Routing Methods for Twin-Trailer Trucks

by
Jonathan Eckstein
A.B., Harvard College
(1980)

Submitted in Partial Fulfillment of the Requirements of the Degree of
Master of Science
In Operations Research

at the
Massachusetts Institute of Technology
May, 1986

Signature of Author: _____
Interdepartmental Program in Operations Research
May 9,1986

Certified by: _____
Yosef Sheffi
Thesis Supervisor

Accepted by: _____
Jeremy F. Shapiro
Chairman, Interdepartmental Committee on Operations Research

Routing Methods for Twin-Trailer Trucks

by
Jonathan Eckstein

Submitted on May 9, 1986 to the department of Civil Engineering
in Partial Fulfillment of the Requirements of the Degree of Master of Science
in Operations Research

Abstract

This thesis presents a body of initial work on a class of routing problems that resemble classical vehicle routing problems (VRP's), but differ in a few key aspects. A particular subset of this class arises in the regional-level operation of an LTL (Less than TruckLoad) truck line using twin-trailer combinations. Other problems in the class, covered in much less detail, may be useful in planning regional or overall operations of shipping lines, railroads, barge companies, or air freight carriers. Generally, the orientation of the problem model is towards freight carriers who consolidate shipments into large quanta or *containers*. This practice is currently widespread as it reduces handling costs.

The thesis describes a class of regional LTL twin-trailer truck problems, called the "central 2-CFP", contained in a broader class named k-CFP. It presents an aggregate commodity flow formulation of the k-CFP, and analyzes the formulation's LP relaxation via Lagrangean duality. The next topic is a Lagrangean branch and bound method for the central 2-CFP, including computational tests. The thesis closes by suggesting improvements and alternatives to the tested method.

Thesis Supervisor: Yosef Sheffi

Title: Associate Professor of Civil Engineering

Acknowledgments

This thesis summarizes work I performed over slightly more than one year, under the supervision of Professor Yosef Sheffi of MIT's Center for Transportation Studies. Thanks are naturally due to Yossi first and foremost, especially for giving me the time and space I needed to grow intellectually at a most critical juncture. Some preliminary results also constituted my term project in Professor Jeremy Shapiro's Network Optimization class; he deserves credit for a few helpful comments.

Thanks are also due to:

- The remaining faculty, staff and students of the OR Center, especially Marcia Ryder/Vanelli/Chapman.
- XEROX corporation for making available their excellent STAR "electronic desk" computer system.
- United Parcel Service and Consolidated Freightways, Inc. (Chicopee terminal) for making time for some informative site visits.
- My parents and my girlfriend Bonnie for their moral support (plus some last-minute proofreading).

Jonathan Eckstein
Cambridge, Massachusetts
May, 1986

Table of Contents

Notation

Scalars are in *italics*.

Vectors and matrices are in **boldface**. A subscripted boldface letter ($\mathbf{x}_q$) indicates a vector or matrix from an indexed collection. Elements of vectors and matrices are in subscripted italics (x_{ij}).

Boldfaced relationships ($=, >, <, \leq, \geq$) hold for each element of the vectors on either side.

$\doteq$ means "defined to be" ($\doteq$ for vectors).

$:=$ means "gets" or computational assignment ($:=$ for vectors).

$\lceil \rceil$ stands for integer round up. $\lceil \rceil$ stands for this operation applied to every element of a vector.

$\lfloor \rfloor$ means integer round down.

Transpose is written as a $^{\mathrm{T}}$. A "prime" mark (') has no fixed significance.

(i, j) denotes the arc from node i to node j in a directed network.

$<i, j>$ denotes the edge connecting nodes i and j a graph.

A long hyphen (—) is a minus sign. *Short* hyphens (-) are used to denote paths and cycles in a network. For instance 0-6-3-0 does not mean the integer -9, but rather the directed cycle composed of arcs (0, 6), (6, 3), and (3, 0).

1. Introduction

Over the past few years, regulatory changes have allowed twin-trailer trucks to be used much more widely on United States highways. In such "combinations", a single tractor may haul two 28-foot trailers ("doubles" or "pups"), instead of one 45- or 48-foot trailer (a "van" or "semi"). A tractor may also travel alone ("bobtail") or pull just one short trailer. Doubles have proved extremely popular with less-than-truckload (LTL) motor carriers. We begin by reviewing how an LTL carrier operates, and describing how doubles can fit into this scheme.

1.1 Background

Typically, an LTL truck line has a three-tiered shipping network, which operates as follows:

(i) Shipments are picked up from individual customers and taken a short distance to a local "end-of-line" or "city" terminal.

(ii) The shipments are then consolidated and transferred to a regional or "breakbulk" terminal up to a few hundred miles away. The word *breakbulk* is used because the contents of trailers are "broken down" (sorted) at such facilities.

(iii) Each shipment then travels over a network of "main line" routes (generally interconnecting breakbulk terminals) until it reaches the breakbulk terminal nearest its destination.

(iv) In the reverse of step (ii), each shipment travels to an end-of-line (EOL) terminal near its final destination.

(v) Reversing step (i), shipments are delivered to their final destinations.

Generally, twin-trailer combinations are inappropriate for local pickup and delivery; therefore doubles have had little impact on the local layer of the operation. On the other hand, doubles have a very positive and well-known effect on "main line" activities (see [SP]): they decrease the amount of manual freight

handling needed to maintain a given level of service, and they provide extra carrying capacity per driver on heavily-used routes.

Doubles can also be used, however, to reduce costs in the intermediate or "group" tier of the operation, between the end-of-line and breakbulk terminals. The means of achieving such savings are poorly understood due to their complex combinatorial nature. This problem provided the inspiration for the research presented here.

The remainder of this chapter, describes a simple version of the regional twin-trailer routing problem, and then defines a large collection of related combinatorial problems which we shall call the k-CFP. We then examine the complexity of this class, both from an NP-hardness and an intuitive point of view. The chapter closes with a short review of the relevant literature, and we then proceed to the issue of problem solution.

1.2 Problem Statement

We now concentrate on the operation of a single breakbulk terminal by an LTL carrier using 28-foot trailers exclusively. We will not consider the main line side of the operation, but only the task of transferring freight to and from the nearby end-of-line terminals. Typically, there will be twenty to fifty such satellite terminals, but in some cases there may be more. Let these terminals be numbered $1,...,n$.

Suppose that on a given day the truck line must deliver some whole number d_i of trailers from the breakbulk terminal (henceforth also known as "the break") to end-of-line terminal i ($i = 1,...,n$). Conversely, it must pick an integer number p_i of trailers from each EOL terminal i and transport them to the break. The total trailer pool at each terminal, including the break, must remain the same at the end of the day: the total number of trailers at each facility cannot change. This requires that empty trailers be removed from any terminals where deliveries exceed pickups ($d_i > p_i$), and must be supplied to any terminals where pickups exceed deliveries ($d_i < p_i$). Such daily balancing is typical of operations in which freight movements do not show a pronounced, predictable weekly pattern. In situations where there *is* such a pattern, carriers may find it advantageous to balance weekly, allowing limited trailer pool fluctuations from day to day. We have not attempted to model this more complicated situation.

Trailer movements are accomplished by *tractors* that can haul up to two trailers at a time. These tractors drive from terminal to terminal, and at each terminal they

may stop and drop off one or two trailers, and may also pick up one or two. We assume that the distances between terminals obey the triangle inequality (for instance, we could use the shortest-path distances or times over a road network). The tractor pool must also be balanced daily; one cannot let tractors accumulate at some terminal nor constantly remove them from another.

The objective is to "cover" all the desired trailer movements with the minimum number of overall tractor miles. We also, assume, for the purposes of most of this thesis, that the distances are small enough and the supply of tractors large enough that we can execute the minimum-mileage solution in one day.

In a real situation, there might be a large number of additional considerations, including restrictions on the kinds of movements tractors may make, or on the time they can take to do so. There might also be time windows on certain pickups and deliveries. Other complications may arise from driver schedules , availability of the "dollies" that are used to connect the two trailers in a combination, or from other factors. We will defer looking at these extra constraints for the moment, and concentrate or the "core" of the problem.

1.3 The CFP and its Classifications

We will call the minimization problem defined above, unencumbered by additional constraints, the "central 2-CFP", where "2-CFP" stands for the two Containerized Freight Problem. *Non-central* 2-CFP problems differ from *central* ones in that there is no distinguished breakbulk or central node, and the required freight movements are given by an origin-destination table. In such problems, any terminal can generate any integral number of trailer-loads of freight for any other; the only restriction is that shipments for different destinations cannot be mixed on a single trailer.

In general, the "k-CFP" problem, central or non-central, will denote the same situation as above, where each tractor (or locomotive, vessel, aircraft) may transport up to k trailers (or boxcars, containers, barges, pallets) at a time.

The reader should be warned that the non-central 2-CFP may not be an appropriate model for either the overall or main line operation of an LTL carrier. The reason is that there is no provision for additional sorts ("break bulk operations") while shipments are enroute. However, in other applications, namely those in which shippers purchase capacity in container units (*e.g.* boxcars) the non-central k-CFP

may be useful. Naturally, there is nothing sacred about tractors and trailers -- the model is equally applicable to any kinds of vehicles and containers. However, we will continue to use the trucking terminology in order to keep the discussion concrete. The central 2-CFP is the primary focus of this thesis.

The CFP group of problems is similar "classical" vehicle routing (VRP) problems, but differs in some key respects. The differences may be summarized as follows:

Pickups and Deliveries: Unlike the basic VRP, loads must be both picked up and delivered, and empty capacity must be accordingly repositioned.

Split Loads: There is no restriction that a *single* vehicle deliver or pick up the entire demand at a given node; in fact, demand may exceed the capacity of one vehicle.

Vehicle Capacity: At any given moment a vehicle (tractor) can transport containers (trailers) for at most k different destinations. For small k (such as $k=2$) this greatly limits the kinds of tours in the optimal solution.

Essentially, the perspective taken in CFP problems is that of the freight carrier: to control handling costs, freight is generally manipulated in relatively large containers (trailers or boxcars), and there is no reason (other than simplicity) that all containers with the same origin and destination need take exactly the same route. To the CFP, a container is "full" if it contains any freight; quanta smaller than one container are not modeled. Finally, all points are potential origins *and* destinations, and repositioning of empty capacity is an essential, integral part of the model.

By contrast, basic vehicle routing problem models tend to take a perspective limited to either step (i) or (v) above, or of a "vertically-integrated" shipper or distributor which operates its own vehicles. The modeling is generally on a smaller scale, capturing the movements of individual shipments which are usually indivisible and much smaller than vehicle capacities.

1.4 A First Look at Complexity

How hard a problem is the k-CFP? Even if we consider only central problems, the k-CFP takes on more and more of the character of a traveling salesman problem as k becomes large. This property can be exploited as follows:

1.4.1. Proposition. If k is considered to be part of the input, the central k-CFP problem is NP-hard.

Proof: Consider any instance of the traveling salesman problem with distances obeying the triangle inequality. We demonstrate a polynomial transformation to the k-CFP. If the TSP instance has m nodes, set the number of EOLs, n, equal to $m-1$. Then set $k=n$, and $p_i=d_i=1$ for all i. Use the same distances in the CFP as in the TSP instance. Because a single "tractor" now has the capability to service all the terminals, it may then be easily seen that the optimal CFP solution is essentially a minimum cost traveling salesman tour of all the nodes. In cases where the triangle inequality is met with equality for some triplets of nodes, it is possible that the optimal CFP solution may break down into more than one tour emanating from the break. But because the triangle inequality is satisfied, any such "kinks" may be straightened out without increasing total cost. **QED**.

1.4.2. Corollary. The non-central k-CFP problem is NP-hard. This is because the central problem is just a special case of the non-central one.

For *fixed* k, which is a case of greater practical interest, the complexity of the k-CFP is an open question. We suspect that some special cases of the central 2-CFP may be reduced to nonbipartite matching problems. We will also show shortly that the general 1-CFP can be reduced to a "classical" (Hitchcock) transportation problem. Thus, it seems at least possible that the general central 2-CFP may be computationally tractable.

1.5 Examples of the Central 2-CFP

To give some feel for the central 2-CFP, we present two particular instances.

Our first example has only two end-of-line terminals, and is meant to give a feeling for how twin-trailer trucks can expand the options available to a breakbulk planner. Suppose we have two EOLs, each of which requires the delivery and pickup of one trailer-load of freight ($p_1=d_1=p_2=d_2=1$). If tractors could haul only *one* trailer at a time, the problem would essentially have only one reasonable solution. One would dispatch one tractor-trailer combination from the break to each EOL. There, each would deliver a trailer of "outbound" freight for the EOL, and pick up one trailer of inbound cargo bound for the break. This may be graphically depicted as follows:

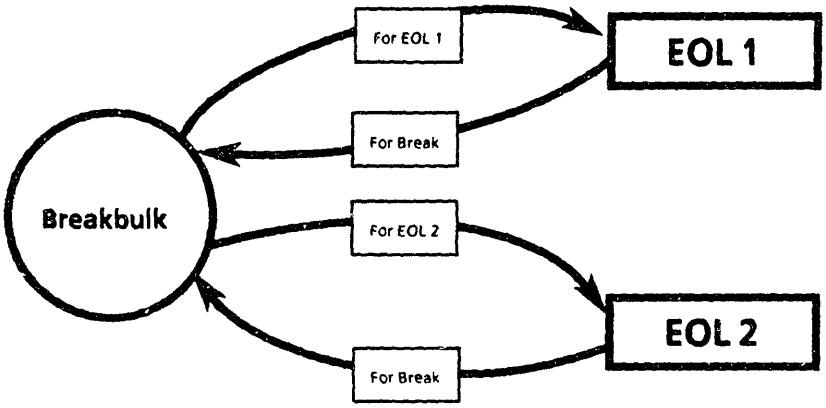

Figure 1a: First example; one trailer per truck

If a truck can haul *two* trailers, though, there is a lower-mileage solution. One simply dispatches a single tractor, hauling a trailer of freight for one EOL *and* one for the other. At the first EOL, it exchanges one of these trailers for a trailer loaded with inbound shipments for the break, and it then proceeds to the second EOL. There, it exchanges the second outbound trailer for an inbound one, and continues to the break. Visually, we have

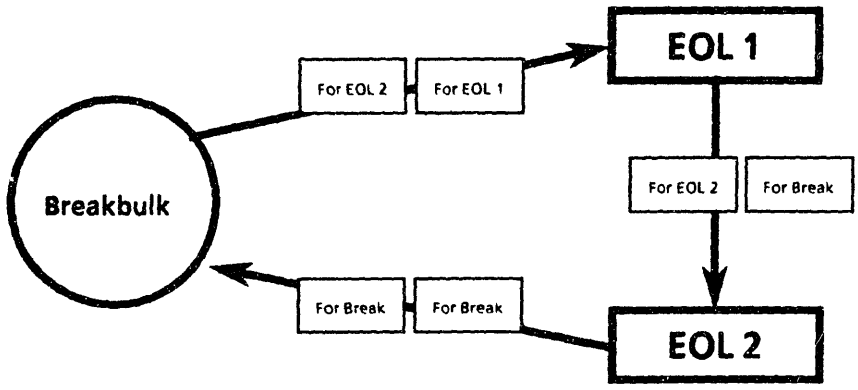

Figure 1b: First example; two trailers per truck

By the triangle inequality, there are fewer tractor miles in the second solution than in the first.

Naturally, this first example was chosen to be easy to understand. In our research, we have found that problems with as few as four EOLs are sometimes too difficult to solve by simple inspection. We now present one such example, created by means of a random number generator:

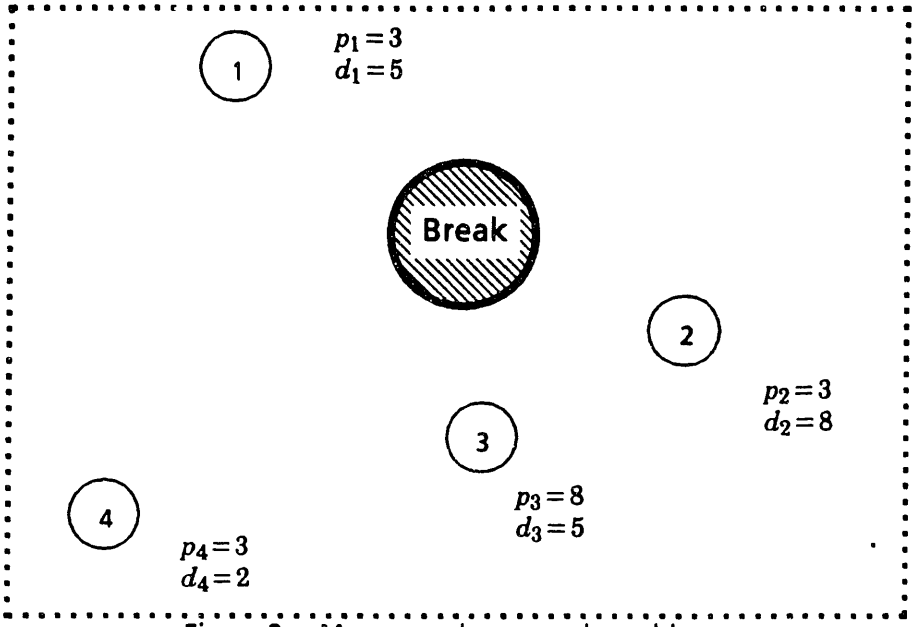

Figure 2a: More complex example problem

One of the possible minimum tractor-mileage solutions is shown below. Each trailer is represented by a box labeled with the destination of its contents, where "B" stands for the break. Two adjacent boxes represent a twin-trailer combination, and empty trailers are unlabeled.

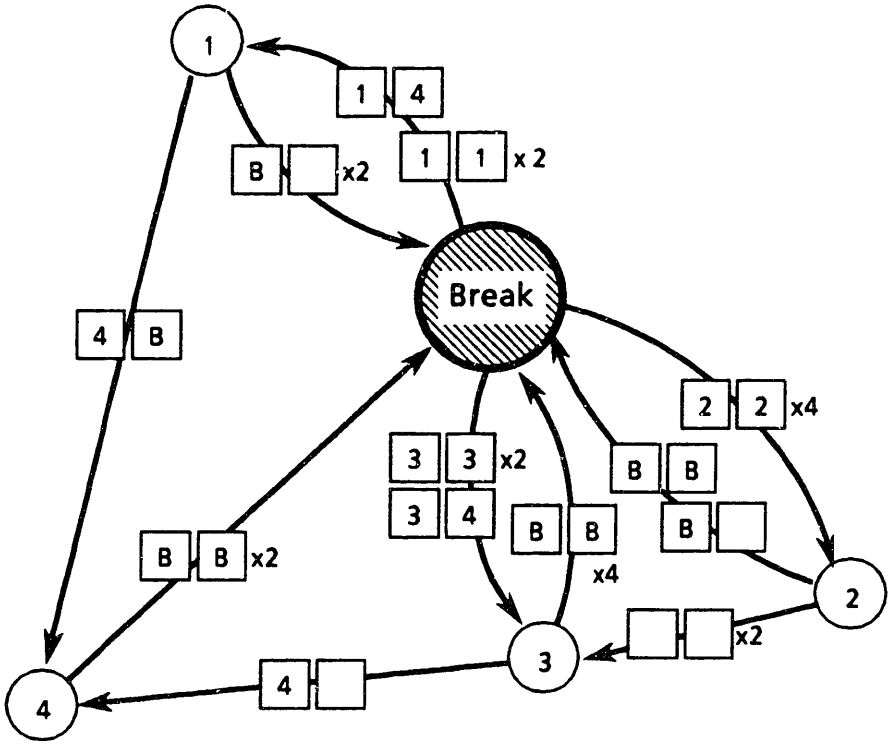

Figure 2b: Complex example solution

The combinatorics of the problem can clearly get quite involved, even for an instance with so few nodes.

1.6 Literature Review

This is certainly not the first time such problems have been studied. In 1978, Assad [AS] proposed modeling the operations of a freight railroad almost exactly as we have described the non-central k-CFP, where k was approximately 50. The principle difference was a more elaborate cost structure.

