progresses. What has become immediately obvious, is that in order to find the optimal solution, the discreteness of (P) has to be addressed.

The GUB branch and bound was chosen to solve this problem for several reasons. Although the inclusion of GUB constraints makes the problem theoretically intractable, the LP relaxation solves to integrality for many problems generated. These were realized when many routing problems with service choices are generated and solved. Thus, there are ample reasons to believe that, in most cases, the LP relaxations provide a tight bound to the original problem. The GUB branch and bound scheme was used because the GUB constraints are implicitly satisfied as the branching progresses. It makes use of the property that, at most, one variable in each GUB constraint is basic at any feasible or optimal solution. This method offers tremendous savings over the single variable branch and bound in terms of expected computational, as well as worst case performance. While each subproblem is solved using conventional LP code, additional computational efficiency can be expected if it is solved using a specialized LP code which handles GUB constraints implicitly. Much research into solving LP with GUB constraints has resulted in tremendous improvements.

When the LP relaxations failed to yield integer solutions, the GUB branch and bound

is shown to do very well, for several medium sized problems. The results of these tests are included in chapter 4. Problems which yield integer solutions to LP relaxations are not included.

The success of the GUB branch and bound is closely related to the nature of the LP solution. The LP solution, when infeasible, defines a set of cycle flows, at least one of which is fractional. These are consequences of the conservation of flow and GUB constraints. It is shown in Chapter 4, that the chances of obtaining infeasible solutions increases as the integer arcs belonging to the same GUB constraints are distributed so that many cycles can be easily formed which include arcs from the same set. GUB branching effectively seeks to break these fractional cycle flows by forcing subsets of fractional variables in a GUB set to zero, sequentially. Naturally, this might result in another set of infeasible cycle flows. However, this has not occurred in most cases. This results from the fact that, in many problems, only a small subset of arcs are responsible for encouraging an optimistic fractional flow pattern. The GUB branch and bound is quick to identify these subsets, and then purge them from the network.

The performance of the GUB branch and bound is measured by the number of nodes generated to find the optimal solution. An estimate of the computational time requirements can be deduced from the number of simplex iterations required. The efficiency of the branch and bound is enhanced by a parametric heuristic procedure to find a good solution prior to branch and bound. The heuristic relies on the idea that the cycle arcs behave like bottlenecks on the flows through the network. As such, controlling the sum of cycle flows provide enough penalty to enforce feasibility in many cases. As shown in Chapter 4, this method is responsible for finding feasible solutions very close to the optimal.

In order to efficiently apply the GUB branch and bound on large scale routing problems with service choices, the problem size needs to be reduced. In Chapter 3, a network aggre-

gation scheme is formalized to reduce the size of the problem. The aggregation exploits the temporal properties of the schedule map network. Better lower bounding procedures are also presented to help in pruning the search for an optimal solution in Chapter 3.

While the success with GUB branch and bound is demonstrated for specific problems in routing problems with service choices, these experiences can provide useful clues to other applications that can be formulated as network flow problems with GUB constraints. It also adds to the list of attempts at solving network flow problems with side constraints and mixed integer programming problems with GUB constraints.

Bibliography

[1] R. D. Armstrong, P. Sinha and A. Zoltners, The multiple choice nested knapsack model. *Management Sci.* **28**, 34-43 (1982).

[2] E. Balas and C. H. Martin, Pivot and complement-a heuristic for 0-1 programming. *Management Sci.* **26**, 86-96 (1980).

[3] J Balintfy and C. Blackburn, General purpose multiple choice programming. Graduate school of Business Administration report, Tulane university (1979).

[4] R. Barkley, An optimal solution to an aircraft routing problem with multiple departure times. M.I.T., S.M. thesis, (1968).

[5] E. M. L. Beale and J. A. Tomlin, Special facilities in a general mathematical system for nonconvex problems using ordered sets of variables. Proc. Fifth Int. Conf. on O. R., 447-454.

[6] J. Bean, A Lagrangian algorithm for multiple choice integer programs. *Ops.Res.* **5**, 1185-1193 (1984).

[7] D. Benbasset, Minimal aircraft flows in a schedule network with bundles. M.I.T., M.S. thesis, (1970).

[8] L. Bodin, B. Golden, A. Assad and M. Ball, Routing and scheduling of vehicles and crews. *Computers and OR* **10**, (1983).