However, this is, as far as we know, the first investigation to define a CFP-like class or to concentrate on central problems with small k. Because we are taking what appears to be a somewhat fresh point of view, there is not a great deal of directly relevant literature. The most pertinent literature is that dealing with the larger issues of posing and solving vehicle routing problems. The voluminous survey of Bodin *et. al.* [BOD] provides a comprehensive introduction to the practice of vehicle routing. Because the k-CFP admits an arbitrary mixture of pickups and deliveries, it could be considered as a variant of the static multi-vehicle dial-a-ride problem. However, we have not yet discovered a way to exploit this connection.

While the Bodin survey is concerned in large part with heuristics and practical approaches to vehicle routing, Magnanti's survey [MAG] focuses on the theoretical issues of trying to construct mathematical programming algorithms for basic VRPs. He describes four types of formulations upon which methods can be based: aggregate commodity flow, disaggregate commodity flow, vehicle flow, and set covering. These categories proved very useful in thinking about how to formulate the k-CFP.

The principle solution technique discussed in this thesis is the now common combination of Lagrangean relaxation, subgradient step, and branch and bound. The two tutorial papers by Fisher [FISH1] and [FISH2] constitute a lucid introduction to this integer programming technique and its relatives. The approach has been used successfully on some vehicle routing problems, most notably as described in [BELL] and [FISH3]. These successes provided some hope that the Lagrangean method might work for the rather different class of problems studied here. Our effort differs in many ways, not the least of which is the use of nonnegative integer variables, as opposed to zero-one variables.

1.7 Organization

The rest of this thesis addresses the issue of solving the k-CFP and, in particular, the central 2-CFP. We will begin by presenting an aggregate commodity flow formulation for the k-CFP, in both central and non-central cases, and then analyze it by means of Lagrangean duality. We then discuss the construction of a Lagrangean branch and bound method for the central 2-CFP, adjoining some easily-enforced constraints that significantly tighten the relaxation. We then present and discuss the rather mixed computational results obtained with this method, and speculate on possible improvements. We close by suggesting some promising heuristic approaches to the central 2-CFP, and by making some general observations about Lagrangean branch and bound methods.

2. Aggregate Flow Formulation

So far, we have intentionally not introduced an explicit mathematical programming formulation of the k-CFP. As with the VRP class of problems, many different formulations are possible, and we did not wish to unduly prejudice the reader. It is well known that the choice of formulation, in particular the "tightness" of its LP relaxation, can greatly influence the performance of certain common integer programming solution approaches, namely Benders' and Lagrangean decomposition methods (see [MAG]).

We will now introduce one possible formulation of the central k-CFP, one which is highly aggregated and resembles the aggregate commodity flow formulation of the VRP discussed in [GG].

2.1 Formulation for Central Problems

Let us consider central problems first. Define a system of $n+1$ terminals, the breakbulk being terminal number 0, and the n end of line terminals being 1,...,n, with pickup and delivery demands p_i and d_i. Define

$$d_0 \doteq -\sum_{i=1}^{m} d_i \qquad p_0 \doteq -\sum_{i=1}^{m} p_i \quad , \tag{1}$$

and

$$\begin{aligned} \mathbf{p} &\doteq [p_0, \ldots, p_n]^{\mathrm{T}} \\ \mathbf{d} &\doteq [d_0, \ldots, d_n]^{\mathrm{T}} \quad . \end{aligned} \tag{2}$$

Further, let

- $\mathbf{A}$ be the node-arc incidence matrix of the complete network connecting the $n+1$ terminals.
- $\mathbf{c}$ be the vector of corresponding distances or costs c_{ij} incurred by driving a tractor between nodes i and j.

We now define four different flow vectors $\mathbf{t}$, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ over the network given by $\mathbf{A}$. These vectors are all conformal to $\mathbf{c}$:

- $\mathbf{t}$ is the vector of scalars t_{ij}, each giving the number of *tractors* traveling on the arc from terminal i to terminal j.

x is the vector of scalars x_{ij}, each giving the number of "outbound" trailers on each arc (i,j). An *outbound* trailer is one loaded with freight to be delivered at some EOL. Naturally, they all originate at the break.

y is, similarly, the vector of "inbound" flows y_{ij}. An *inbound* trailer is one loaded with freight destined for the breakbulk terminal, and originating at some EOL.

e is the vector of flows e_{ij} of empty trailers.

To have a stationary pool of tractors at each terminal we must have

$$\mathbf{At} = \mathbf{0} \quad . \tag{3}$$

To make the required deliveries at the EOLs, we need

$$\mathbf{Ax} = -\mathbf{d} \quad . \tag{4}$$

Considering **x** to be a network flow, this equation means that we have a demand of d_i at each end of line terminal i, and a supply of $-d_0 = \Sigma d_i$, that is, all the outbound trailers, at the break. Similarly, we need

$$\mathbf{Ay} = \mathbf{p} \tag{5}$$

to ensure that all the inbound loads are picked up and delivered to the breakbulk terminal. Finally, note that for all i, including $i=0$, the number of empty trailers that must be removed from terminal i is $p_i - d_i$. If this quantity is negative, it indicates that $|p_i - d_i|$ empty trailers must be supplied, rather than removed. Thus, stationary trailer inventories require

$$\mathbf{Ae} = \mathbf{p} - \mathbf{d} \quad . \tag{6}$$

There is one thing missing from the formulation: a connection between the "motive power" network of tractor flows and the networks of trailer movements. These "binding" or "linking" constraints simply express that a tractor can haul at most k trailers:

$$kt_{ij} \geq x_{ij} + y_{ij} + e_{ij} \quad \forall (i,j) \quad , \tag{7}$$

which may also be written

$$k\mathbf{t} - \mathbf{x} - \mathbf{y} - \mathbf{e} \geq \mathbf{0} \quad . \tag{8}$$

The complete aggregate flow formulation of the central k-CFP may thus be expressed as follows

$$
\begin{array}{lrrrrrrr}
\text{Minimize} & \mathbf{c}^T\mathbf{t} & & & & & & \\
\text{Such That} & \mathbf{At} & & & & = & \mathbf{0} & \\
& & \mathbf{Ax} & & & = & -\mathbf{d} & \\
& & & \mathbf{Ay} & & = & \mathbf{p} & \\
& & & & \mathbf{Ae} & = & \mathbf{d}-\mathbf{p} & \\
& k\mathbf{t} & -\mathbf{x} & -\mathbf{y} & -\mathbf{e} & \geq & \mathbf{0} & \\
& \mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & \geq & \mathbf{0} & \\
& \mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & & \text{integer} & .
\end{array}
\tag{9}
$$

Note that even though there are $n+1$ possible freight destinations in the problem, its central nature allows loaded trailers to be represented by only two commodities, **x** and **y**. This device will not work in the non-central case. There, one must label all trailers either by their origins or destinations, and there will be at least as many commodities as there are terminals. We will return to the non-central case shortly. For now, we comment on certain features of the formulation as it stands.

Suppose that the optimum solution of the integer program could be found by some direct procedure, such as an "off-the-shelf" IP code. The resulting optimum value of **t** would be a flow vector and not an explicit set of driver instructions. To extract such instructions, one must subsequently decompose **t** into tours for the tractors to follow, typically all beginning and ending at the same terminal. One must then assign trailers to each leg of all such tours. Such an extraction is clearly possible: as **t** is a balanced flow with no exogenous sources or sinks, it can be decomposed into tours by a process very similar to finding an Euler tour of an even-degree graph. In a real situation, there may well be meaningful restrictions, in terms of time, distance, or number of stops, on the kinds of driver/tractor tours allowable. We call such additional requirements *side constraints*.

One is faced, then, with two "post-processing" problems: first, to determine if the given optimal value of **t** can be decomposed into *feasible* tours (in terms of the side constraints), and second, to perform such a decomposition. The aggregate nature of the formulation may make it difficult to express side constraints mathematically and incorporate them into the model -- if such considerations are paramount, then a different type of formulation may be required.

2.2 The Lagrangean Dual

The integer program constructed above has a familiar and important structure: it consists of a number of separate network problems linked together by some supplementary non-network constraints. A very common technique for dealing with problems with such recognizable embedded structure is forming the Lagrangean dual, whereby multipliers are attached to some constraints, and they are moved to the objective function. In this case, we choose to dualize the linking constraints, yielding, for each nonnegative $\mathbf{u} \in \mathbf{R}^{(n+1)n}$ (there are $(n+1)n$ linking constraints), the *Lagrangean relaxation*

$$
\begin{array}{lrrrrrr}
L(\mathbf{u}) \doteq \text{Min } \mathbf{c}^{\mathrm{T}}\mathbf{t} - \mathbf{u}^{\mathrm{T}}(k\mathbf{t} - \mathbf{x} - \mathbf{y} - \mathbf{e}) & & & & & & \\
\text{S.T. } \mathbf{At} & & & & = & \mathbf{0} & \\
\mathbf{Ax} & & & & = & -\mathbf{d} & \\
\mathbf{Ay} & & & & = & \mathbf{p} & \\
\mathbf{Ae} & & & & = & \mathbf{d}-\mathbf{p} & \\
\mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & \geq & \mathbf{0} & \\
\mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & & \text{integer}. &
\end{array}
\tag{10}
$$

If z is the optimum objective value for the original problem, a simple analysis (see, for example, [SHAP], p. 145) gives that $L(\mathbf{u})$ is a lower bound on z for all $\mathbf{u} \geq \mathbf{0}$. Furthermore, by rearranging the cost row, the optimization needed to compute $L(\mathbf{u})$ can be seen to consist of four independent network problems. Since network problems automatically have integer optima if their right-hand sides are integer, we can drop the explicit integrality constraints and write

$$
\begin{array}{lrrrrrr}
L(\mathbf{u}) = \text{Min } (\mathbf{c}-k\mathbf{u})^{\mathrm{T}}\mathbf{t} + \mathbf{u}^{\mathrm{T}}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{y} + \mathbf{u}^{\mathrm{T}}\mathbf{e} & & & & & & \\
\text{S.T.} \quad \mathbf{At} & & & & = & \mathbf{0} & \\
\mathbf{Ax} & & & & = & -\mathbf{d} & \\
\mathbf{Ay} & & & & = & \mathbf{p} & \\
\mathbf{Ae} & & & & = & \mathbf{d}-\mathbf{p} & \\
\mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & \geq & \mathbf{0}. &
\end{array}
\tag{11}
$$

Now, $L(\mathbf{u})$ is an easy quantity to compute, as it consists of four decoupled networks. It is also a lower bound on the original problem objective. Naturally, one would like to compute the *best* lower bound

$$D \doteq \text{Max}_{\mathbf{u} \geq \mathbf{0}} \, L(\mathbf{u}) \tag{12}$$

This maximization problem is the *Lagrangean Dual*. At this point, consider the LP relaxation of the original formulation, (9). The strong duality theory of linear programming can now be employed to show that $D = z_{LP}$, where z_{LP} is the optimum of the LP relaxation. In summary, the Lagrangean relaxation cannot be any tighter than the LP relaxation, and is exactly as tight for the correct choice of $\mathbf{u}$.

2.3 Solving the LP Relaxation

In the case of the k-CFP, it is actually possible to pick *a priori* the values of the multipliers $\mathbf{u}$ that yield the largest possible value of $L(\mathbf{u})$. The appropriate choice is

$$\mathbf{u}^* \doteq \left(\frac{1}{k}\right)\mathbf{c} \, . \tag{13}$$

To see how this choice yields the maximum $L(\mathbf{u})$, consider the problem of computing the $\mathbf{t}^*, \mathbf{x}^*, \mathbf{y}^*, \mathbf{e}^*$ that are minimal in $L(\mathbf{u}^*)$. To compute $\mathbf{x}^*$,we must solve

$$\begin{aligned} &\text{Min } (1/k)\mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\text{S.T.} \quad \mathbf{A}\mathbf{x} \;\; = \; -\mathbf{d} \\ &\qquad\quad \mathbf{x} \;\; \geq \; \mathbf{0} \, . \end{aligned} \tag{14}$$

The costs $\{(1/k)c_{ij}\}$ in this problem obey the triangle inequality because the $\{c_{ij}\}$ do. Recalling that the "outbound" trailers represented by $\mathbf{x}$ all originate at the break and go to the EOLs, we conclude that every such trailer may take a direct path from the break to its final destination. Similarly, we can conclude that all the "inbound" y_{ij} trailers can proceed directly from their originating EOLs to the break. This gives us $\mathbf{x}^*$ and $\mathbf{y}^*$. The empty trailers require a little more work, since supply and demand for them is spread out all over the network. However, they can still take direct paths from their origins to their destinations. This means that $\mathbf{e}^*$ may be computed via a simple transportation simplex procedure in which the sources are those terminals with an excess of empties, and the sinks are those with a deficit.

Finally, what is the form of $\mathbf{t}^*$? Note that the costs on the $\mathbf{t}$ network are

$$\mathbf{c} - k\mathbf{u}^* = \mathbf{c} - k(1/k)\mathbf{c} = \mathbf{0} \, , \tag{15}$$

so *any* feasible $\mathbf{t}$ will be optimal. However, select the particular value

$$\mathbf{t}^* \doteq \frac{1}{k}(\mathbf{x}^* + \mathbf{y}^* + \mathbf{e}^*) \geq \mathbf{0} \quad . \tag{16}$$

We first remark that

$$\mathbf{A}\mathbf{t}^* = \frac{1}{k}(\mathbf{A}\mathbf{x}^* + \mathbf{A}\mathbf{y}^* + \mathbf{A}\mathbf{e}^*) = \frac{1}{k}(-\mathbf{d} + \mathbf{p} + (\mathbf{d} - \mathbf{p})) = \mathbf{0} \ . \tag{17}$$

So, $\mathbf{t}^*$ is feasible and hence optimal. What is more, $(\mathbf{t}^*, \mathbf{x}^*, \mathbf{y}^*, \mathbf{e}^*)$ and $\mathbf{u}^*$ jointly satisfy the *global optimality conditions* (see [SHAP], pp. 142-144) which guarantee that $(\mathbf{t}^*, \mathbf{x}^*, \mathbf{y}^*, \mathbf{e}^*)$ are optimal for the LP relaxation, and $\mathbf{u}^*$ optimal in the Lagrangean dual. In our formulation, the conditions reduce to

(i) $(\mathbf{t}^*, \mathbf{x}^*, \mathbf{y}^*, \mathbf{e}^*)$ minimal in $L(\mathbf{u}^*)$

(ii) $\mathbf{u}^{*\mathrm{T}}(k\mathbf{t}^* - \mathbf{x}^* - \mathbf{y}^* - \mathbf{e}^*) = 0$

(iii) $(\mathbf{t}^*, \mathbf{x}^*, \mathbf{y}^*, \mathbf{e}^*)$ globally feasible, in particular $k\mathbf{t}^* - \mathbf{x}^* - \mathbf{y}^* - \mathbf{e}^* \geq \mathbf{0}$.

Condition (i) is fulfilled by construction, and (ii) and (iii) are true because we have chosen $\mathbf{t}^*$ such that

$$k\mathbf{t}^* - \mathbf{x}^* - \mathbf{y}^* - \mathbf{e}^* = \mathbf{0} \ . \tag{18}$$

Therefore, $(\mathbf{t}^*, \mathbf{x}^*, \mathbf{y}^*, \mathbf{e}^*)$ and $\mathbf{u}^*$ are optimal in the LP relaxation. If $\mathbf{t}^*$ happened to be integer, then these same conditions would guarantee optimality for the integer program as well; if $\mathbf{t}^*$ is not integer, then $\mathbf{t}^*$ is not feasible for the IP, and the global optimality condition break down. However, if $k = 1$, then $\mathbf{t}^*$ *must* be integer. This means that the central 1-CFP can be essentially solved by the transportation simplex procedure needed to compute $\mathbf{e}^*$. We conclude that the unrestricted central 1-CFP is a very easy problem.

If $k > 1$, then $\mathbf{t}^*$ will, in general, not be integer. Simple rounding up will not work because one may lose the property $\mathbf{A}\mathbf{t} = \mathbf{0}$ and/or the complementary slackness conditions (ii). However, the LP relaxation is very easy to solve. Furthermore, the only variables that need be non-integer in the LP optimum are the t_{ij}. At this point, one might consider using a branch and bound method to find the integer optimum. However, such a method could not take advantage of a construction like (16). The reason is that there is no explicit way of forcing $\mathbf{t}^*$ to obey branching constraints of the form $t_{ij} \leq m$ or $t_{ij} \geq m + 1$ (where m is integer).

Note that if the network represented by **A** is not complete, our results still essentially apply: simply substitute "shortest path" for each occurrence of "direct arc" in the above development. In this case, computing **e*** involves a transhipment problem rather that a transportation problem.

2.4 The Non-Central Case

The preceding development was done in the context of the central k-CFP. However, it still carries through in the non-central case. There, we have a system of n terminals (or simply "nodes") numbered $1,...,n$, with no distinguished central node. We define a network $\boldsymbol{N}$ of directed arcs (i,j) connecting these terminals. Let

- **A** be the node-arc incidence matrix of $\boldsymbol{N}$.
- **c** be a conformal vector of distances or costs c_{ij} obeying the triangle inequality.
- **B** be an $n \times n$ matrix of elements b_{ij}, giving the number of trailer-loads of freight originating at node i for delivery at node j.

We define, for $j=1,...,n$,

$$b_{jj} \doteq -\sum_{i \neq j} b_{ij} \quad , \tag{19}$$

that is, b_{jj} is minus the total number of trailers to be delivered at node j.

The decision variables are similar to those in the central case, except that loaded trailers are distinguished by their destinations, rather than merely categorized as "inbound" or "outbound". We define, all conformally to **c**,

- **t** to be the vector of tractor flows t_{ij} on each arc (i,j).
- $\mathbf{x}_q$ to be the vector of flows x_{qij} of trailers loaded with freight for final destination q on arc (i,j). Here, $q=1,...,n$.
- **e** to be, again, the flow of empties.

The formulation for the non-central problem is then

$$
\begin{array}{lllll}
\text{Min} & \mathbf{c}^{\mathrm{T}}\mathbf{t} & & & \\
\text{S.T.} & \mathbf{At} & = & \mathbf{0} & \\
 & \mathbf{Ax}_q & = & \mathbf{b}_q & q=1,\ldots,n \\
 & \mathbf{Ae} & = & -\mathbf{B1} & \\
 & k\mathbf{t} - \sum_{q=1}^{n} \mathbf{x}_q - & \mathbf{e} \geq & \mathbf{0} & \\
 & \mathbf{t}, \quad \mathbf{x}_q, & \mathbf{e} \geq & \mathbf{0} & q=1,\ldots,n \\
 & \mathbf{t}, \quad \mathbf{x}_q, & \mathbf{e} & \text{integer} & q=1,\ldots,n\ ,
\end{array}
\tag{20}
$$

where $\mathbf{b}_q$ is column q of $\mathbf{B}$, and $\mathbf{1}$ is the vector of all ones in $\mathbf{R}^n$ (hence $\mathbf{B1}$ is the sum of the columns of $\mathbf{B}$).

From here, the analysis can proceed virtually unchanged. By setting $\mathbf{u}^* = \mathbf{c}/k$, one can construct an optimal LP solution in which loaded trailers proceed to their destinations via shortest-path routes, and empties are redistributed by solving a simple transportation or transhipment problem. Again, the difficulty is that if $k>1$, the constructed tractor flows may not by integral.

By appropriate renumbering of terminals, any central k-CFP problem instance can be recast as a special case of the non-central k-CFP. The non-central formulation would then be a kind of "disaggregation" of the central one, in which outbound trailers are labeled according to their destinations, even though this is not strictly necessary. The construction above demonstrates that these two formulations yield LP relaxations with identical objective values. Thus, passing from the aggregate formulation (9) to the more disaggregated form (20) does *not* strengthen the LP relaxation for this particular problem and formulation approach. For a variety of recently studied integer programming problems, decreased model aggregation has resulted in tighter LP relaxations and, consequently, better convergence of solution methods (see [MAG] or [MW]). However, we have just seen that the particular type of disaggregation entailed by moving from formulation (9) to (20) is not helpful for the central k-CFP.