[9] S. G. Chang and D. W. Tcha, A heuristic for multiple choice programming. *Comput. Ops. Res.* **12**, 25-37 (1985).

[10] S. Chen and R. Saigal, A primal algorithm for solving a capacitated network flow problem with additional linear constraints. *Networks* **7**, 59-79 (1977).

[11] G. B. Dantzig and R. M. Van Slyke, Generalized upper bounding techniques. *J. Comput. System Sci.* **1**, 213-226 (1967).

[12] J. Desrosier, F. Soumis, and M. Desrochers, Routing problem with time window by column generation. *Networks* **4**, 59-79 (1984).

[13] M. Fisher, The lagrangian relaxation method for solving integer programming problems. *Management Sci.* **27**, 1-18 (1981).

[14] J. J. H. Forrest, J. P. H. Hirst and J. M. Tomlin, Practical solution of large mixed integer programming problems with UMPIRE. *Management Sci.* **20**, 736-773 (1974).

[15] R. S. Garfinkel, Branch and bound for integer programs. Combinatorial Optimization, edited by N. Christofides, A. Mingozzi, P. Toth and C. Sandi, 1-20 (1978)

[16] J. Gauthier and G.Ribiere, Experiments in mixed integer linear programming using psuedo-costs. *Math. Program.* **12**, 26-47 (1977).

[17] F. Glover, Convexity cuts for multiple choice problems. *Discrete Maths* **6**, 221-234 (1973).

[18] Glover and Klingman, Mathematical programming models and methods for the journal selection problem. *Ops.Res.* **21**, 144-155 (1973).

[19] F. Glover and J. Mulvey, Network relaxation and lower bound for multiple choice problems. *INFOR* **20**, 385-393 (1982).

[20] M. D. Grigoriadis, A dual generalized upper bounding technique. *Management Sci.* 5, 1-18 (1971).

[21] J. K. Hartman and L.S. Lasdon, A generalized upperbounding method for multicommodity flow problems. *Networks* 1, 333-354 (1972).

[22] W. C. Healy, Multiple choice programming. *Ops.Res.* 12, 122-138 (1964).

[23] R. Heldt, A solution algorithm for the aircraft routing problem with multiple departure times. S.M. thesis, M.I.T. (1969).

[24] L. S. Lasdon and R. S. Terjung, An efficient algorithm for multi-item scheduling. *Ops. Res.* 19, 998-1022 (1971).

[25] L. Lasdon, Optimization theory for large systems. (1970).

[26] A. Levin, Some fleet routing and scheduling problems for air transportation systems. M.I.T., FTL report no R68-5, (1969).

[27] T. Magnanti, Combinatorial Optimization and vehicle fleet planning: Perspectives and prospects. *Networks* 11, 179-213 (1981).

[28] P. Mevert and U. Suhl, Implicit enumeration with generalized upper bounds. *Anns. Discrete Maths* 1, 392-402 (1977).

[29] G. Sadiq, Multifleet routing problem. M.I.T., M.S. thesis, (1978).

[30] L. Schrage, Linear Interactive Discrete Optimizer(LINDO). University of Chicago (1982).

[31] J. Shapiro, A survey of lagrangian techniques for discrete optimization. M.I.T., Technical Report no 133, OR Center (1977).

[32] R. Simpson, Scheduling and routing models for airline systems. M.I.T., FTL report no. R68-3, (1969).

[33] P. Sinha and A. Zoltners, The multiple choice knapsack problem. *Ops. Res.* **27** (1973).

[34] D. J. Sweeney and R. A. Murphy, Branch and bound method for multi-item scheduling. *Ops. Res.* **29**, 853-864 (1981).

[35] J. A. Tomlin, An improved branch and bound method for integer programming. *Ops. Res.* **19**, 1070-1074 (1971).

[36] R. D. Young, Hypercylindrically-deducted cuts in zero-one integer programs. *Ops. Res.* **19**, 1393-1405 (1971).

Appendix A

Branch and Bound

This appendix presents a review of branch and bound for the minimization problem, and defines related terms used in this thesis.

Definitions

1. A Search tree is an acyclic directed tree with a root node from which all nodes can be reached via a unique path. Let 0 be the index of the root node.

2. Node j is referred to as a descendant of node i, if a path exists from i to j. A node with no descendant node is called a terminal node.

3. A node i is called a parent of j if there is arc (i,j) which connects i to j. j is called the child of i. Except for node 0, each node has a unique parent, but in general more than one child.

4. For an optimization problem $\bullet$, $F(\bullet)$ denotes its feasible solutions.

5. For an optimization problem S^i, the optimal objective value is v^i.

6. Let P^i be a relaxation of S^i, whose optimal objective value is z_i and $z_i \leq v^i$. The solution of P^i need not be feasible in S^i.

Given an optimization problem, S, a series of problems $S^0, S^1, \cdots, S^k$ are generated such that they include all possible solutions of S, or $F(S^0) \cup F(S^1) \cdots, \cup F(S^k) = F(S)$.

The optimal solution to S is the best solution found by solving S^i for $i = 0, 1, \cdots, k$. If each optimization subproblem corresponds to a node in the search tree, then the tree is referred to as a direct search tree. The direct search procedure involves evaluating all nodes in the direct search tree in order to find the optimal solution. If, in addition, the children of i determines a set of subproblems S^j, $j \in$ child of i, such that $F(S^i) \supseteq F(S^j)$ and $\cup_{j \in \text{ child of } i} S^j = S^i$ for all i, then the tree is referred to as a branch and bound tree. Each of the arcs emanating from node i corresponds to a constraint restricting S^i to a subproblem S^j. Although not necessary in many cases, S^j, $j \in$ child of i, is a partition of S^i. The collection of subproblems S^j is known as a separation of S^i. Note that $F(S) = F(S^0)$. Finding the optimal solution in a branch and bound tree constitutes the discussion which follows.

Finding The Optimal Solution In A Branch And Bound Tree

In general, the optimization problem S is difficult to solve. As a result, S^i is also difficult. Thus, at each node i, a relaxation P^i is solved instead. As an example, if S^i is an integer programming problem, P^i can be selected as its LP relaxation. The following rules relate to finding an optimal solution to S by solving a series of problems P^i corresponding to S^i at node i. The subproblems S^i generated, must ensure that solving P^i finds a feasible solution that is optimal in S . One way to effect this is to make sure that solving P^i at the terminal nodes finds all feasible solutions to S.

Let z^{best} be the best feasible solution known for S. z^{best} is called the incumbent solution. A node i is said to be fathomed if evaluating nodes j which are descendants of i, will not yield a better solution than what is available. Node i is fathomed if:

1. P^i is feasible.

2. $z_i \geq z^{best}$.

If P^i is feasible, then all nodes j which are descendants of i, need not be evaluated because $z_i = v^i \leq v^j$ for all j. $v^i \leq v^j$ because $F(S^i) \supseteq F(S^j)$. z^{best} is updated if $z_i < z^{best}$.

Condition 2 includes the case when P^i is infeasible. In this case, $z_i = \infty$. Furthermore, if $z_i \geq z^{best}$, $v^i \geq z_i$ and $v^i \geq z^{best}$. Since any descendant node j yields $v^j \geq v^i$, they need not be considered.

The set of nodes that are not fathomed are called the set of live nodes. A better incumbent obtained through fathoming via condition 1 is used to fathom other nodes in the set of live nodes. This process is called killing live nodes. The quality of the relaxation or lower bounding determines how effective nodes can be killed.

Based on the information provided above, a branch and bound algorithm is provided as follows:

Branch and Bound algorithm

Step (0): Initialization: Begin at live node 0, $z^{best} = \infty, z^0 = -\infty$. Go to Step (1).

Step (1): Select a live node i, if none, go to (6). If P^i is solved, go to (2), else, go to (3).

Step (2): Branching: Choose a separation of S^i which determines the childs of node i. Put all childs of i in set of live nodes. Remove i from list of live nodes. Go to (1).

Step (3): Solve P^i. If P^i is unbounded, $v^i = -\infty$ go to (1). Otherwise go to (4).

Step (4): Fathoming (Condition 1). If solution to P^i is not feasible in S^i, go to (5). Otherwise, $z_i = v^i$ and i is fathomed, remove i from list of live nodes. Let $z^{best} = min\{z^{best}, z_i\}$. Go to (5).