2.5 Physical Interpretation and Comments

Setting the multipliers $\mathbf{u}$ to $\mathbf{c}/k$ has a useful physical interpretation. In the computation of $L(\mathbf{u})$, it gives each arc (i,j) an imputed tractor costs of 0 and an

imputed trailer cost of c_{ij}/k. Essentially, one can think of this as allowing fractions of tractors and permanently attaching one k^{th} of a tractor to each trailer. The constraint $k\mathbf{t}-\mathbf{x}-\mathbf{y}-\mathbf{e}\geq\mathbf{0}$ (or similarly for the non-central case) is then automatically satisfied with zero slack, and trailers circulate in a shortest-path manner, subject to a lowest-cost repositioning of empties. This highlights a rather bothersome feature of our formulation: the linking constraints $k\mathbf{t}-\mathbf{x}-\mathbf{y}-\mathbf{e}\geq\mathbf{0}$ do not really have any effect without the additional restriction of integrality.

3. Constructing a Lagrangean Branch and Bound Method

We now describe a further exploitation of the Lagrangean relaxation of the aggregate flow formulation, this time to construct a branch and bound method for the central 2-CFP. Most of the development in this chapter is applicable to both the central and non-central k-CFP, but the incumbent generation strategies covered in the next chapter are not.

3.1 Background

Lagrangean branch and bound methods resemble conventional branch and bound methods for integer programs, except that they use lower bounds arising from a Lagrangean relaxation rather than from a pure LP relaxation. In cases such as ours, where the Lagrangean bound cannot be tighter than the LP relaxation, the main advantage to this approach is that the Lagrangean bound may be much easier to compute. The price paid for this advantage is that some time has to be spent adjusting the multipliers $\mathbf{u}$ in order to obtain adequate bounds. Lagrangean branch and bound (LBB) methods usually also include specialized heuristics to generate good incumbent solutions, but it should be noted that there is no reason why such routines could not be incorporated into LP-based methods. Figure 3 depicts the general form of LBB methods. For more information, readers should consult the well-known papers [FiSH1] or [FISH2].

All the steps in the flowchart in figure 3 are to varying degrees problem-dependent and arbitrary. Depending on exactly how each step is done (and on the particular problem instance input data) a method so constructed may converge quickly or slowly, or may never converge at all.

3.2 Strengthening the Relaxation: Node Activities

As we have seen, the lower bound given by $L(\mathbf{u})$ is at best as strong as that provided by the LP relaxation of our formulation. We now explore a method for strengthening the relaxation.

Recall the original formulation,

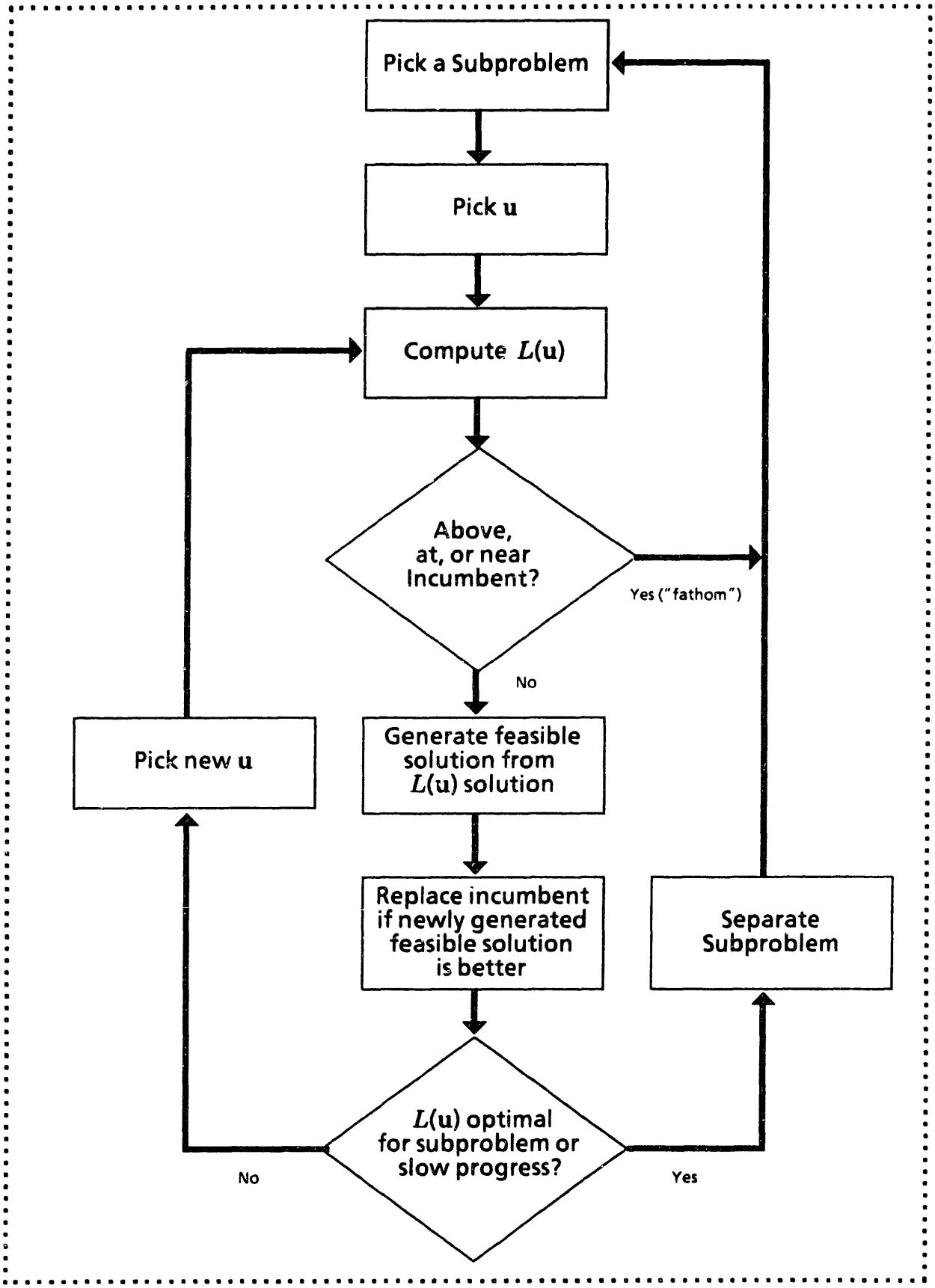

Figure 3: General Lagrangean Branch-and-Bound

$$
\begin{array}{lrrrrrl}
\text{Minimize } & \mathbf{c}^T\mathbf{t} & & & & & \\
\text{Such That } & \mathbf{At} & & & & = & \mathbf{0} \\
& & \mathbf{Ax} & & & = & -\mathbf{d} \\
& & & \mathbf{Ay} & & = & \mathbf{p} \\
& & & & \mathbf{Ae} & = & \mathbf{d}-\mathbf{p} \\
& k\mathbf{t} & -\mathbf{x} & -\mathbf{y} & -\mathbf{e} & \geq & \mathbf{0} \\
& \mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & \geq & \mathbf{0} \\
& \mathbf{t}, & \mathbf{x}, & \mathbf{y}, & \mathbf{e} & & \text{integer} \ .
\end{array}
\tag{21}
$$

Note that if d_i trailers must be delivered at EOL terminal i, at least $\lceil d_i/k \rceil$ tractor trips must visit that terminal ("$\lceil \cdot \rceil$" denotes the *ceiling* or upwards integer rounding function). Similarly, at least $\lceil -d_0/k \rceil$ tractor trips must be dispatched from the break in order to ship out the required number $-d_0$ of outbound trailers. A analogous argument applies to pickups, and so we can conclude that at least

$$
v_i \doteq \text{Max}\{\lceil |p_i|/k \rceil, \lceil |d_i|/k \rceil\} \tag{22}
$$

tractors must pass through terminal i, for $i=0,\ldots,n$. One way of expressing this restriction is

$$
\sum_{j \neq i} t_{ij} \geq v_i \qquad i=0,\ldots,n \ . \tag{23}
$$

Such "node activity" constraints are redundant and may be added to the basic aggregate flow formulation without altering it. However, when the linking constraints are dualized, they are no longer redundant: they alter the tractor part of the computation of $L(\mathbf{u})$ considerably. Instead of having

$$
\begin{array}{lrl}
\text{Min } & (\mathbf{c}-k\mathbf{u})^T\mathbf{t} & \\
\text{S.T.} & \mathbf{At} & = \mathbf{0} \\
& \mathbf{t} & \geq \mathbf{0} \ ,
\end{array}
\tag{24}
$$

with the restriction that t be integer left implicit, we have

$$
\begin{aligned}
&\text{Min } (\mathbf{c}-k\mathbf{u})^{\mathrm{T}}\mathbf{t} \\
&\text{S.T.} \quad \mathbf{At} = \mathbf{0} \\
&\qquad \sum_{j\neq i} t_{ij} \geq v_i \qquad i=0,\ldots,n \\
&\qquad \mathbf{t} \geq \mathbf{0} \\
&\qquad \mathbf{t} \quad \text{integer} \quad .
\end{aligned} \tag{26}
$$

Actually, a simple transformation of the network topology allows one to enforce these the extra constraints while staying within a network framework and thus preserving the automatic integrality of the solution (see [GM]). Take each node i

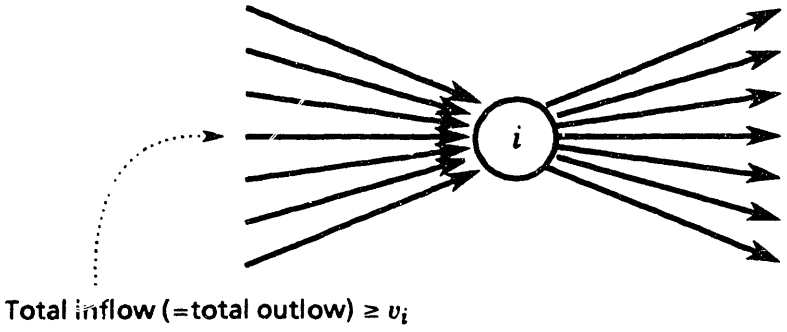

Figure 4a: Before node splitting

and split it into two with a single intervening link of minimum flow v_i and cost 0:

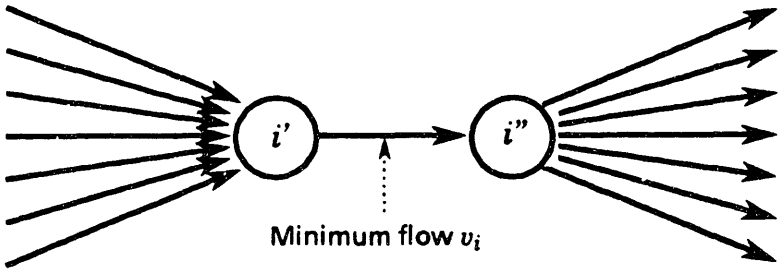

Figure 4b: After node splitting

As the v_i are integral, the optimum solution in the altered tractor network remains integral.

The additional constraints (23) are cutting planes that disallow the previous LP optimum

$$t^* = \frac{x^* + y^* + e^*}{k} \tag{27}$$

as constructed in the previous section, unless it is all integer (and hence optimal). The LP relaxation of the formulation is thus greatly strengthened, in general, by the addition of these superficially redundant constraints. At this point in our research, however, we do not completely understand how the altered LP relaxation behaves.

In practice, we have found that addition of these node activity constraints improves the median run times of the Lagrangean branch and bound method by roughly a factor of 5.

3.3 Implementation Details

We now describe in detail how each step in our Lagrangean branch and bound method is performed. Below, the word *subproblem* denotes a point on the enumeration tree: that is, the original integer program, together with some added separation constraints. An *active* subproblem is one which has not been further separated, and is thus an endpoint of the tree.

At this point is is helpful to keep in mind another picture of how the LBB method works. At any given time, each subproblem p has some best lower bound β_p on its LP-relaxed objective value. This lower bound is the highest value of $L(\mathbf{u})$ found for the subproblem, or any of its ancestors, for all the $\mathbf{u}$'s tried so far. The lowest value the global integer optimum could possibly have is

$$\beta \doteq \text{Min}\{\beta_p \mid p \text{ an active subproblem}\}. \tag{28}$$

The goal of the method is to squeeze this worst lower bound β and the upper bound z_{INC} (the objective value for the incumbent) together until they are near enough to conclude that z_{INC} is optimal or acceptably close to optimality.

3.3.1 Computing $L(\mathbf{u})$

Given a fixed $\mathbf{u}$ and subproblem p, $L(\mathbf{u})$ breaks down into four separate network optimizations. Therefore, we compute $L(\mathbf{u})$ by using primal network simplex four times. When doing the $\mathbf{t}$ part of this problem, the network is modified to enforce the node activity constraints discussed above. It should be noted that separation

constraints -- in our case, upper and lower bounds on arc flows -- can be added without disturbing network structure.

3.3.2 Fathoming

A subproblem is "fathomed" -- removed from further consideration -- if a lower bound for its objective value provided by an $L(\mathbf{u})$ computation is above, equal to, or sufficiently close to the upper bound on the global optimum that is provided by the incumbent.

Suppose the elements of the cost vector $\mathbf{c}$ have some common divisor Δ (for example, $\Delta = 1$ if $\mathbf{c}$ is integer). For any feasible solution, $\mathbf{t}$ is integer, and its cost $\mathbf{c}^T\mathbf{t}$ is divisible by Δ. Hence, we may strengthen any lower bound $L(\mathbf{u})$ by rounding it up to the next multiple of Δ, that is, taking $\Delta\lceil L(\mathbf{u})/\Delta \rceil$. Equivalently, one can easily see that if

$$z_{\mathrm{INC}} - L(\mathbf{u}) < \Delta \ , \tag{29}$$

where z_{INC} is the objective value of the incumbent, necessarily divisible by Δ, that the optimal objective value z_p for the subproblem must be greater than or equal to z_{INC}, and one cannot possibly improve upon the incumbent by pursuing this branch of the enumeration tree. Therefore, we fathom the subproblem if condition (29) is met.

We may also wish to fathom when the subproblem being considered offers at best a small gain over the incumbent, say by some small percentage P. Such a strategy does not guarantee that the final solution will be optimal, but only that it is within P percent of being so. The appropriate fathoming condition is

$$\Delta \left\lceil \frac{L(\mathbf{u})}{\Delta} \right\rceil \geq \left(1 - \frac{P}{100}\right) z_{\mathrm{INC}} \ . \tag{30}$$

By using this criterion and increasing the value of the user-specified parameter P, one can prune the enumeration tree more quickly, trading accuracy for speed of solution.

3.3.3 Generating Candidate Incumbents

A critical part of any LBB method is the heuristic that takes $L(\mathbf{u})$-based solutions and creates feasible integer solutions from them. This is the only way an incumbent

solution, and hence the algorithm's eventual final output, can be generated. It is naturally very desirable to come up with a good solution near the beginning of the run, thus increasing the chance that branches of the enumeration tree may be pruned close to the root, and mitigating the exponential growth of the number of subproblems.

In the k-CFP, one fairly obvious strategy presents itself. Given the (t, x, y, e) optimal in $L(\mathbf{u})$, we can imagine replacing t with the minimum-cost integer tractor flow vector $\mathbf{t_Z}$ required to support the trailer flows x, y, and e. On each link (i,j), this flow must be at least $(x_{ij}+y_{ij}+e_{ij})/k$, but also integer, hence at least $\lceil (x_{ij}+y_{ij}+e_{ij})/k \rceil$. However, we must also have *balanced* flow, hence $\mathbf{t_Z}$ should be the optimal solution to the problem

$$\begin{aligned} &\text{Min } \mathbf{c}^T\mathbf{t} \\ &\text{S.T. } \mathbf{At} = \mathbf{0} \\ &\qquad \mathbf{t} \geq \left\lceil \frac{1}{k}(\mathbf{x}+\mathbf{y}+\mathbf{e}) \right\rceil , \end{aligned} \tag{31}$$

where "$\lceil \cdot \rceil$" denotes the integer round-up operation applied to each element of a vector. Then ($\mathbf{t_Z}$, x, y, e) will be a feasible solution to the k-CFP. The optimization (31) is a network circulation problem, and may be solved by network simplex.

This round-up heuristic has, at least, the appealing quality that if the integer optimum x, y, and e are fed to it as input data, an integer optimum value of $\mathbf{t_Z}$ will result. As we shall see, however, such inputs are not likely in practice, and we will need a smarter incumbent generation procedure in order to have much hope of creating a practical method. The next chapter is devoted to the incumbent generation issue.

3.3.4 Detecting Slow Improvement

If, for a given subproblem, $L(\mathbf{u})$ shows little or no increase over several trial values of u, one assumes that the LP lower bound for the subproblem is too weak, and the subproblem must be split, that is, further separation constraints must be added.

The method decides that this slow improvement condition exists if at iteration $m > K$ we have

$$\frac{L(\mathbf{u})-z_{\mathrm{INC}}}{L(\mathbf{u}_0)-z_{\mathrm{INC}}} \geq 1-m\delta \ , \tag{32}$$

where $\mathbf{u}_0$ is the first multiplier vector used for the subproblem, and K and δ are adjustable parameters. In other words, we must have tried at least K iterations and achieved an average improvement of no more that a fraction δ of the way to the incumbent value per step.

The reason that a minimum of K iterations must be performed before separation is that one cannot rely upon $L(\mathbf{u})$ being increased at every step, even if it is far from its maximum value for the subproblem at hand. Without the $m>K$ restriction, a single "bad" step at the outset of subproblem analysis -- not an uncommon occurrence -- would cause an immediate, unnecessary separation. Every such mishap would double the work the method must do to fathom a particular branch of the enumeration tree. Separations are thus costly, and should be avoided unless they are absolutely necessary.

On small test problems, we have had the best practical success with $K=3$ and $\delta=0.02$.

3.3.5 Updating the Multipliers

If slow improvement is not detected, the algorithm alters $\mathbf{u}$ in an effort to increase $L(\mathbf{u})$. So far, we have found no procedure that is *guaranteed* to increase $L(\mathbf{u})$, so we have settled on the familiar subgradient method. The customary analysis gives that a valid subgradient is

$$\mathbf{g} \doteq \mathbf{x}+\mathbf{y}+\mathbf{e}-k\mathbf{t} \ , \tag{33}$$

and so we compute the new value of the multipliers via

$$\mathbf{u} := \mathbf{u}+s\mathbf{g} \ , \tag{34}$$

where the step s is given by

$$s = \alpha \frac{z_{INC}-L(\mathbf{u})}{\|\mathbf{g}\|^2} \ . \tag{35}$$

This is the basic subgradient method, where we have taken the "target" value of $L(\mathbf{u})$ to be z_{INC}. This is the highest possible target value one might choose, but it is

justified because, using the methods of the next chapter, high quality incumbents can be generated.

In practice, we have found that $\alpha = 1.0$ or $\alpha = 1.1$ seem to to give the best results.

Note that if, as the result of a subgradient step, we have for some (i,j),

$$u_{ij} > \frac{c_{ij}}{k} \quad , \tag{36}$$

then the imputed cost $c_{ij} - k u_{ij}$ of (i,j) in the Lagrangean tractor network will be negative. We have found that this entails a high risk of creating a negative cycle in the imputed tractor costs. If such a cycle exists, then $L(\mathbf{u}) = -\infty$, and it is a useless lower bound on the subproblem LP optimum. Ideally, one would want to restrict $\mathbf{u}$ to the dual feasible region

$$U \doteq \{\mathbf{u} \geq \mathbf{0} \mid \mathbf{c} - k\mathbf{u} \text{ has no negative cycles}\} \ . \tag{37}$$

This region is a polytope, but it has an exponentially growing number of constraints. We have found it easier to make the approximate restriction

$$\mathbf{0} \leq \mathbf{u} \leq \mathbf{c}/k \ , \tag{38}$$

which assures, more strongly, that $\mathbf{c} - k\mathbf{u}$ has no negative cost *arcs*. After each subgradient step, the resulting new value of $\mathbf{u}$ is projected onto this subset of the feasible region. Because this region is box-shaped, the projection computation is trivial: it is simply

$$u_{ij} := \text{Max}\{0, \text{Min}\{u_{ij}, \frac{c_{ij}}{k}\}\} \quad \forall (i,j) \ . \tag{39}$$

We have found that strategies other than projection for enforcing that $\mathbf{0} \leq \mathbf{u} \leq \mathbf{c}/k$, such as truncating the step, do not allow $L(\mathbf{u})$ to grow as quickly.

3.3.6 Choosing the First Value of u

When the algorithm is first started, there is only one subproblem: the original, full integer program without any separation constraints. The question is: what value of $\mathbf{u}$ should initially be used for this problem?

In the absence of node activity constraints, the obvious choice is the known optimal value, $\mathbf{c}/k$. When node activities are enforced, $\mathbf{c}/k$ is often not the best choice because it ensures that the "tractor contribution" $(\mathbf{c} - k\mathbf{u})^{\mathrm{T}}\mathbf{t}$ to the value of $L(\mathbf{u})$ is

zero, and the initial value of $L(\mathbf{u})$ will simply be the same value,

$$\frac{1}{k}\,\mathbf{c}^{\mathrm{T}}(\mathbf{x}^{*}+\mathbf{y}^{*}+\mathbf{e}^{*}) \quad , \tag{40}$$

as it is when $\mathbf{u}$ is chosen optimally without node activity constraints.

To shed some more light on this matter, redefine $\mathbf{t}^{*}$ to be the optimum of

$$\begin{aligned} \text{Min} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{t} \\ \text{S.T.} \quad & \mathbf{At} = \mathbf{0} \\ & \sum_{j \neq i} t_{ij} \geq v_i \quad i=0,\ldots,n \\ & \mathbf{t} \geq \mathbf{0} \quad , \end{aligned} \tag{41}$$

and $\mathbf{x}^{*}$, $\mathbf{y}^{*}$, and $\mathbf{e}^{*}$ to be the respective optima in the LP relaxation of the formulation (2), namely

$$\begin{aligned} \text{Min} \quad & \frac{1}{k}\mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{S.T.} \quad & \mathbf{Ax} = -\mathbf{d} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{42}$$

$$\begin{aligned} \text{Min} \quad & \frac{1}{k}\mathbf{c}^{\mathrm{T}}\mathbf{y} \\ \text{S.T.} \quad & \mathbf{Ay} = \mathbf{p} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{43}$$

$$\begin{aligned} \text{Min} \quad & \frac{1}{k}\mathbf{c}^{\mathrm{T}}\mathbf{e} \\ \text{S.T.} \quad & \mathbf{Ae} = \mathbf{d}-\mathbf{p} \\ & \mathbf{e} \geq \mathbf{0} \end{aligned} \tag{44}$$

Scaling all the cost vectors by some positive constant will not change the optimality of any of $\mathbf{t}^{*}$, $\mathbf{x}^{*}$, $\mathbf{y}^{*}$, and $\mathbf{e}^{*}$, and will appropriately scale the corresponding objective values. Recalling that in this initial, unseparated problem, we have

$$
\begin{aligned}
L(\mathbf{u}) = \text{Min } & (\mathbf{c}-k\mathbf{u})^{\mathrm{T}}\mathbf{t} + \mathbf{u}^{\mathrm{T}}\mathbf{x} + \mathbf{u}^{\mathrm{T}}\mathbf{y} + \mathbf{u}^{\mathrm{T}}\mathbf{e} \\
\text{S.T.} \quad & \mathbf{At} = \mathbf{0} \\
& \mathbf{Ax} = -\mathbf{d} \\
& \mathbf{Ay} = \mathbf{p} \\
& \mathbf{Ae} = \mathbf{d}-\mathbf{p} \\
& \sum_{j \neq i} t_{ij} \geq v_i \qquad i=0,\ldots,n \\
& \mathbf{t}, \quad \mathbf{x}, \quad \mathbf{y}, \quad \mathbf{e} \geq \mathbf{0} \quad ,
\end{aligned}
\tag{45}
$$

we can conclude that if we set $\mathbf{u}=(\lambda/k)\mathbf{c}$, where $\lambda \epsilon (0,1)$, we will get

$$
\begin{aligned}
L(\mathbf{u}) &= (\mathbf{c}-\lambda\mathbf{c})^{\mathrm{T}}\mathbf{t}^* + \frac{\lambda}{k}\mathbf{c}^{\mathrm{T}}\mathbf{x}^* + \frac{\lambda}{k}\mathbf{c}^{\mathrm{T}}\mathbf{y}^* + \frac{\lambda}{k}\mathbf{c}^{\mathrm{T}}\mathbf{e}^* \\
&= (1-\lambda)\mathbf{c}^{\mathrm{T}}\mathbf{t}^* + \frac{\lambda}{k}\mathbf{c}^{\mathrm{T}}(\mathbf{x}^*+\mathbf{y}^*+\mathbf{e}^*) \quad .
\end{aligned}
\tag{46}
$$

Letting $\lambda \to 0$ or $\lambda \to 1$, we can conclude that a valid lower bound on z_{IP}, the integer program global optimum, is

$$
\text{Max}\left\{\mathbf{c}^{\mathrm{T}}\mathbf{t}^*, \frac{1}{k}\mathbf{c}^{\mathrm{T}}(\mathbf{x}^*+\mathbf{y}^*+\mathbf{e}^*)\right\} \quad . \tag{47}
$$

If $\mathbf{c}^{\mathrm{T}}\mathbf{t}^* > (\mathbf{c}/k)^{\mathrm{T}}(\mathbf{x}^*+\mathbf{y}^*+\mathbf{e}^*)$, it seems better to choose λ near zero, hence $\mathbf{u}$ near 0, rather than λ near 1 and $\mathbf{u}$ near $\mathbf{c}/k$. Computational experiments have shown that the value of λ strongly effects solution time, but we do not currently see a clear pattern in the results. Our best *average* times for the central 2-CFP have been with $\mathbf{u}$ initially set to $\mathbf{c}/4$, that is, $\lambda = \frac{1}{2}$ and $\mathbf{u}$ at the exact center of the box $\{\mathbf{u} \mid 0 \leq \mathbf{u} \leq \mathbf{c}/2\}$ approximating the dual feasible region.

Also, one should note that the above analysis, in particular equation (46), does not work when $\mathbf{u}$ is not proportional to $\mathbf{c}$. Thus, the above observations only apply in the very beginning of the method, before the starting multiplier vector $\mathbf{u}=(\lambda/k)\mathbf{c}$, which *is* proportional to $\mathbf{c}$, is altered by a subgradient step.

For newly-separated subproblems, the first value tried for $\mathbf{u}$ is the one that yielded the highest $L(\mathbf{u})$ in the immediate parent subproblem.

3.3.7 Separation

If slow improvement is detected, the current problem is split in two. Our model has unbounded nonnegative integer variables, as opposed to 0-1 variables, so we cannot eliminate variables from consideration. We can only split the problem in two and require that some primal decision variable v be less than or equal to some integer m in one child, and greater than or equal to $m+1$ in the other.

In practice, we have so far found that it seems most efficient to split on the tractor variables t_{ij}, rather than the trailer variables x_{ij}, y_{ij}, or e_{ij}. However, this experience was gained from experiments preceding our introduction of node activity constraints, and so is somewhat suspect.

One way of understanding this result, at least in the absence of node activity or separation constraints on t, is that we know that the optimum, x, y, and e values are always integer. Hence, imposing a restriction that some element of one of these vectors be at most m or at least $m+1$ will not "cut away" the LP optimum from consideration. This argument suggests that well-chosen splits on the t_{ij}, will be much stronger that splits on the x_{ij}, y_{ij}, and e_{ij}. Certainly, trailer splits alone, without node activity forcing or tractor splits, would be totally ineffectual.

It is less clear what strategy is best in the presence of node activity constraints. We have had good experience with the following simple heuristic: split on the t_{ij} for the longest arc (i,j) for which the number of trailers in the current $L(\mathbf{u})$ solution is odd. We thus add the constraint

$$t_{ij} \leq m \quad , \tag{48}$$

to one child subproblem, and

$$t_{ij} \geq m+1 \tag{49}$$

to the other. The value of m is chosen to be $\lfloor x_{ij}+y_{ij}+e_{ij} \rfloor$, unless t_{ij} has been otherwise constrained (the "$\lfloor\cdot\rfloor$" denotes the *floor* or integer round-down function).

The separation scheme used for our actual computer runs was a little more complicated than that described above. We used a "scoring" method for choosing the splitting arc. Essentially, each arc with odd trailer flow was assigned a "score" based on its length, whether it emanated from or terminated in the break, and

whether it had been separated on before. The algorithm then split on the arc with the highest score. After much adjustment of parameters, this method provided about a ten percent run time improvement over the simpler "longest odd arc" strategy. In view of these rather modest gains, we will not burden the reader with further details.

For the "lower" ($t_{ij} \leq m$) child, where we have at most m tractors allowed on some arc (i,j), we can deduce with certainty that there can be at most km trailers of each kind on (i,j). We can therefore add the following redundant constraints:

$$\begin{aligned} x_{ij} &\leq km \\ y_{ij} &\leq km \\ e_{ij} &\leq km \quad . \end{aligned} \tag{50}$$

These simple upper bounds strengthen the Lagrangean relaxation, and can be enforced without breaking network structure.

3.3.8 Subproblem Selection

After the algorithm has fathomed or separated a subproblem, it is faced with the decision of which subproblem to try next (unless there are none left unfathomed, in which case it terminates). We have found that the best strategy is to first process one of the problems with the lowest β_p. Not surprisingly, this yields the fastest convergence between the overall lower bound β and the incumbent value z_{INC}.

We also have a feature that allows the method to switch to high-β_p subproblems in the event that there are so many active subproblems that the program is close to exhausting its virtual memory allocation. The intent is that these subproblems may be fathomed relatively quickly, freeing up memory for the more important ones. This feature allows the algorithm to handle larger problems with a given amount of memory, albeit with some speed penalty for larger instances. If we had been working on a much larger computer, or if memory resources had been very cheap, we would perhaps not have needed this high-β_p feature.

3.3.9 Basis Preservation

In our computer implementation, we used a number of tricks to improve run times. These tricks involve using information from earlier network simplex bases to speed up calls to the network simplex code.

When $L(\mathbf{u})$ is computed, four network simplex optimizations must be performed. At each subgradient iteration, we use the previous four optimal bases for the subproblem as the four starting bases in the recomputation of $L(\mathbf{u})$. The hope is that the optimal bases for two consecutive values of $\mathbf{u}$ should resemble one another, and so fewer pivots will be needed than if we were to construct all initial bases "from scratch".

Actually, we go further than the above trick: when we first compute $L(\mathbf{u})$ for a subproblem, we essentially start with the four bases that were optimal in the last iteration of its parent. The difficulty responsible for the "essentially" is that the added separation constraint may make one or more of the old optimal bases infeasible for the child. There is, however, a "fix": suppose, for example, we have added a constraint $t_{ij} \leq m$, and that the parent problem currently has $t_{ij} = r > m$, rendering the parent basis infeasible in the offspring. Now, all our network representations contain a "super transhipment" node connected to all other nodes by "artificial" arcs of very high ("big M") cost. To maintain feasibility in the child, we perturb the flows as follows:

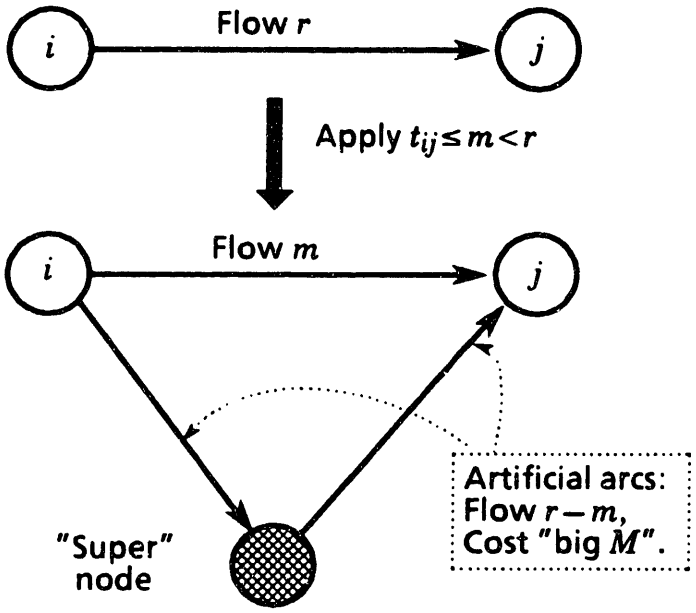

Figure 5: Basis modification for a violated upper bound

By also setting the maximum flow *capacity* of the two artificial arcs to $r-m$, we can avoid having to add them to the basis (although either of them could already be in the basis in a degenerate manner), and the original spanning tree of the parent basis remains valid in subproblem's perturbed network. Of course, when the

network simplex routine is called, the artificial arcs will immediately have flow removed from them, because they have such high costs.

Analogous techniques can be used for the addition of constraints of the form $t_{ij} \geq m+1$, and also for the x, y, and e bases.

We could have retained old basis information without perturbations by using a *dual* method to reoptimize the offspring of a subproblem, as in regular branch and bound methods for integer programming, but this would have required the implementation of both primal *and* dual network simplex algorithms.

The drawback to using basis-preservation tricks is that basis information must be stored for all active subproblems, increasing program memory requirements. In retrospect, it might have been wiser to make this feature optional, allowing more active subproblems to exist simultaneously. Though there would be a slight additional overhead of a few extra pivots per subproblem evaluation, one could use memory more efficiently and thereby stave off the switch to the high-β_p mode of subproblem selection. This might in turn result in faster overall convergence.

However, there are other possible ways to save memory. In a more sophisticated programming environment, one could envision dynamically trading off basis storage space for additional subproblems as memory becomes tight. Also, one could make some use of old basis information without any storage penalty by switching to a depth-first tree exploration strategy: one could use the parent basis for *one* of the children of a subproblem without having to allocate any more memory, as long as that one child is analyzed immediately after separation.

4. Incumbent Generation

This chapter is devoted to the most complicated part of the Lagrangean branch and bound algorithm, the incumbent generator. This is a heuristic that must take solutions to the computation of $L(\mathbf{u})$ and alter them so that they are globally feasible, and hence possible candidate to replace the current incumbent. We first explain the deficiencies of the simple round-up heuristic proposed in the previous chapter, and then outline two more intelligent strategies.

4.1 Why Round-Up is not Enough

Earlier, we mentioned that a simple round-up incumbent generation strategy is inadequate. To demonstrate this, consider the simple example problem

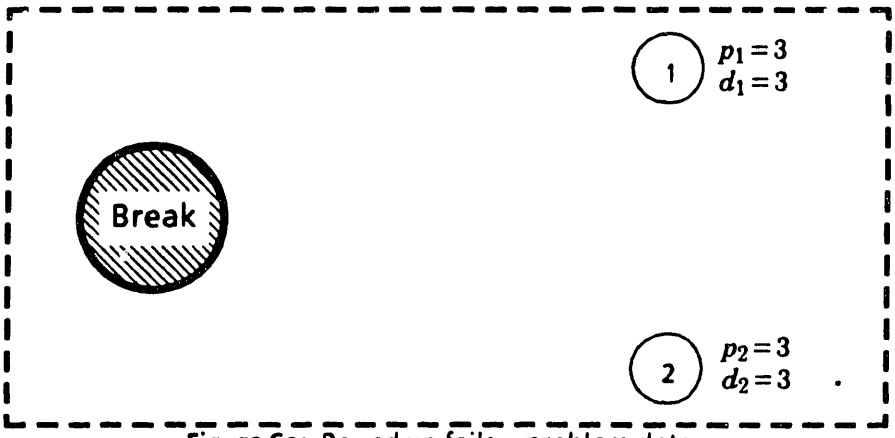

Figure 6a: Round up fails -- problem data

One of the two optimal solutions is

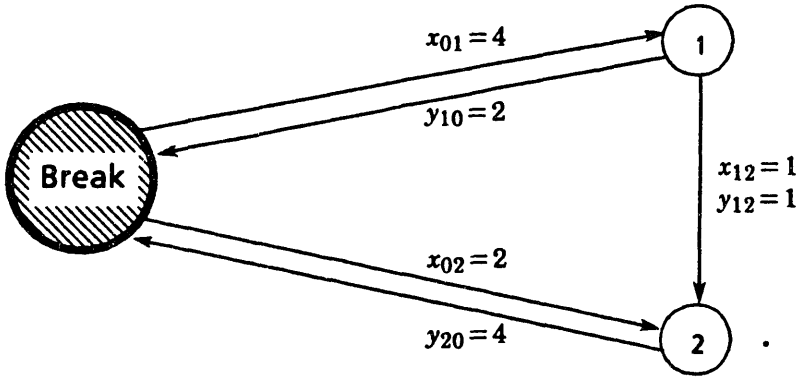

Figure 6b: Round up fails -- integer optimum

In the other optimum, the roles of EOLs 1 and 2 are interchanged.

Suppose our algorithm separates only on the t_{ij} (tractor) variables. Then, as we compute $L(\mathbf{u})$, all *trailers* will still always be free to take the shortest path to their final destinations. For instance, all outbound traffic for terminal 2 could take the route

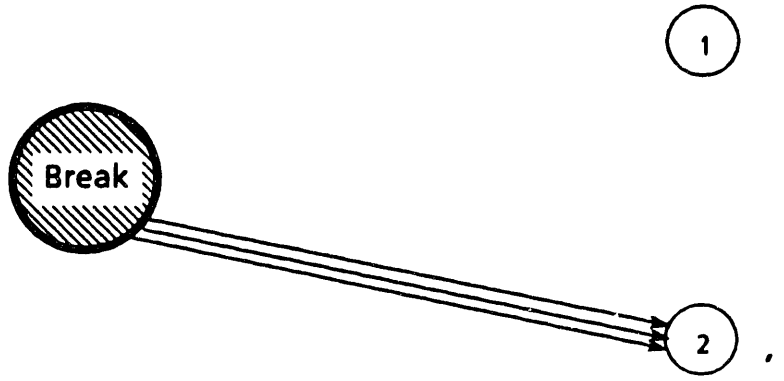

Figure 6c: Round up fails -- one possible routing from break to 2 in $L(\mathbf{u})$

or, with a different **u**,

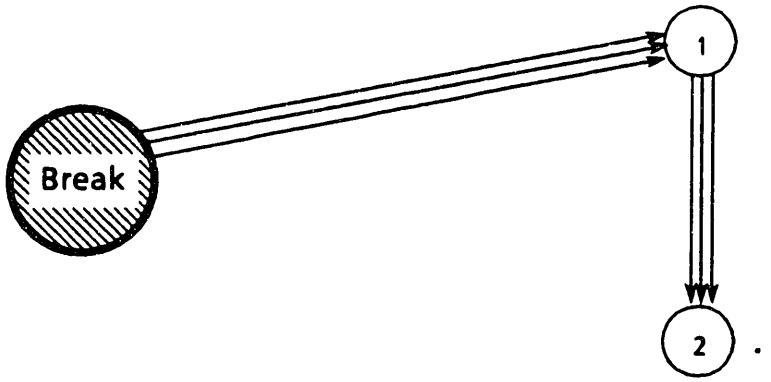

Figure 6d: Round up fails -- another possible routing from break to 2 in $L(\mathbf{u})$

However, in the IP optimum we have routings in which two *different* paths are used, such as

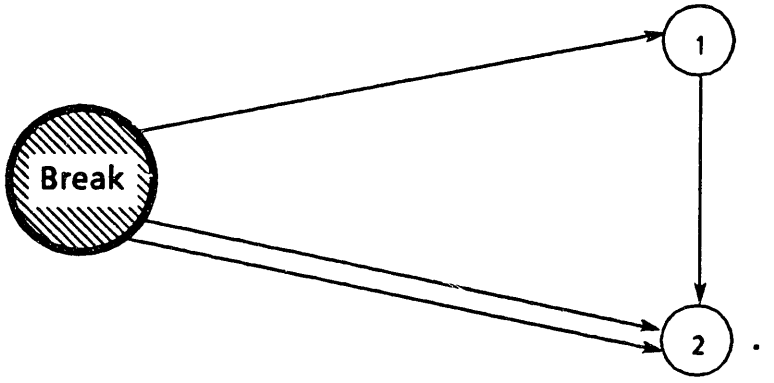

Figure 6e: Round up fails -- a routing that cannot occur in $L(\mathbf{u})$

The simple round-up heuristic described in the previous chapter never changes any trailer routes, so it can never discover the optimum.

In order to detect the optimum, we must do one (or both) of two things: separate on trailer variables, or create an incumbent finder that intelligently alters trailer routes. The former strategy puts most of the burden of finding a solution on the enumeration component of the algorithm -- a very dangerous course. We have therefore concentrated on the second option.

The techniques we will now discuss have been developed *solely* for the central 2-CFP, and take advantage of that problem's special structure. Some of them may possibly be generalized, with difficulty, to more complicated CFP problems.