Step (5): Fathoming (Condition 2). Any node i, such that $z_i \geq z^{best}$ is fathomed, remove i from list of live nodes.

Step (6): Termination. If $z^{best} = \infty$, no solution exists. If $z^{best} < \infty$, feasible solution corresponding to z^{best} is optimal.

Issues Relating to Branch and Bound

The efficiency of the branch and bound depends on:

1. Node Selection (Step 1).

2. Branching (Step 2).

The live node selected should correspond to one that yields the greatest potential of fathoming. Fathoming by condition 2 can result in an improved incumbent, whereby other live nodes can be killed (Step 5). Branching at node i involves 2 steps, selecting the appropriate restrictions for S^i, and forming subproblems based on such restrictions. For example, in single variable 0,1 branch and bound, a fractional variable is selected, whereby 2 subproblems are created, one corresponding to setting the variable to zero, and the other to one. Variable selection has significant impact on the efficiency of the branch and bound. Extensive experiemental rules for node and variable selection for the single variable branch and bound can be found in Forrest [14] and Gauthier [16].

Appendix B

Solving LP with GUB constraints

This appendix shows how the LP relaxation of the routing problem with service choices can be solved efficiently.

Method of Dantzig And Van Slyke

For ease of exposition, consider the LP relaxation of the the routing problem with service choices, ignoring the upperbounding on variables $X_v \leq U_v$. The resultant problem is,

$$\begin{aligned} \min \quad & C_v X_v + C_s X_s \\ & N_v X_v + N_s X_s = 0 \quad \text{(m rows)} \\ & B X_s + I_Y = 1 \quad \text{(p rows)} \\ & (X_v, X_s, Y) \geq 0 \end{aligned}$$

after addition of slacks variables Y, to convert the GUB constraints to equality form. Hereafter, I refer to the identity matrix. Assume that there are m conservation of flow constraints and p GUB constraints. The important thing to note is that each element in the last p rows is zero or one, and each column of these rows contains at most one non-zero entity. Due to this special structure, an (m+p) $\times$ (m+p) basis matrix β can be constructed so that a p $\times$ p identity submatrix I appears in the lower right corner. R, $S$and T have conformal

dimensions. That is, $\mathcal{B}$ can be written as:

$$\mathcal{B} = \begin{bmatrix} R & S \\ T & I \end{bmatrix} \begin{matrix} \text{m rows} \\ \text{p rows} \end{matrix}$$

Now, solving the LP with the above basis matrix involves iteratively solving the following 2 systems of equations:

1. $u\mathcal{B} = C_{\mathcal{B}}$

2. $\mathcal{B}d = a$

where,

- $C_{\mathcal{B}}, C_{\mathcal{N}}$ = cost coefficients of basic and nonbasic variables
- a = column of nonbasic matrix $\mathcal{N}$

Equation 1 is used to establish optimality by finding the simplex multipliers u, and checking for dual feasibility, $u\mathcal{N} \leq C_{\mathcal{N}}$, for the given primal basis $\mathcal{B}$.

Equation (2) is used to select the basic variable to leave, given that a non basic variable corresponding to a is introduced. (2) expresses a in terms of the basis vectors, to find weights d. It is also used to check for unboundedness of solutions. Typically, the matrix $\mathcal{B}^{-1}$ is used to solve equations (1) and (2) and gets updated[1]. The main idea here is that the working basis $\mathcal{B}^{-1}$ is of size (p+m) $\times$ (p+m). In what follows, it is shown, how an m $\times$ m working bais can be derived to find u and d.

Define a (p+m) $\times$ (p+m) lower triangular matrix L,

$$L = \begin{bmatrix} I & 0 \\ -T & I \end{bmatrix}$$

where I is a p $\times$ p identity matrix. Then,

$$\mathcal{B}L = \begin{bmatrix} Z & S \\ O & I \end{bmatrix}$$

[1] $u\mathcal{B} = C_{\mathcal{B}} \Rightarrow u = \mathcal{B}^{-1}C_{\mathcal{B}}$
$\mathcal{B}d = a \Rightarrow d = \mathcal{B}^{-1}a$

where $Z = R - ST$. Note that Z is a m $\times$ m matrix. It is now shown how Z^{-1} can be used to derive u and d. For convenience, let (y^m, y^p) denote the first m and last p components of vector y.

Finding u

From equation 1 :

$$uB = C_B$$

$$\Rightarrow uBL = C_B L$$

$$\Rightarrow (u^m, u^p) \begin{bmatrix} Z & S \\ O & I \end{bmatrix} = ((C_B)^m, (C_B)^p) \begin{bmatrix} I & O \\ -T & I \end{bmatrix}$$

$$\Rightarrow u^m Z = (C_B)^m - (C_B)^p T$$

$$u^m S + u^p = (C_B)^p$$

$$\Rightarrow u^m = [(C_B)^m - (C_B)^p T] Z^{-1}$$

$$u^p = (C_B)^p - u^m S^1$$

Thus, $u = (u^m, u^p)$ can be calculated knowing Z^{-1}.

Finding d

To find d, first find $w = (w^m, w^p)$ by solving the following system of equations:

$$(BL)w = a \quad (3)$$

Then, by substitution,

$$d = Lw \quad (4)$$

since

$$Bd = a$$

Expanding equation (3):

$$\begin{bmatrix} Z & S \\ O & I \end{bmatrix} \begin{pmatrix} w^m \\ w^p \end{pmatrix} = \begin{pmatrix} a^m \\ a^p \end{pmatrix}$$

$$\Rightarrow \begin{aligned} Zw^m + Sw^p &= a^m \\ w^p &= a^p \end{aligned}$$

$$\Rightarrow \begin{aligned} w^m &= Z^{-1}[a^m - Sw^p] \\ w^p &= a^p \end{aligned}$$

Now, Equation (4) implies that

$$\begin{aligned} d^m &= w^m \\ d^p &= w^p - Tw^m \end{aligned}$$

therefore,

$$d^m = Z{-}1[a^m - Sw^p]$$

$$d^p = a^p - Td^m$$

Therefore, $d = (d^m, d^p)$ can be determined from Z^{-1}. Note that since there is one redundant constraint in the conservation of flow constraints, the size of the basis Z^{-1} is equal to the number of nodes minus 1.

Method of Dantzig Wolfe

The following LP:

$$\min\ C_v X_v + C_s X_s$$

$$N_v X_v + N_s X_s = 0$$

$$B X_s \leq 1$$

$$0 \leq X_v \leq U_v$$

$$0 \leq X_2 \leq 1$$

is solved using Dantzig Wolfe decomposition, treating the network flow constraints:

$$N_v X_v + N_s X_s = 0$$

$$0 \leq X_v \leq U_v$$

$$0 \leq X_s \leq 1$$

as the subproblem constraints, and

$$B X_s \leq 1$$

as the complicating constraints.

Suppose, K subproblem proposals $\{(X_v^1, X_s^1), \cdots, (X_v^K, X_s^K)\}$ are available. (X_v^i, X_s^i) for $i = 1, 2, \cdots, K$, satisfies network flow constraints.

Before presenting the pair of subproblem and restricted master problem, a few terms are defined.

Let,

$$P_J = C_v X_v^J + C_s X_s^J \quad J = 1, 2, \cdots, K$$

$$r_i^J = B_i X_s^J \quad i = 1, 2, \cdots, p$$

where B_i denotes the i^{th} row of B and p the number of rows in B. Then P_J denotes the contribution of proposal J, and r_i^J the consumption of the ith resource using proposal J.

The restricted master corresponding to K problem proposal is $(DW)^K$ expressed as:

$$(DW)^K \begin{cases} \min \sum_{J=1}^{k} P_J \lambda_J & \text{Dual variables} \\ \sum_{J=1}^{K} r_i^J \ \lambda_J \leq 1 \quad \text{for } i = 1,2,\cdots,p & \Pi_i^K \\ \sum_{J=1}^{K} \lambda_J = 1 & \sigma_K \\ \lambda_J \geq 0 \quad J = 1,2,\cdots,K & \end{cases}$$

where Π_i^K, σ_K are dual variables corresponding to the i^{th} resource constraint λ, and the convexity constraint, respectively. Given $\Pi^K = (\Pi_1^K, \Pi_2^K, \cdots, \Pi_p^k)$, the following subproblem:

$$(SP)^K \begin{cases} v^K = \min \ C_v X_v + (C_s - \Pi^K B) X_s \\ N_v X_v + N_s X_s = 0 \\ B X_s \leq 1 \\ 0 \leq X_v \leq U_v \\ 0 \leq X_s \leq 1 \end{cases}$$