4.2 A Simple Local Improvement Scheme

The incumbent generation heuristic should take the current $L(\mathbf{u})$ solution as input. Its output should be sensitive to that input, for otherwise the subgradient component of the method will be doing only half the work that is should. That is, it will be helping to supply lower bounds so that subproblems may be fathomed, but it will leave the detection of the actual optimum to the combined efforts of the separation and incumbent-finding rules. Ideally, the multiplier updating scheme should, so to speak, be "pointing the incumbent finder in the right direction", and thus contributing not only to raising the lower bound on the objective, but also to the lowering of the upper (incumbent) bound.

Consequently, we first developed a *simple local improvement* heuristic that takes the basic form of the $L(\mathbf{u})$ solution, and makes some local modifications to it. Our idea was to make slight perturbations to the trailer routings, so that the minimum integer tractor movements needed to "cover" them would be reduced. The $\mathbf{t}$ part of the $L(\mathbf{u})$ solution is ignored. We stuck to very simple perturbations, because we wished to produce a heuristic that would run quickly. In the combined $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ solution to $L(\mathbf{u})$, the method detects all patterns of the following form, where solid arrows represent arcs with an *odd* total number of trailers:

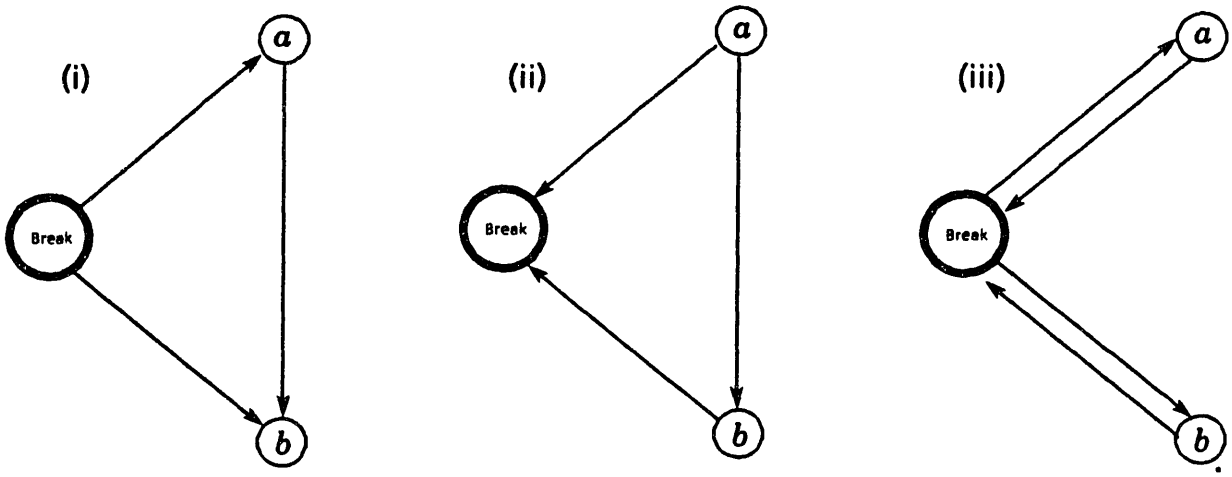

Figure 7: Patterns recognized by the simple local improvement heuristic

The following local improvements are applied, respectively, to each of the three patterns above (dotted lines represent arcs whose trailer flows have been increased from odd to even values, and boxes represent particular trailers):

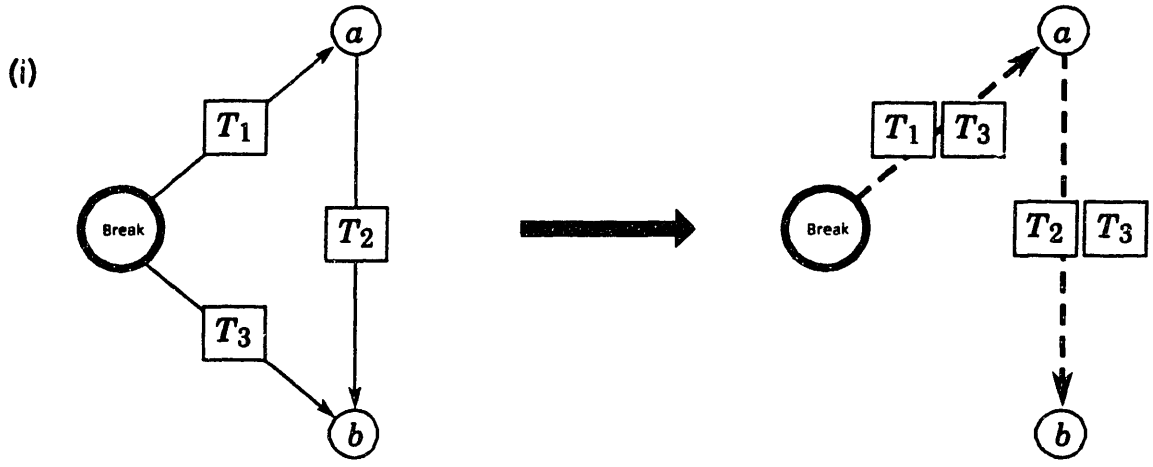

Figure 8a: Local improvement for pattern (i)

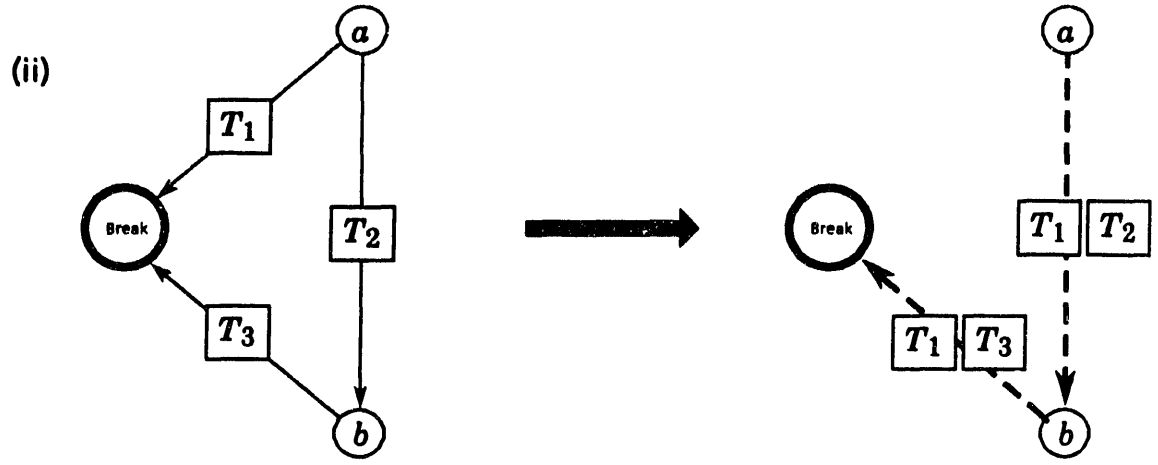

Figure 8b: Local improvement for pattern (ii)

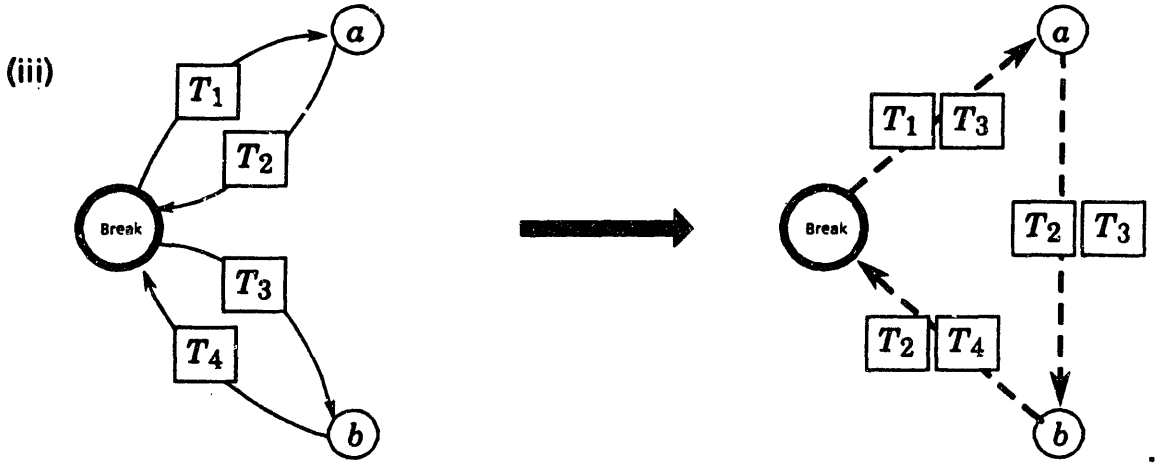

Figure 8c: Local improvement for pattern (iii)

The heuristic ranks these modifications by their total savings, that is, c_{0b} in the first case, c_{a0} in the second, and

$$(c_{0a}+c_{a0}+c_{0b}+c_{b0})-(c_{0a}+c_{b0}+c_{ab}) = c_{a0}+c_{0b}-c_{ab} \geq 0 \qquad (51)$$

in the third. It then applies these improvements in a greedy manner, highest savings first, until no more can be implemented. Note that because some of the detected patterns might overlap, performing one might preclude applying some others. We have not yet experimented with more sophisticated schemes for deciding which combinations to apply. After these modifications are made, the flow-based round up procedure (31) is performed to compute the t part of the candidate incumbent, this time using the *modified* trailer flows as input. The candidate so constructed replaces the current incumbent if it has lower total cost.

4.3 Slacks and Bobtailing

The simple local improvement scheme just described has some weaknesses. Consider the problem

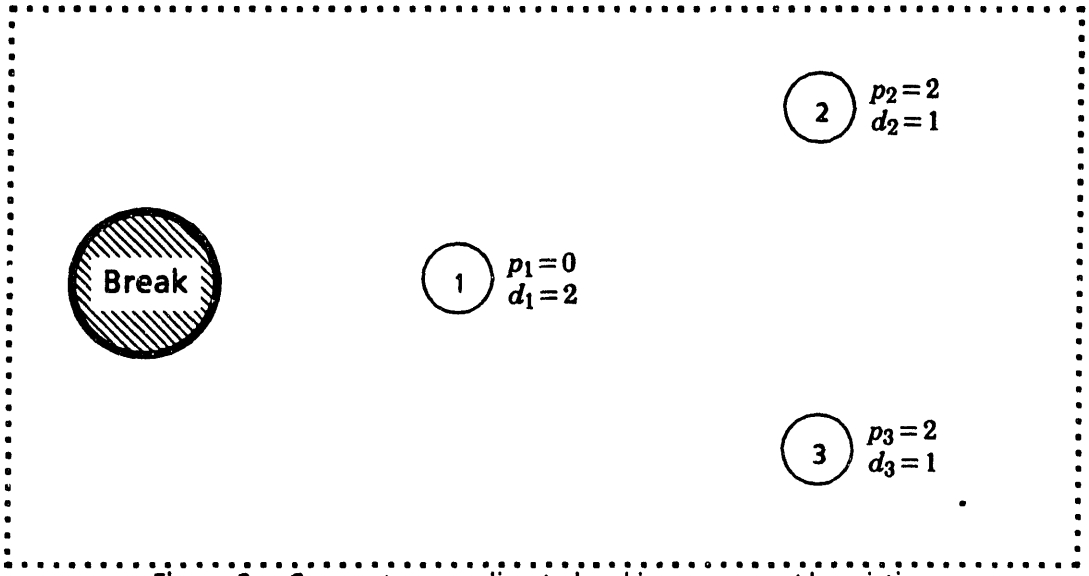

Figure 9a: Case not responding to local improvement heuristic

Its LP solution (in the absence of node activity constraints) looks like this:

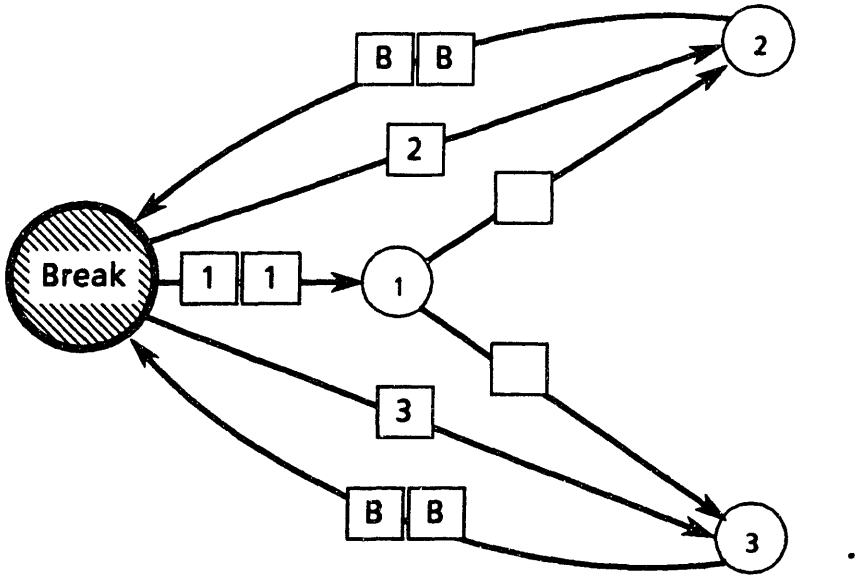

Figure 9b: LP solution to figure 9a

None of the simple patterns (i)-(iii) defined above are present in this solution. However, the two elements

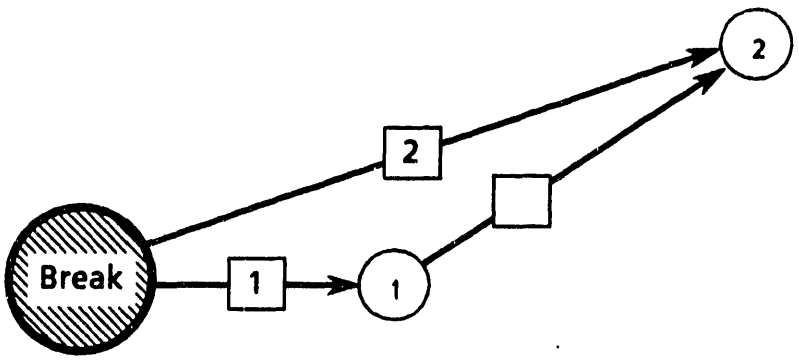

Figure 9c: Local improvement pattern "hidden" in figure 9b

and

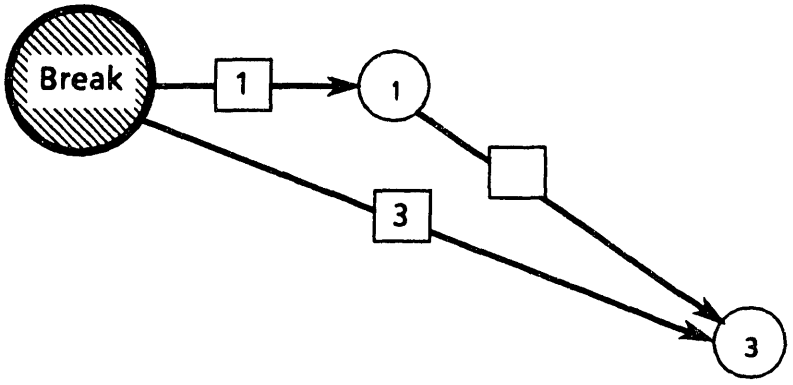

Figure 9c: Second local improvement pattern "hidden" in figure 9b

which both fit pattern (i), *are* present, but they have been collapsed together so that the number of trailers on arc (0, 1) is *even*, not odd, and therefore both patterns would go unrecognized in the simple local improvement scheme outlined above. There is still something unusual about node 1, however: if we were just to put enough tractors on each link to handle the flow present, without worrying about balance, we would have one tractor entering node 1, and two leaving it. To get a *balanced* tractor movement vector t, we will have to bring another tractor into node 1.

This example introduces another idea: why not compute the minimum tractor covering *first* and then look for patterns in the *slack*, or excess trailer capacity, along each arc? That is, after computing the minimum integer t needed to support the trailer movements x, y, and e, compute the slack (recall that we now assume $k=2$)

$$\mathbf{s} \doteq 2\mathbf{t}-\mathbf{x}-\mathbf{y}-\mathbf{e} \ . \tag{52}$$

Then, s_{ij} is the room available for extra trailer movements on arc (i,j). We can then

look for local improvement patterns resembling (i)-(iii) above in this vector of slacks.

For instance, we might now expect that the minimum cost integer tractor flow t needed to support the LP relaxation trailer flows for the current example might look like:

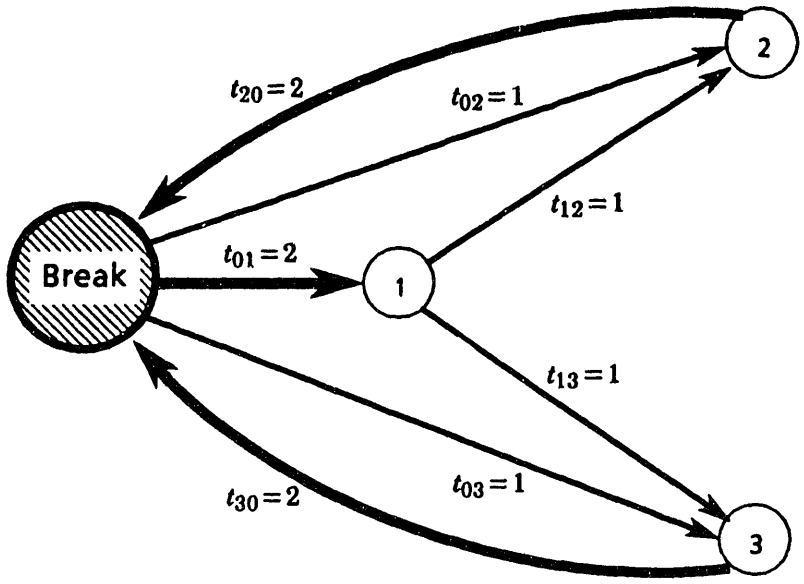

Figure 10a: "Intuitive" tractor flows needed to cover trailer flows in figure 9b

The reader may confirm that this would imply a slack vector of

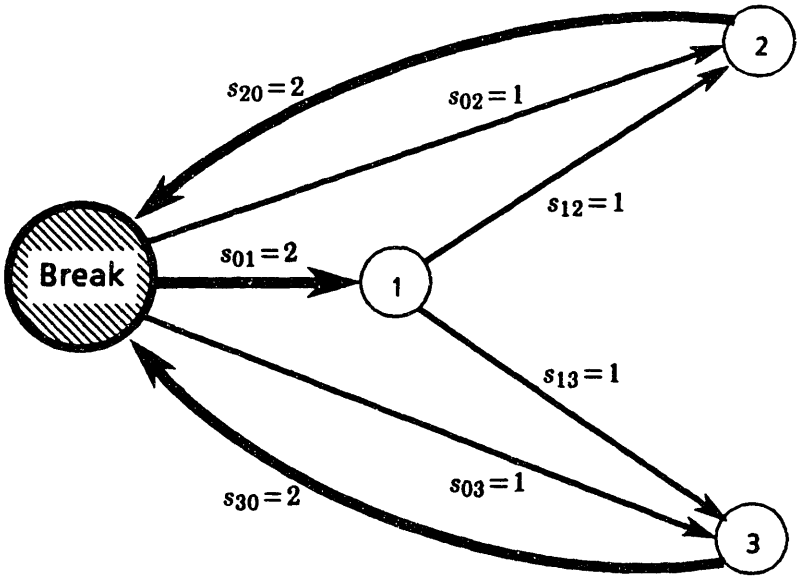

Figure 10b: Slacks corresponding to figures 9b and 10a

One could now detect, among other things, that we have the slack patterns

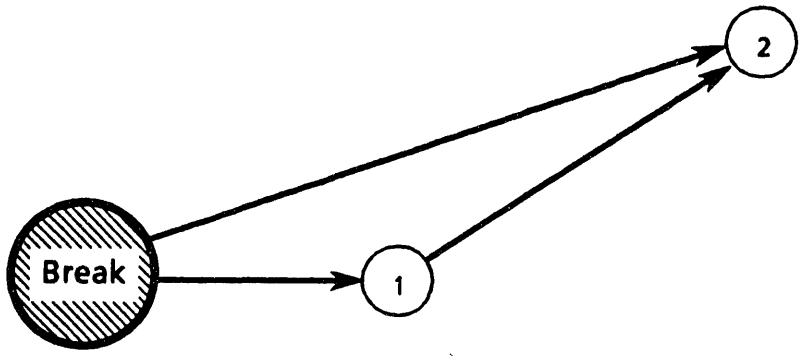

Figure 10c: Pattern detectable in figure 10b

and

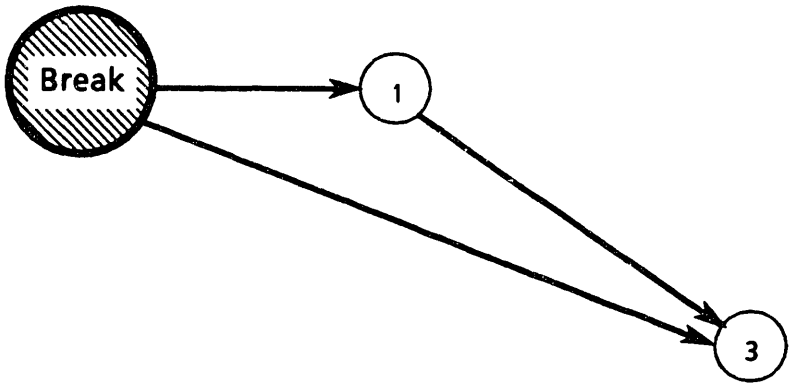

Figure 10d: Second pattern detectable in figure 10b

added together. One could then realize that because we have excess capacity on the paths 0-1-2 and 0-1-3, we can eliminate the direct 0-2 and 0-3 tractor trips by sending the corresponding deliveries via node 1. This yields the solution

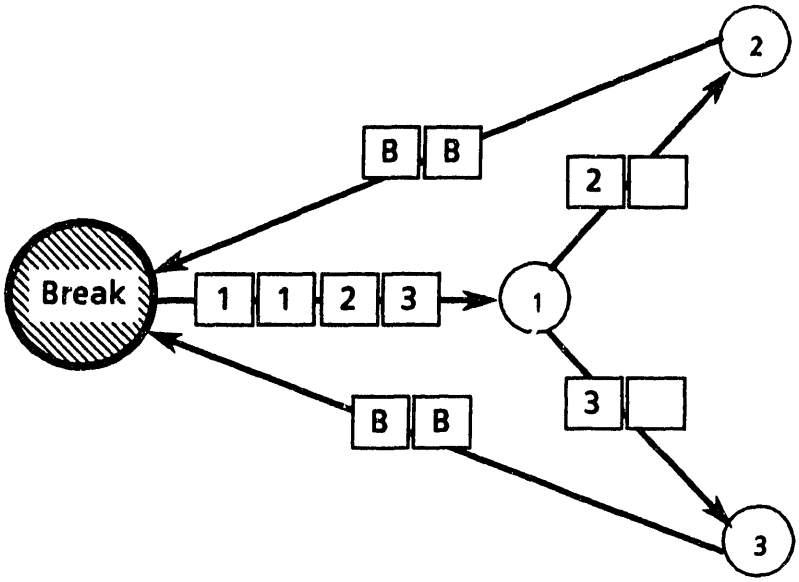

Figure 11: Optimum integer solution to problem 9a data

which is the IP optimum.

However, there is a slight problem. The *actual* minimum cost tractor flow to cover the LP trailer movements is in fact the following:

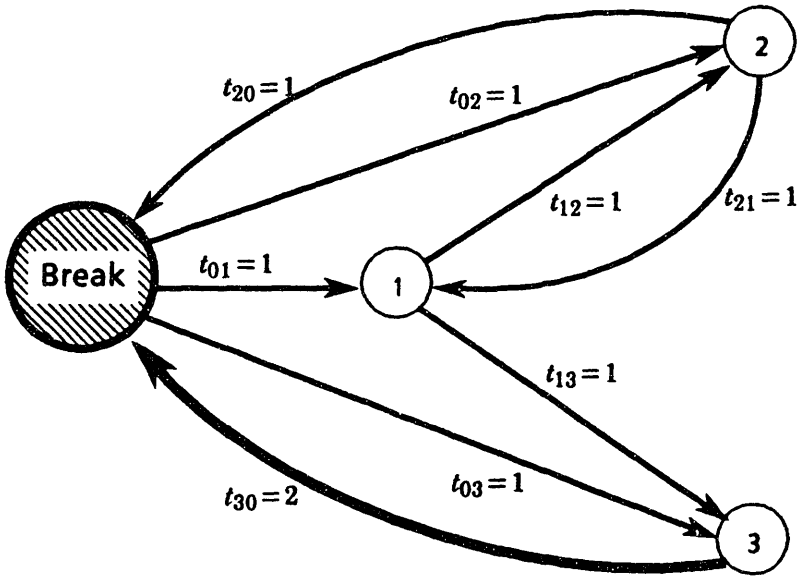

Figure 12a: Actual minimum cost tractor flows to cover figure 9b trailers

That is, the cheapest way to bring the required extra tractor into node 1 is not to send it from the break, but to "bobtail" it -- run it with no trailers -- from node 2. (Actually, if the system is as symmetric as it appears, the minimum tractor round-up

is not unique: we can also bobtail from node 3 instead of node 2.) The corresponding slack vector is

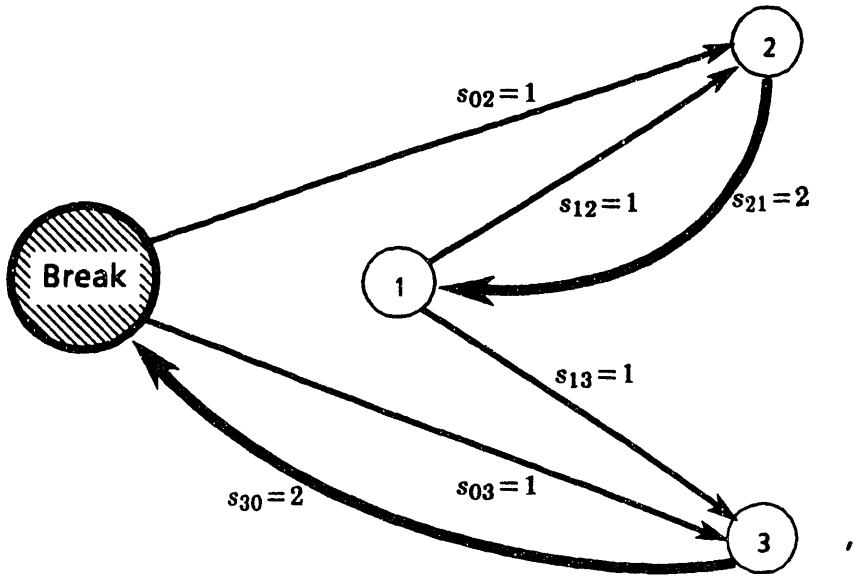

Figure 12b: Actual slacks for figures 9b and 12a

which does not display such simple local improvement patterns of the figure 10b slack vector that we would have preferred.

A solution to this difficulty is to perform the tractor round up additionally specifying that tractors can *only* be added to arcs that already have some trailers. That is, given **x**, **y**, and **e** from the solution of $L(\mathbf{u})$, let $\mathbf{t}_S$ be the solution of

$$\begin{aligned} &\text{Min } \mathbf{c}^T\mathbf{t} \\ &\text{S.T. } \mathbf{At} = \mathbf{0} \\ &\qquad \mathbf{t} \geq \left\lceil \frac{1}{k}(\mathbf{x}+\mathbf{y}+\mathbf{e}) \right\rceil , \\ &\qquad t_{ij} = 0 \quad \forall (i,j) \text{ with } x_{ij}+y_{ij}+e_{ij}=0 . \end{aligned} \tag{53}$$

This problem again has network structure, and can be solved by network simplex. The "constrained" round up tractor flow $\mathbf{t}_S$ seems in practice to be a much better basis for local improvements than the simple minimum cost round-up tractor flow $\mathbf{t}_Z$. Although we do not have a rigorous theoretical argument to back up this observation, an intuitive justification is that the bobtailing movements that often occur in $\mathbf{t}_Z$ are opposite to the prevailing flow of trailers, and the slack capacity they create therefore points in "useless" directions.

4.4 Cycle Splicing

Given $\mathbf{t_S}$, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$, how can one make local improvements? We have adopted what we call a *cycle splicing* heuristic. Since $\mathbf{A}\mathbf{t_S} = \mathbf{0}$, one can conclude that if we define, much as before,

$$\mathbf{s} \doteq 2\mathbf{t_S} - \mathbf{x} - \mathbf{y} - \mathbf{e} \quad , \tag{54}$$

we then have $\mathbf{s} \geq \mathbf{0}$ and

$$\begin{aligned} \mathbf{A}\mathbf{s} &= 2\mathbf{A}\mathbf{t_S} - \mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{y} - \mathbf{A}\mathbf{e} \\ &= \mathbf{0} + \mathbf{d} - \mathbf{p} - (\mathbf{d} - \mathbf{p}) \\ &= \mathbf{0} \end{aligned} \quad . \tag{55}$$

Thus, the vector of slack capacities $\mathbf{s}$ may also be considered to be a balanced "flow" in the inter-terminal network. Furthermore, by the construction of $\mathbf{t_S}$, any link (i,j) for which $s_{ij} > 0$ necessarily has some trailers on it.

Since $\mathbf{t_S}$ and $\mathbf{s}$ are balanced flows, they can be expressed as a sum of flows around directed cycles. We define a *simple* cycle to be one that repeats no arcs or nodes. It is clear that any set of cycles may be further decomposed into simple cycles. For now, assume that some decomposition of $\mathbf{s}$ into simple, directed cycles has been given. (We will show later that if $\mathbf{u}$ obeys the strict triangle inequality and there are no separation constraints on $\mathbf{x}$, $\mathbf{y}$, or $\mathbf{e}$, such a decomposition is essentially unique, so there is no issue of choosing the *best* one.)

To be precise, we give the following definition:

4.4.1 Definition. A *decomposition* of some balanced flow $\mathbf{w}$ ($\mathbf{w} \geq \mathbf{0}$, $\mathbf{A}\mathbf{w} = \mathbf{0}$) *into simple cycles* is a sequence $R_1, \ldots, R_J$ of (not necessarily distinct) simple directed cycles such that

$$\sum_{l=1}^{J} \mathbf{f}(R_l) = \mathbf{w} \quad , \tag{56}$$

where $\mathbf{f}(R_l)$ denotes a flow vector that has a "1" in each position (i,j) corresponding to an arc of R_l, and zeroes in all other positions.

One should also note that, disallowing the unlikely case that $\mathbf{c}$ has zero-cost cycles, the vector $\mathbf{s}$ will always decompose into a sequence of *distinct* simple cycles: if

there were a slack of two or more around some cycle R_l, which would occur if that cycle were to be repeated in the decomposition, then t_S would not be a minimum-cost solution to (53): the solution $t_S - f(R_l)$ would be a cheaper alternative. We conclude that on any cycle R_l in a decomposition of s, there is at least one arc (i,j) for which $s_{ij}=1$.

Now consider two simple cycles R and S in the decomposition of s having the property that they cross only at the break. We now describe a procedure for "splicing" R and S together. Let (i,j) be any arc of R. Let a be the node in S directly after the break, and b the node directly before the break. To splice S into arc (i,j) of R, we

(a) Remove one tractor from arc $(0,a)$ in S. This may leave one trailer "uncovered" -- that is, with no possible tractor to haul it. If so, call this trailer T_1.

(b) If necessary, reroute T_1 from the break to i, via the arcs of R. This step will require no extra tractor movements, as there is positive slack all the way around R. Note that if there is no uncovered trailer T_1, or if $i=0$, this step requires no actual operations.

(c) Remove one tractor from (i,j). This may uncover another trailer, which we shall denote by T_2.

(d) Place a tractor on (i,a), and use it, if needed, to carry T_1 and/or T_2.

(e) Reroute T_2 from a to b, using the excess capacity along S. If no trailer T_2 has been uncovered, this step is null.

(f) Remove a tractor from $(b,0)$. This could uncover a third trailer, T_3.

(g) Place a tractor on (b,j) and use it, as necessary, to carry T_2 and/or T_3.

(h) Route T_3 (if it exists) through the remainder of R (if any; we might have $j=0$) to the break, using the available slack capacity.

The reader may confirm that the trailer flows remain balanced, and that each of the trailers T_1, T_2, and T_3 whose routes may have been altered are still delivered to their final destinations. Also, R and S are no longer valid slack cycles in the modified solution (recall that R and S must each have an arc with a slack of exactly one, and the amount of slack has been reduced all the way around both cycles). The

decrease in tractor cost is

$$\Delta z = c_{0a} + c_{ij} + c_{b0} - c_{ia} - c_{bj} \quad , \tag{57}$$

which may be positive, negative, or zero.

As an example, consider

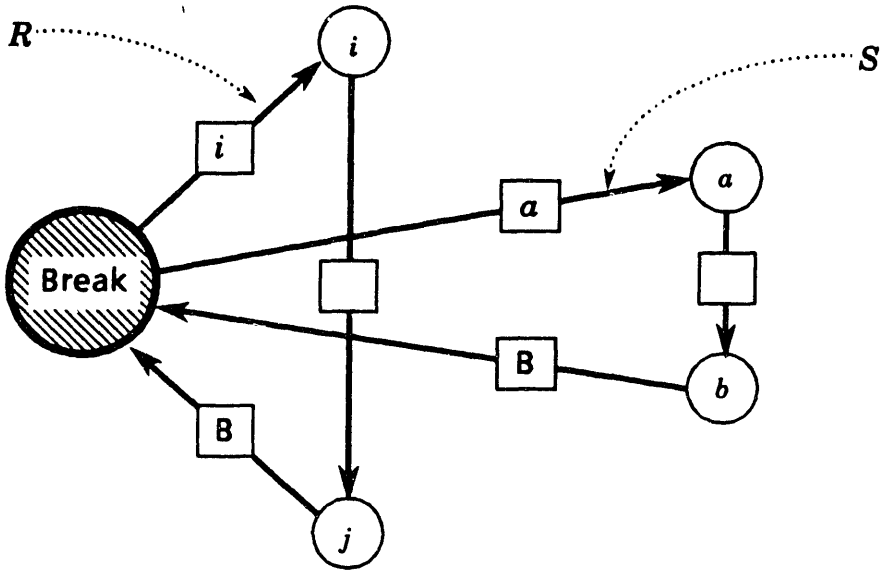

Figure 13a: Two cycles ready for splicing

In this case, T_1 is an outbound trailer to be delivered at a, T_2 is the empty exchanged by i and j, and T_3 is a pickup emanating from b. The splicing operation yields

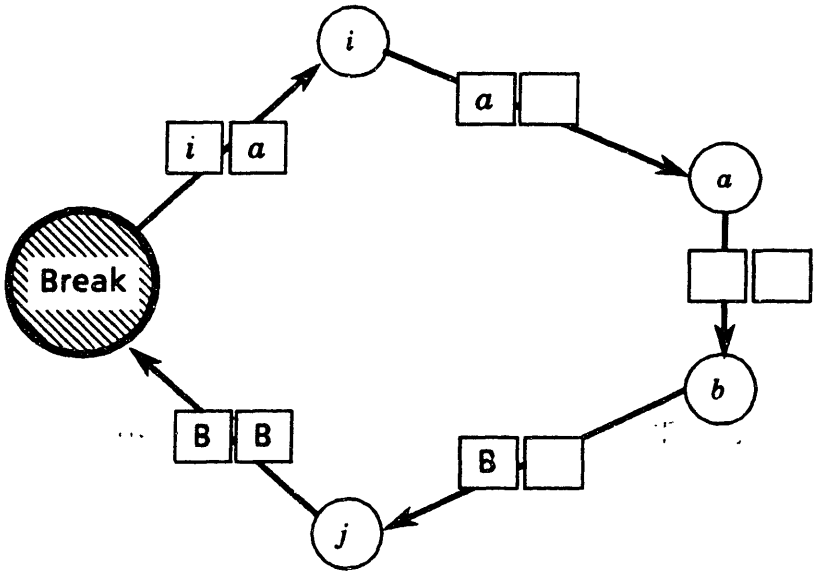

Figure 13b: Outcome of the splicing operation

for a substantial improvement in cost, if the distances are roughly as pictured.

The above splicing methodology can be extended to the case in which R and S share some arcs, but there is a slack of at least 2 on all shared arcs (this *must* be the case if R and S are members of a set of cycles that sum to s). Essentially, each non-shared part of one cycle must spliced into some non-shared arc of the other, and the number of tractors on all shared arcs is reduced by one. For example, consider the problem

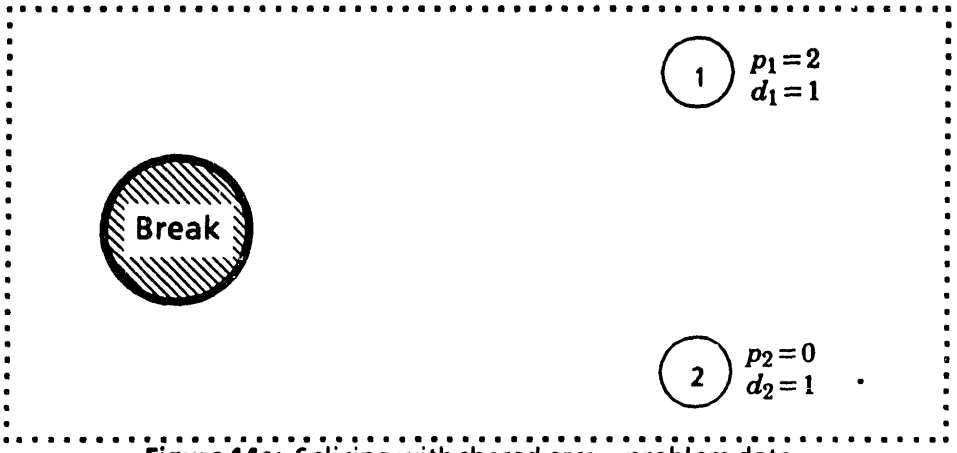

Figure 14a: Splicing with shared arcs -- problem data

The LP solution (ignoring node activity constraints) is

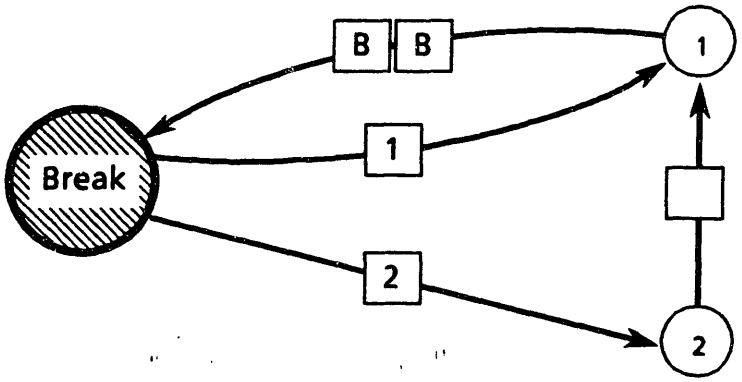

Figure 14b: Shared-arc splicing -- LP optimum trailer flows

and the corresponding slacks s look like

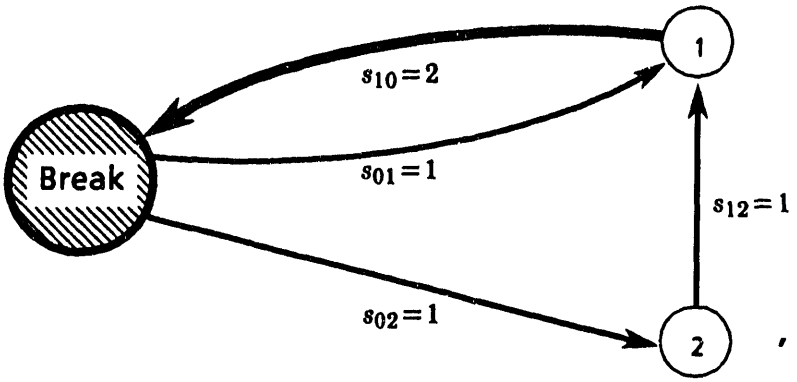

Figure 14c: Slacks for shared-arc splicing example

which can be broken down into the cycles $S=0\text{-}1\text{-}0$ and $R=0\text{-}2\text{-}1\text{-}0$. By splicing the non-shared part of R, 0-2-1, into the non-shared arc $(0, 1)$ of S, we get the IP optimum,

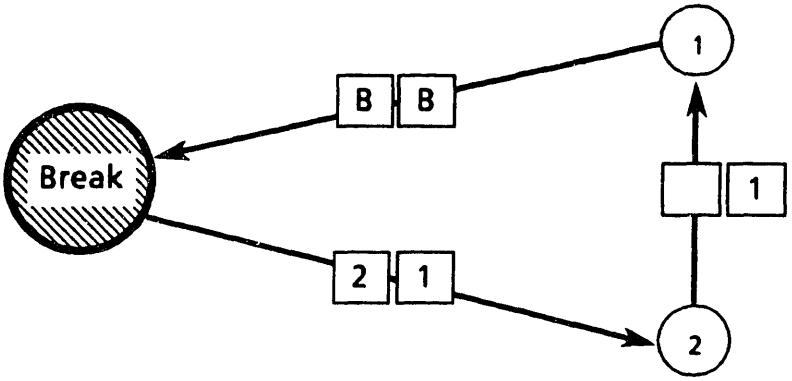

Figure 14d: Outcome of shared-arc splice (also IP optimum)

The exact details of the modifications to steps (a)-(h) necessary to accommodate cases of this type are left to the reader.