is solved as a capacitated minimum cost flow circulation problem The solution is (X_v^{K+1}, X_s^{K+1}). If $(v^K \geq \sigma_K)$,then the solution is optimal. Otherwise, a new column, $K+1$ is generated from solution to $(SP)^K$ and appended to the restricted master problem, to get $(DW)^{K+1}$, which is then solved. The column generated is:

$$[\ P_{K+1}, r_1^{K+1}, r_2^{K+2}, \cdots, r_p^{K+1}\ ]$$

whose components are obtained as described previously. The process iterates until $(v^Q \geq \sigma_Q)$ at iteration Q.

It is shown that the LP relaxation of the routing problem with service choices reduces to solving a series of capacitated minimum cost flow circulation problems, and restricted master LP problems having p rows where p = number of GUB constraints.

Appendix C

Applying Lagrangian relaxation to solve the routing problem with service choices

This appendix shows:

1. How the Lagrangian relaxation is applied to solve the routing problem with service choices by relaxing the GUB constraints.

2. That the optimal dual variables of the Lagrangian dual corresponding to the Lagrangian relaxation above, is obtained by solving the LP relaxation of the routing problem with service choices.

Solving the Routing problem with service choices using Lagrangian relaxation.

From section 3.1, the Lagrangian relaxation of the routing problem with service choices, obtianed by pricing out the GUB constraints results in two problems:

The Lagrangian subproblem $L(W)$:

$$\begin{aligned} z(L(W)) \;=\; \min\; & C_v X_v + W(B + C_s)X_s - 1W \\ & N_v X_v + N_s X_s = 0 \\ & 0 \le X_v \le U_v \\ & 0 \le X_s \le 1 \end{aligned}$$

where, $X_s \in \{0,1\}$ is replaced by $0 \le X_s \le 1$, without affecting the solution of $L(W)$.

The Lagrangian dual(Master) problem (DL) is:

$$\max_{W \ge 0}\; z(L(W))$$

Since the feasible region of $L(W)$ is bounded, let $K = \{(X_v^t, X_s^t) | t = 1, 2, \cdots, T\}$ be the set of extreme points which encloses $L(W)$. Then (DL) can be written as :

$$\max_{W \ge 0}\; \{C_v X_v^t + W(B + C_s)X_s^t - 1W \quad t = 1, 2, \cdots, T\}$$

Let $\Lambda = C_v X_v^t + W(B + C_s)X_s^t - 1W$, problem (DL) can be expressed as:

$$\begin{aligned} & \max \quad \Lambda \\ & \Lambda \le C_v X_v^t + W(B + C_s)X_s^t - 1W \quad t = 1, 2, \cdots, T \\ & W \ge 0 \end{aligned}$$

whose decision variables are Λ and W. Since T is large, solving (DL) is difficult. (DL) can be solved iteratively given a subset of extreme points $J \ll T$.

Given a subset of extreme points J, the resultant restricted master proble $(DL)^J$ is solved[1]:

$$(DL)^J \begin{cases} \max \quad \Lambda \\ \Lambda \le C_v X_v^t + W(B + C_s)X_s^t - 1W \quad t = 1, 2, \cdots, J \\ W \ge 0 \end{cases}$$

[1] $(DL)^J$ corresponds to instances of (SDL) in chapter 3

The solution yields (Λ^J, W^J). Given, W^J, $L(W^J)$ is solved to yield $z(L(W^J))$ whose solutions are (X_v^{J+1}, X_s^{J+1}). If $z(L(W^J)) = \Lambda^J$, the solution (X_v^{J+1}, X_s^{J+1}) is optimal to the Lagrangian relaxation. Otherwise, a constraint $\Lambda \leq C_v X_v^{J+1} + W(B + C_s)X_s^{J+1} - 1W$ is appended to $(DL)^J$. The process iterates by setting $J \leftarrow J + 1$. Note that the linear constraints in $(DL)^J$ represent a lower envelope corresponding to intersections of linear functions. The resultant function, $L(W)$, is a concave and continuous but nondifferential. As such, a psuedo gradient approach, called subgrdient optimization, can be used to find W and Λ. For large problems, this approach is generally taken, and is described in section 3.1.

The optimal dual variables are found by solving the LP relaxation, (P^0)

The LP relaxation of (P) is:

$$\begin{array}{rl} \min \;\; C_v X_v + C_s X_s & \text{Dual variables} \\ N_v X_v + N_s X_s = 0 & \Pi \\ B X_s \leq 1 & \alpha \\ 0 \leq X_v \leq U_v & \beta_1 \\ 0 \leq X_s \leq 1 & \beta_2 \end{array}$$

Solving (P^0) yields the following variables, Π, α, β_1, β_2 corresponding to the conservation of flow constraints, the GUB constraints and the bounding costraints $-X_v \geq -U_v$ and $-X_s \geq -1$ respectively. Conversely, the dual variables can be found by solving the dual problem of (P^0):

$$\begin{aligned} \max \;\; & -\alpha 1 - \beta_1 U_v - \beta_2 1 \\ & \Pi N_v - \beta_1 \leq C_v \\ & \Pi N_s - \alpha B - \beta_2 \leq C_s \\ & \alpha, \;\; \beta_1 \;\; \beta_2 \geq 0 \end{aligned}$$

It is now shown that $W = \alpha$ solves (DL). Recall that (DL) is:

$$\max_{W \geq 0} \; z(L(W)) \;=\; \max_{W \geq 0} \left\{ \begin{array}{rl} \min \;\; C_v X_v + W(B + C_s) X_s - 1W \\ N_v X_v + N_s X_s = 0 \\ 0 \leq X_v \leq U_v \\ 0 \leq X_s \leq 1 \end{array} \right\}$$

Since, { } is an LP, the dual[2] exits, and (DL) can be written as:

$$\max_{W \geq 0} \left\{ \begin{array}{l} \max \quad -W1 - \beta_1 U_v - \beta_2 1 \\ \qquad \Pi N_v - \beta_1 \leq C_v \\ \qquad \Pi N_s - \beta_2 \leq C_s + WB \\ \qquad \beta_1 \ \beta_2 \geq 0 \end{array} \right.$$

$$= \left\{ \begin{array}{l} \max \quad -W1 - \beta_1 U_v - \beta_2 1 \\ \qquad \Pi N_v - \beta_1 \leq C_v \\ \qquad \Pi N_s - WB - \beta_2 \leq C_s \\ \qquad W, \ \beta_1 \ \beta_2 \geq 0 \end{array} \right.$$

Solving (DL) is thus equivalent to solving the dual of (P^0). The optimal value of W is thus equals to α. Conversely, the optimal value of $W = \alpha$ can be obatined by solving (P^0), where α corresponds to the optimal dual variables of the GUB, or priced out, constraints.

This result always hold whenever the Lagrangian subproblem can be solved as an LP without affecting its solution. This is known as the integrality property of the dual.

[2]Let Π, β_1, β_2 be defined as before

Appendix D

Applying Benders method to solve the routing problem with service choices

This appendix shows how Benders decomposition is applied to solve the routing problem with service choices.

From section 3.1, the Benders subproblem is $R(X_p)$ given by:

$$\begin{array}{ll} C_s X_p + \min\ C_v X_v & \text{Dual variables} \\ N_v X_v = -N_s X_s & \Pi \\ 0 \le X_v \le U_v & \beta_1 \end{array}$$

where $X_P \in Q = \{X_s | BX_s \le 1 \text{ and } X_s \in \{0,1\}\}$.

Since $R(X_p)$ is an LP, its dual, $DR(X_p)$, is given by:

$$DR(X_p) \begin{cases} C_s X_p + \max\ -\Pi(N_s X_s) - \beta_1 U_v \\ \Pi N_v - \beta_1 \le C_v \\ \beta_1 \ge 0 \end{cases}$$