In summary, our cycle splicing heuristic works as follows:

(A) Given $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ optimal in $L(\mathbf{u})$, it solves the constrained round up problem (53), and computes the resulting slacks $\mathbf{s}$.

(B) It decomposes $\mathbf{s}$ into a sum of simple cycles.

(C) For each pair of such cycles, it examines all the possible ways of splicing them together. There are a variety of different splicing opportunities because the roles of R and S may be interchanged, and the insertion arc (i,j) may be varied. However, if the cycles are both short, there will only be a handful of possibilities. For each

pair, the heuristic identifies the most attractive splicing opportunity and records it in a list.

(D) It performs the recorded splicing operations in a greedy manner, starting with the one with the greatest savings, and proceeding to the next most profitable one which is still allowable, until the list is exhausted.

One could also imagine using some type of optimization to determine which cycle pairs are best to splice. We will show in a subsequent section that, under certain conditions, the best combination may be chosen by solving a nonbipartite matching problem.

Even without a more intelligent strategy for choosing which cycle pairs to combine, the cycle splicing heuristic is remarkably powerful. Even if it is run just once, with the $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ inputs those arising from the computation of $L(\mathbf{u})$, where $\mathbf{u}$ is proportional to $\mathbf{c}$ (hence $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ are the same as in the LP relaxation optimum without node activity forcing), the results are quite often optimal. Table 1 summarizes its performance on a series of randomly-generated problems. For these problems, EOL terminals were uniformly distributed over the interstices of a 40 by 40 grid centered on the break, with all p_i's and di's drawn from the probability mass function

$$p(x) = \begin{cases} 0.4, & x=1 \\ 0.4, & x=2 \\ 0.1, & x=3 \\ 0.1, & x=4 \end{cases} . \tag{58}$$

This PMF was chosen to resemble the values that might arise in a "metro" operation of United Parcel Service, which makes extensive use of "doubles" [MYE]. Distances were Euclidean, rounded up to integer values, with small adjustments and "stop-off" costs added to insure that the triangle inequality was always *strictly* met. Each row in the table is based on runs of 20 distinct, randomly-generated problem instances.

Note that the apparently poor performance on the 16-node problems may be a reflection of our inability to compute tight lower bounds on the optima of larger problems. It is not necessarily an indication of deteriorating heuristic performance.

Number of EOLs (n)	Percent of time heuristic finds optimum	Upper bound on difference from optimality			
		25th percentile	50th percentile	75th percentile	Maximum
4	60%	0%	0%	3%	11%
8	35%	0%	3%	7.5%	10%
16*	(unknown)	10.5%	18.5%	25%	41%

Table 1: Performance of the cycle splicing heuristic

4.5 Relation of the Heuristics to the Lagrangean

We have now seen two heuristics -- a simple local improvement heuristic and the cycle splicing heuristic. Both of these methods take as input the $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ arising from a computation of $L(\mathbf{u})$. They then attempt to "juggle" some of the routings in such a manner as to reduce the cost of covering these trailer movements with tractors of capacity two. Neither heuristic ever really changes some fundamental aspects of the incoming solution, in particular the assignment of empty trailers. If some EOL terminal a receives a given number of empties from EOL b in the solution to $L(\mathbf{u})$, it will still do so in the perturbed solution output from either heuristic. The only difference will be that certain detours may be inserted in *paths* taken by these trailers. Both heuristics are essentially letting the $L(\mathbf{u})$ computation route the empties.

4.6 Combining the two Heuristics

In practice, the cycle splicing heuristic works very well, but so slowly that it actually slows down the overall run time of the Lagrangean branch and bound method if it is run for every subproblem. We have found that a more efficient approach is to use the cycle splicing heuristic just once, at the very beginning of the run, with $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ inputs as in the LP relaxation (without the node activity constraints). This tactic provides a good initial incumbent. Also, the cycles that arise in the decomposition of $\mathbf{s}$ in these circumstances are always short (see section 6.4 for a proof), allowing for efficient implementation. Subsequently, we use the much faster simple local improvement heuristic at every iteration of every subproblem processed. In this way, we get the advantage of a good initial incumbent, without the computational burden of running the cycle splicing heuristic repetitively.

5. Computational Performance

To test the branch and bound algorithm outlined in the above chapters, we implemented it in VAX FORTRAN. This chapter describes the results.

We decided to run the debugged code on four sets of 20 randomly generated problems, one set with 4 EOLs per instance, one with 8, one with 16, and the last with 32. This last set represents instances of realistic size. The procedure used to generate the random test problems was the same as that described in section 4.4. Runs were performed on a Digital Equipment VAX 11/780 "supermini" computer. The computational power of this machine, at least for compute-bound numerical tasks, is now roughly matched by some of the recently released workstations and "high-end" personal computers.

At varying levels of the percentage fathoming parameter P, we ran each of the four-, eight-, and sixteen-node problem sets. For budgetary reasons, each individual problem run was limited to roughly 5 megabytes of virtual memory and 15 minutes of CPU time. In short, results were excellent for the 4-node problems, acceptable for 8 nodes, and disappointing for the 16-node examples. Because of the poor performance on the 16-node problem set, we did not attempt to run the 32-node data.

Table 2 summarizes the results for selected combinations of numbers of EOLs and levels of P. It should be noted that runs were simply cut off if the available time or memory limits had to be exceeded. Our present method is quite memory-intensive, as basis information is retained from subproblem to subproblem. On the 8- and 16-node problems, the algorithm would often have to switch,upon reaching about 300 active subproblems, to the high-β_p mode in which it attempted to fathom unpromising subproblems first in an effort to free up space. This procedure tended to slow down convergence, and was sometimes unsuccessful in holding down memory requirements (when high-β_p subproblems could not be fathomed), in which case the program simply halted upon reaching its 5 megabyte storage limit.

These results are not altogether encouraging. Note that with $P=5\%$, run time increases by a factor of 30 from the 4- to the 8-node problems, an increase of only a factor of 2 in problem size. Naively extrapolating, this growth rate would imply that a 32-node "real life" problem would have an expected run time of over 40 hours, with a median of about 8. Speaking very roughly, the method is about one

Number of EOLS	Fathoming Percentage P	Run Time (sec)		Duality Gap	
		Mean	Median	Mean	Median
4	0 %	16.7	4.3	0.1%	0.0%
4	5 %	5.6	0.9	5.0%	5.0%
8	0 %	439.0	348.9	1.6%	0.0%
8	5 %	166.7	32.9	5.2%	5.0%
16	5 %	900 +	900 +	15.8%	16.5%

Table 2: Performance of Lagrangean branch and bound method

or two orders of magnitude short, either in terms of speed or size of problem solvable, of being practical.

In particular, the 16-node runs produce a duality gap only marginally smaller than that one would get by simply running the cycle splicing heuristic and the LP relaxation, and then comparing the two. None of the twenty 16-node cases run showed any improvement in the incumbent after the first iteration of the first subproblem, and the improvement in the lower bound attributable to enumeration was not very great. This suggests that the best course at present is to use the cycle splicing heuristic as a stand-alone procedure, and optionally employ the rest of the algorithm as a means of assessing nearness to optimality.

In the smaller problems that we were able to analyze in detail, there seemed to be a strong dependence of run time on the initial value of $\mathbf{u}$. On average, the best results were obtained with $\mathbf{u} = \mathbf{c}/4$. However, for individual problems, different values of the form $\mathbf{u} = (\lambda/2)\mathbf{c}$, where $0 < \lambda \leq 1$, would often work much better. The dependency of λ seems to be present even in problems in which much enumeration is needed to verify the solution. If we could understand this phenomenon, we feel that we might be able to take considerable advantage of it.

In summary, the method is quite a long way from being useful, even for the comparatively simple 2-CFP case. Certainly, the current results do not merit an attempt to extend the methodology to problems more complicated than the central 2-CFP.

Perhaps the poor performance may be partially attributed to our model's use of general nonnegative integer variables, as opposed to simpler 0-1 variables. As far

as we know, successful applications of Lagrangean branch and bound methods have all used binary variables. When all one's integer variables are binary, the general features of the solution are in some sense much more firmly "locked in" by each separation operation in the enumeration tree. In principle, our current model could be converted to an equivalent problem using 0-1 variables only, given a reasonable upper bound on each original variable, but we suspect this would significantly "destructure" the formulation.

As for the problem of trying to construct a practical method for 2-CFP's of realistic size, there are two possible avenues for future work. On the one hand, one can turn, as vehicle routing practitioners usually have, to promising heuristics. Alternately, one could try to further refine an exact method like that presented here, hoping for another one or two orders of magnitude improvement in performance. Here, the outlook is a little doubtful, but considering that over our one year project we achieved something like two orders of magnitude improvement over our initial working algorithm, not entirely bleak.

In the remainder of this thesis, we first discuss the refinement option, and then turn to heuristics and other fresh approaches.

6. Potential Improvements to the Current Method

The rest of this thesis discusses approaches to the central 2-CFP that might prove practically viable. This chapter concerns itself with possible improvements to the method just described. We discuss four separate topics: different ways to update $\mathbf{u}$, tightening the formulation, selective arc removal, and using matching in the incumbent finders.

6.1 Improving the Dual Step

One possible improvement to the LBB method would involve inserting new kinds of dual update procedures, in addition to the current subgradient steps. In particular, we envision steps in which all the dual variables are simultaneously scaled by some real number λ. We have already mentioned that for some small problems, altering the initial value of $\mathbf{u}$ in this manner could produce dramatically better results. If one understood more fully the mechanism behind this phenomenon, one might be able to create a much faster algorithm. One could either scale only the initial $\mathbf{u}$, which is proportional to $\mathbf{c}$ in our present approach, or one might also be able to scale at later times, when $\mathbf{u}$ is not proportional to $\mathbf{c}$ and/or there are branching constraints on $\mathbf{t}$.

This idea relates to a curious property of the subgradient method: if one is not careful, the search for the right value of $\mathbf{u}$ may be artificially restrictive. For example, the canonical subgradient

$$\mathbf{g} \doteq \mathbf{x}+\mathbf{y}+\mathbf{e}-k\mathbf{t} \tag{59}$$

that we have been using can easily be seen to have the property $\mathbf{Ag}=\mathbf{0}$ at all times. If our steps are always exactly in the direction given by $\mathbf{g}$, then $\mathbf{u}$ will always be in the affine space $\mathbf{u}_0+\mathrm{N}(\mathbf{A})$, where $\mathbf{u}_0$ is the initial choice of $\mathbf{u}$, and $\mathrm{N}(\mathbf{A})$ denotes the null space of $\mathbf{A}$. Of course, our steps are not always precisely in the $\mathbf{g}$-direction, because we always project $\mathbf{u}$ back onto the region $\{\mathbf{u} \mid \mathbf{0}\le\mathbf{u}\le\mathbf{c}/2\}$. Still, using only one kind of subgradient step could be artificially depressing our lower bonds. A related topic worth investigating is whether or not restricting to $\{\mathbf{u} \mid \mathbf{0}\le\mathbf{u}\le\mathbf{c}/2\}$, rather than the more complex region (37), might also be hurting the lower bounds.

Perhaps our uncertainty in this area reflect the deeper problem that we do not as yet have a good feeling for how the LP relaxation for our model behaves in the presence of the node activity forcing constraints described in section 3.2. While we

have an excellent understanding of the LP relaxation *without* these constraints, we do not have a good idea how the optimal LP multipliers change when they are included. This is probably worth some further study, possibly by solving some instances using a "commercial" LP package. A better understanding of the augmented LP relaxation might help one find a way to choose better initial multiplier values, or even to construct a superior dual update procedure. It might also reveal additional cutting planes that could serve to tighten the relaxation even further.

6.2 Tightening the Formulation

The mention of cutting planes brings to mind another approach, which would be to try to tighten the LP relaxation, and thus the Lagrangean lower bounds, by using a different problem formulation. Here it is useful to refer to Magnanti's vehicle routing survey [MAG], dividing VRP formulation into aggregate and disaggregate commodity flow, vehicle flow, and set covering categories. We will discuss set covering in a later section. Among the other formulations, ours resembles the most "aggregated", aggregate commodity flow. For many combinatorial problems, researchers have recently observed that decomposition methods seem to generally perform better when they are based on more "disaggregate" formulations, largely because the LP relaxations of such formulations produce bounds that are closer to the IP solution. Thus, one might expect that we could benefit by disaggregating our current formulation.

Moving to the *dis*aggregate commodity flow category of formulations would essentially be the same as treating central k-CFP problems as if they were non-central, that is, labeling each loaded trailer by its destination, even though it is not strictly necessary to distinguish between outbound trailers. Unfortunately, as we saw in section 2.4, this operation does not tighten the LP relaxation at all. Thus, to make any progress in this direction, one would have to go immediately to an even *more* disaggregated description, perhaps analogous to a *vehicle* flow formulation of the VRP. It remains unclear, at this stage, whether or not such a change would be beneficial.

6.3 Selective Arc Removal

There is a common trick that we have not yet tried: solving the problem on a restricted network containing only arcs selected by some heuristic as likely to be in

the solution, and then checking to see if any previously excluded arcs should subsequently be included. We do not know how this type of approach would best be applied for CFP-class problems, and it would doubtless require clever data structures. Still, it is definitely worth some study.

6.4 Matching in the Incumbent Finders

Another fruitful avenue might be to use a nonbipartite matching algorithm, rather than the greedy heuristic, to choose which improvements to apply in either the simple local improvement or cycle splicing incumbent generators. This holds the potential of providing better incumbents, but at some price in speed, depending on how often matching is attempted. In the small problems whose runs we have been able to analyze fully, our experience indicates that slow run times are mostly attributable to a slow climb in the lower bounds, rather than poor incumbents; however, this may not be the case in larger instances, and matching may prove very worthwhile.

Let us now turn our attention to the cycle splicing heuristic. Suppose we have decomposed s into a sum of simple cycles. Construct a *derived graph* based on this decomposition, with a node corresponding to each cycle, and a complete set of edges. Because each individual cycle splice "consumes" the slack on the two cycles it processes, and removes slack only from those two cycles, one can conclude that any choice of a set of possible pairs of cycles to splice corresponds to a matching in the derived graph, and *vice versa*. Assign to every edge $<R, S>$ in the graph a weight $w(R, S)$ equal to the savings from the most profitable way of splicing cycles R and S. Then a maximum weight matching will yield the best possible set of splicings, *given* the original decomposition of s into cycles.

At this point one naturally asks whether there is more than one way to decompose s into simple cycles, and, if so, whether the choice of decomposition would have any bearing on the outcome of the combined matching and splicing heuristic. These are both to some extent open questions, but we *can* give some conditions assuring a unique decomposition. To do this, we must make a short theoretical digression.

6.4.1. Definition. Given a network with node-arc incidence matrix $\mathbf{A}$, having some distinguished central node numbered 0 ("the break"), and a balanced flow vector $\mathbf{w}$ (with $\mathbf{Aw}=\mathbf{0}$, $\mathbf{w}\geq\mathbf{0}$), we say $\mathbf{w}$ has the *fanning property* if for every node $i\neq 0$,

either all outflow from i is to the break ($w_{ij}=0\ \forall j\neq 0$) *or* all inflow to i is from the break ($w_{ji}=0\ \forall j\neq 0$).

The fanning property seems very restrictive, but in fact it is met in a very important case by the solution to $L(\mathbf{u})$:

6.4.2. Lemma. If $\mathbf{u}$ obeys the strict triangle inequality, $\mathbf{A}$ is complete,and there are no additional branching constraints on $\mathbf{x}$, $\mathbf{y}$, or $\mathbf{e}$, then $\mathbf{r}\doteq\mathbf{x}+\mathbf{y}+\mathbf{e}$ has the fanning property, where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{e}$ are optimal in $L(\mathbf{u})$.

Proof: All trailers take direct arcs from origin to destination. The only kinds of trailers that travel on arcs not touching the break are empties, and they must necessarily take direct arcs from terminals that are sources of empties ($p_i<d_i$) to ones that are sinks for empties ($p_i>d_i$).

To prove fanning, we need to show that for all nodes $i,j,l\neq 0$, either $r_{ij}=0$ or $r_{jl}=0$. Suppose that $r_{ij}>0$. Then j must be a sink for empties, and consequently $r_{jl}=0$. **QED**.

6.4.3. Corollary. When $\mathbf{u}$ obeys the strict triangle inequality, the $\mathbf{t}_S$ and $\mathbf{s}$ vectors constructed by the cycle splicing heuristic have the fanning property.

Proof: By construction, $\mathbf{t}_S$ may be nonzero only on arcs where $\mathbf{r}$ is nonzero. Therefore, it inherits the fanning property. Finally $\mathbf{s}=2\mathbf{t}_S-\mathbf{r}$ is also zero on all arcs where $\mathbf{t}_S$ and $\mathbf{r}$ are zero, so it too is a fanning flow. **QED**.

Provided $\mathbf{c}$ is strictly triangular, as it was in our test problems, the hypotheses of the lemma are met in the one situation where we now use the cycle splicing heuristic: at the very beginning of the run, when there are no branching constraints and $\mathbf{u}=(\lambda/2)\mathbf{c}$. If $\mathbf{u}$ is triangular, but not strictly so, we can only conclude that there is *an* optimal solution to $L(\mathbf{u})$ that yields $\mathbf{r}$, $\mathbf{t}_S$,and $\mathbf{s}$ having the fanning property.

Now comes our main result.

6.4.4. Proposition. Any flow $\mathbf{w}$ having the fanning property has a *unique* decomposition into simple cycles, and these cycles all have length 2 or 3 and include the break.

Proof: Consider any $i,j\neq 0$ for which $w_{ij}>0$. In any decomposition of $\mathbf{w}$ into simple cycles, there must be some cycle R with positive flow on (i,j). The fanning property gives that the arc in R following (i,j) must by $(j,0)$, and the arc before (i,j) must be $(0,i)$. To be simple, R must be equal to 0-i-j-0 (hence it has length 3 and

includes the break). Subtract one unit of flow from $\mathbf{w}$ for every arc in R. The resulting flow $\mathbf{w}'$ still has $\mathbf{Aw}' = \mathbf{0}$, $\mathbf{w}' \geq \mathbf{0}$, and is still fanning, and the remaining cycles in the decomposition of $\mathbf{w}$ form a decomposition of $\mathbf{w}'$.

We can repeatedly use operations like that above to deduce an entire sequence of cycles that must be in any decomposition of $\mathbf{w}$, and finally reduce to the case that $\mathbf{w}$ only has flow on arcs touching the break. A similar argument shows that such a flow is a unique sum of 2-cycles that include the break. **QED**.

This proposition implies that if $\mathbf{u}$ is strictly triangular, then there is only one way to decompose $\mathbf{s}$ into simple cycles, and thus that a nonbipartite matching will give the best possible collection of cycle splices that can be performed on $\mathbf{s}$.

Note that we are not saying that one should not use matching to choose the best splices when $\mathbf{u}$ is not strictly triangular, but only that we have not ruled out the possibility that the choice of how to decompose $\mathbf{s}$ into simple cycles could affect the outcome.