Note that if $DR(X_p)$ is infeasible, $R(X_p)$ is also infeasible since it is bounded. However, since the feasibility of $DR(X_p)$ does not depend on X_p, this means that for all X_p, $R(X_p)$ is infeasible, or (P) is infeasible. Conversely, if $DR(X_p)$ or $R(X_p)$ is feasible, then, a feasible solution can be found given by (X_p, X_q) where $X_v = X_q$ is the solution to $R(X_p)$ is

unbounded, then $R(X_p)$ is infeasible, from the duality theory. Let $H = \{(\Pi, \beta_1) | \Pi N_v - \beta_1 \leq C_v, \ \beta_1 \geq 0\}$, the feasible region of $DR(X_P)$, consists of a finite number of extreme points, y^l, $(l = 1, 2, \cdots, L)$ and extreme rays[1] v^m, $(m = 1, 2, \cdots, M)$. Now, if for some X_p, there exists m such that $v^m \hat{b} > 0$, where $\hat{b}$ is the vector $(-N_s X_p \quad -U_v)$, then the objective value of $DR(X_p)$, $h\hat{b}$ for $h \in H$ is unbounded. Conversely, if the maximum of $h \times \hat{b}$ for $h \in H$ is unbounded, then for some X_p, there is a ray v^m such that $v^m \hat{b} > 0$.

Thus, to ensure that $DR(X_p)$ is not unbounded (which implies that (P) is infeasible, as described before),

$$v^m \hat{b} \leq 0 \qquad\qquad m = 1, 2, \cdots, M$$

This provides the necessary and sufficient conditions on X_s to permit feasible solutions to the problem (P). Assuming that (P) is feasible, $DR(X_p)$ can be written as:

$$\max_{l=1,2,\cdots,L} \quad y^l \hat{b}$$

$$v^m \hat{b} \leq 0 \quad m = 1, 2, \cdots, M$$

$y^l \hat{b}, v^m \hat{b}$ are known as Benders cuts.

The Benders master program, (RM) can then be written as:

$$\max_{X_p \in Q} \left\{ \begin{array}{c} C_s X_p + \max\limits_{l=1,2,\cdots,L} \quad y^l \hat{b} \\ \\ v^m \hat{b} \leq 0 \quad m = 1, 2, \cdots, M \end{array} \right\}$$

Letting

$$r = C_s X_p + \max_{l=1,2,\cdots,L} \quad y^l \hat{b}$$

[1]Extreme rays of $DR(X_p)$ can be enumerated by finding all extreme rays of $\Pi N_v - \beta_1 \leq 0, \beta_1 \geq 0$. Extreme rays can also be obtained from the nonbasic columns when $DR(X_p)$ is solved, yielding an unbounded solution.

problem (RM) can be rewritten as:

$$\min \ r$$

$$r \geq C_s X_p + y^l \hat{b} \qquad l = 1, 2, \cdots, L$$

$$0 \geq v^m \hat{b} \qquad m = 1, 2, \cdots, M$$

$$X_p \in Q$$

Note that $X_p \in Q$ are integer constraints. This means that (RM) is an integer programming problem with one continuous variable r. The Benders decomposition is described as follows:

Step 0 Ititialization. Select $X_p \in Q$ and $z^{big}(z^{small})$, arbitrarily large(small). Go to 1.

Step 1 Solve LP. Solve the LP:

$$\max - \Pi(N_s X_p) - \beta_1 U_v$$

$$\Pi N_v - \beta_1 \leq C_v$$

$$\beta_1 \geq 0$$

This yields an optimal extreme point y^l or extreme ray v^m. If y^l is obtained, set $z^{big} \leftarrow \min\{z^{big}\ ,\ C_s X_p + y^l \hat{b}\}$. Go to 2.

Step 2 Solve IP. Solve the integer program.

$$\min \ r$$

$$r \geq C_s X_p - y_1^l (N_s X_s) - y_2^l U_v \qquad l = 1, 2, \cdots, A$$

$$0 \geq -v_1^m (N_s X_s) - v_2^m U_v \qquad m = 1, 2, \cdots, B$$

$$1 \geq B X_s$$

$$X_s \in \{0, 1\}$$

where $y^l = (y_1^l, y_2^l)$ and $v^m = (v_1^m, v_2^m)$ and A , B includes the number of extreme points and rays found from Step 1. $z^{small} \leftarrow$ objective value r, and $X_p \leftarrow$ optimal value of X_s. Go to 3.

Step 3 Termination test. If $z^{small} < z^{big}$, go to Step 2. Otherwise, $z^{small} = z^{big}$, and X_p is optimal. In this case, let the solution of $R(X_p)$ be X_q, then, (X_p, X_q) is the optimal flow values with an objective vlaue of $C_s X_p + C_v X_q$.