We have not studied the simple local improvement heuristic in similar depth, but we suspect that a similar result may be obtained for it.

7. Alternate Solution Strategies

We now describe some alternate solution approaches to the central 2-CFP that cannot be viewed as enhancements to the algorithm developed and tested in chapters 3 through 5.

The first of these is a *trip matching* heuristic inspired by the idea of combined matching and cycle splicing that we have just examined, but conceptually simplified. This alternate approach has the advantage of being relatively accommodating to side constraints. We will also briefly examine a few other fresh approaches, namely resource-directed decomposition, set covering, and LP-based branch and bound.

7.1 The Trip Matching Heuristic

The *trip matching* heuristic is an approach to solving the central 2-CFP that draws on the ideas first encountered in the context of cycle splicing. We have not been able to test it computationally, but believe that it is nevertheless worth describing.

First, imagine solving the LP relaxation of the aggregate commodity flow formulation (9), without node activity constraints. As we already know, this amounts to solving a transportation problem among the empty trailers. Making sure we choose the optimal solution in which all trailers take direct arcs from origin to destination (there could be some non-uniqueness if the triangle inequality is met with equality for some node triplets), we form the sum of trailer flows,

$$\mathbf{r} \doteq \mathbf{x}+\mathbf{y}+\mathbf{e} \quad . \tag{60}$$

Note that $\mathbf{r}$ has the fanning property described in the previous chapter. It can therefore be uniquely decomposed into a sum of two and three cycles, all including the break. We are now decomposing $\mathbf{r}$, not $\mathbf{s}$, and so there is a possibility that some cycles may be repeated in the decomposition; the $R_1,\ldots,R_J$ constituting the decomposition may not all be distinct, but, up to their order, they will still be unique. We start by hypothetically assigning one tractor to each of these cycles, or "trips". This step yields a solution that would be optimal were there only one trailer allowed per tractor.

Now consider any pair of such cycles (possibly equal if there are repetitions in $R_1,\ldots,R_J$). As in our cycle splicing heuristic, there will be one or more possible ways of combining them into a trip executed by a single tractor, hauling two trailers at a

time, that effects the same end-result trailer movements. For example, one might make a transformation such as

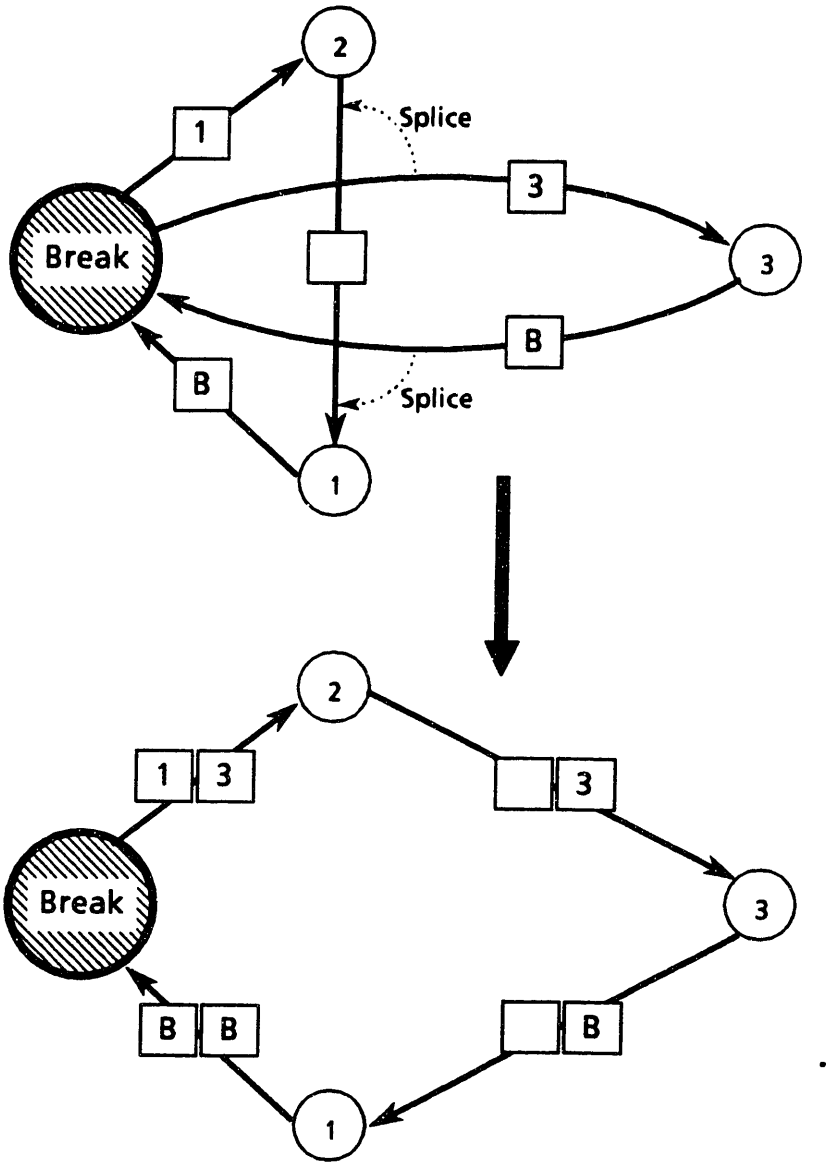

Figure 15: Splicing two trips

For every pair of trips, there are a number of ways to perform such a splice, depending on which cycle is inserted into the other, and where the insertion takes place. As both trips have length at most 3, the number of possible combinations is limited, the one with the highest savings can easily be found by "brute force". Our proposed heuristic works as follows:

Trip Matching Heuristic (TMH)

(i) Solve the transportation problem for the empty trailers, thus solving the LP relaxation of the aggregate flow formulation (9).

(ii) Construct the sequence of trips of length 2 and 3 that constitutes the unique decomposition of the resulting total trailer flow r.

(iii) Examine each pair (R, S) of such trips and find the maximum savings way of splicing them together. Denote the maximum savings by $w(R, S)$.

(iv) Construct a complete graph with a node corresponding to each trip, and find its maximum nonbipartite matching using weights $w(R, S)$.

(v) Construct the integer solution corresponding to the splicing operations indicated by this matching.

This method is conceptually simpler than the slack cycle splicing method we have tested, and should give much the same results. It is elegant enough that one might be tempted to think it might actually be an exact method. Unfortunately, this is false.

7.1.1 Trip Matching is not an Exact Procedure

Trip matching is an inexact method for the general case of the 2-CFP for the reason that it, like the simple local improvement or cycle splicing heuristics as applied to LP relaxation input data, essentially allows the LP to decide the disposition of empty trailers, and never really changes it except to add a detour of a node or two in some routes. It is quite easy to construct examples for which the assignment of empties is *different* in the IP and LP optima, and which therefore cannot be optimized by the trip matching heuristic as we have stated it. One example is

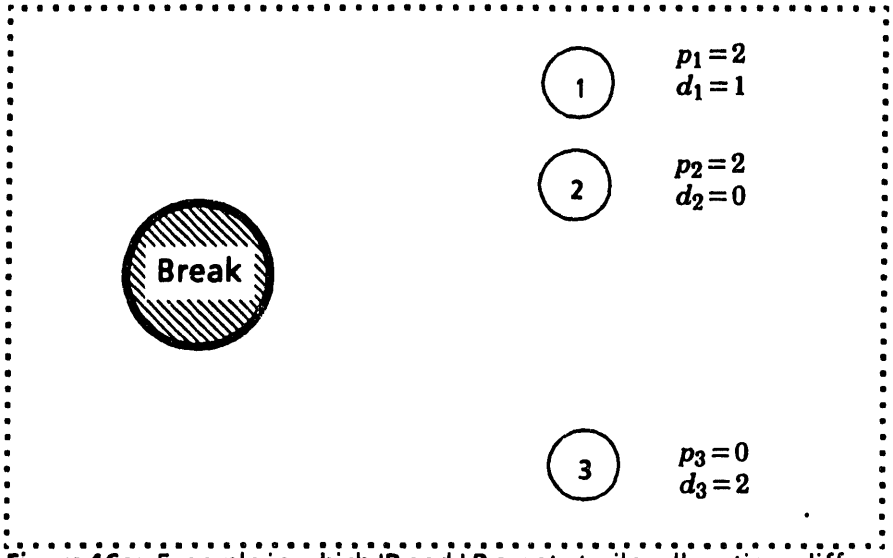

Figure 16a: Example in which IP and LP empty trailer allocations differ

The LP relaxation solution, without node activity constraints, is

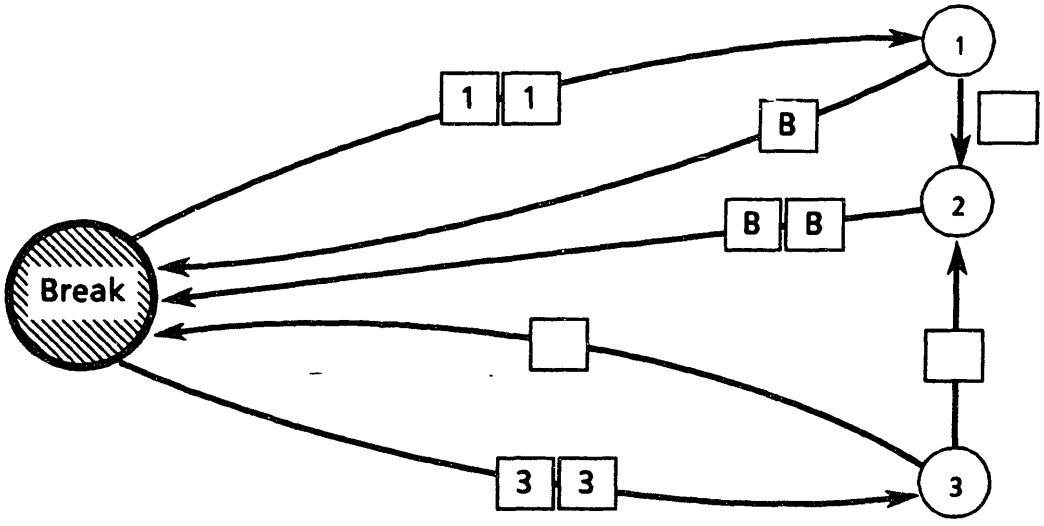

Figure 16b: LP relaxation optimum for figure 16a data

while the integer optimum is

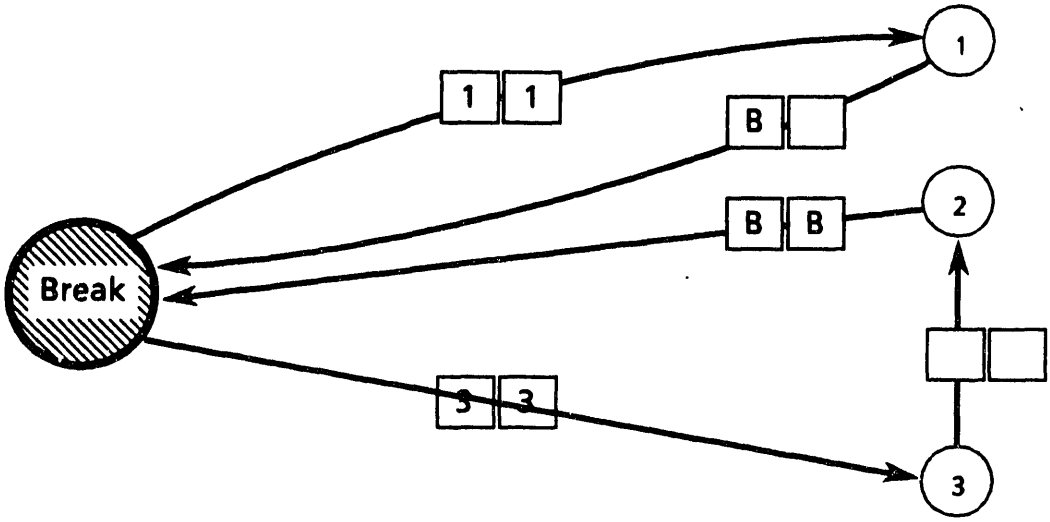

Figure 16c: Integer optimum for figure 16a data

In fact, even if the specified empty movements are correct, there are cases in which trip matching may not produce the best possible answer. We call this phenomenon "trip hopping". Consider the problem instance

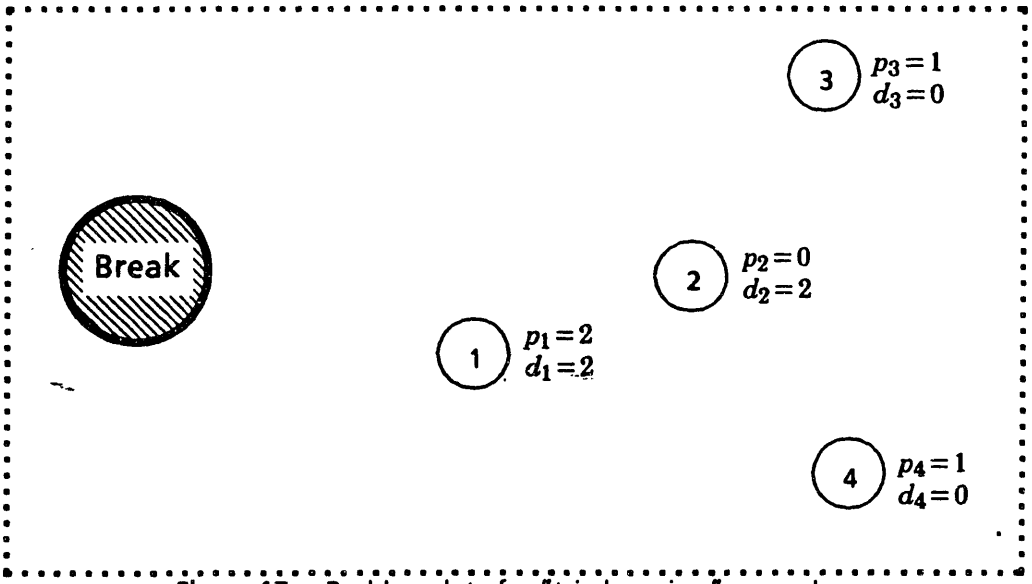

Figure 17a: Problem data for "trip hopping" example

This example has the integer optimum

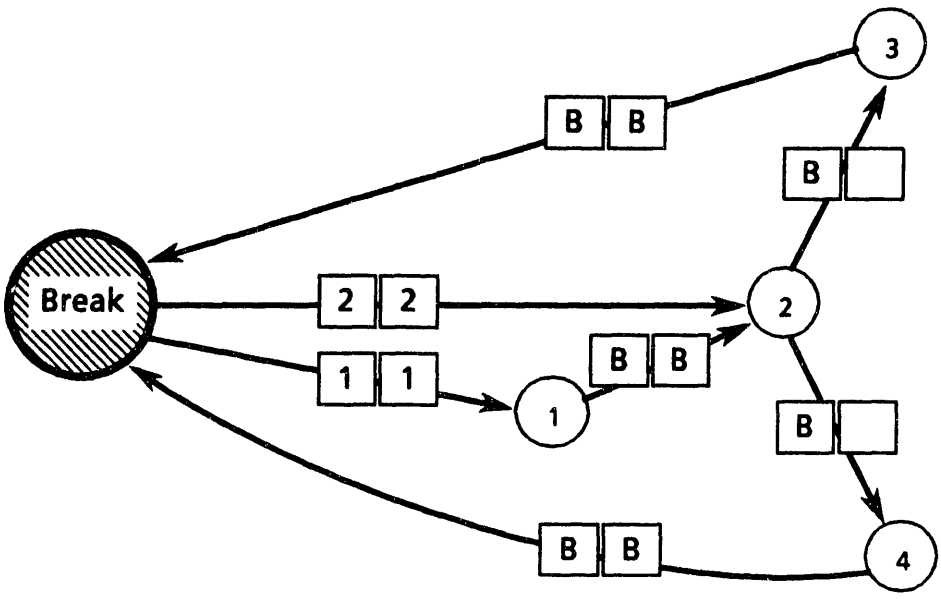

Figure 17b: Integer optimum for "trip hopping" example

In this curious example, an inbound trailer must be switched from one tractor to another at node 2, "hopping" between tractor trips. Such a solution cannot be generated by the trip matching heuristic. On the other hand, such solutions may be impractical, and one might be able to argue that they should be prohibited.

The run time of the trip matching heuristic, *after* the execution of the transportation simplex method, depends not on the number of EOL terminals n, but on the number of single-trailer trips implied by the LP solution, which we shall denote by N. This is equal to the total number of trailers to be dispatched from the break in one day, $\mathrm{Max}(|p_0|, |d_0|)$. Computing the maximum savings from all pairs of trips requires $O(N^2)$ time, and the matching $O(N^3)$ time, so total run time is $O(N^3)$, plus the time to solve the empty allocation transportation problem. In real problems, N will be on the order of hundreds, so run time may be substantial, but not an insurmountable obstacle. At worst, one might have to resort to a heuristic matching procedure for very large instances.

7.1.2 Handling Side Constraints

Despite the drawbacks just outlined, the trip matching heuristic has some very attractive properties. For one thing, routes of individual tractors are explicitly constructed by the algorithm. It needs no "post-processing" to decompose the overall tractor flow into individual trips. More importantly, the heuristic could easily be extended to handle a wide range of side constraints. For example, when examining all the possible ways of merging a particular pair of trips, the heuristic

could rule out those that would produce tours that would be too long, would take too much time, would have too many stops, or might violate some other secondary restriction. If there turned out to be no feasible way of combining a given pair, the corresponding edge would be assigned a weight of zero in the derived graph.

In the trip matching context, there may also turn out to be ways to enforce certain kinds of timing constraints without making the costly switch to a time-space network model. These might also involve placing upper bounds on some arcs in the empty allocation network. Finally, there is a potential to handle more complicated cost structures that are not simple linear functions of the tractor flows.

7.1.3 Further Investigation

The potential capabilities outlined above make the trip matching heuristic appear quite promising for practical application. We suggest that the following represent a good agenda for future research in this area:

(i) Examine the method's performance empirically.

(ii) Determine the conditions and restrictions (*e.g.* absence of trip hopping, correct empty disposition) under which the matching approach will give optimal answers.

(iii) Armed with this knowledge, attempt to augment the heuristic to better handle situations in which it is not optimal. Perhaps one could construct a master-subproblem scheme in which the master problem would try different routings for the empties, while the subproblem would use matching to try and best "cover" the implied movements.

(iv) Try a "real-life" pilot application with an actual carrier, incorporating that carrier's particular side constraints.

An additional topic might be to study generalizations to the non-central or $k>2$ cases. The latter may prove very difficult, because 3-matching and higher order partitioning problems are themselves NP-complete.

7.2 Resource-Directed Decomposition

In his treatment of the related railroad blocking problem, Assad [AS] suggests a resource-directed approach in which the master problem fixes the number of locomotives (in our setting, tractors) on each arc, and the subproblem must optimally route all freight, given this allocation. At first glance, this seems a very difficult approach, as the resulting subproblem, *integer* capacitated multicommodity flow, is itself extremely hard. In a price-directed scheme, one at least gets integrality in the subproblems "for free".

We are not sure if there is any way to make the resource directed approach attractive. We mention it here mainly for completeness.

7.3 Set Covering

A more promising area that we have not yet explored is to use an approach modeled after the "set covering" methods for the VRP (see [BQ]). In such a line of attack, one would enumerate a very large number J of possible tours that a tractor could possible make. By a *tour* we mean not only the path taken by the tractor, but also the particular swapping operations it performs at each terminal stop.

The k-CFP could then be formulated as the integer program:

$$
\begin{aligned}
\text{Min} \quad & \sum_{j=1}^{J} c_j z_j & & & \\
\text{S.T.} \quad & \sum_{j=1}^{J} a_{ij} z_j & = & d_i & i=1,\ldots,n \\
& \sum_{j=1}^{J} b_{ij} z_j & = & p_i & i=1,\ldots,n \\
& z_j & \geq & 0 & j=1,\ldots,J \\
& z_j & & \text{integer} & j=1,\ldots,J
\end{aligned}
\tag{61}
$$

Here a_{ij} is the number of outbound trailers route j drops at terminal i, and b_{ij} the number of inbound trailers it picks up. The decision variable z_j gives the number of "type j" tours in the solution. We suspect that one can prove that any optimal solution may be constructed exclusively out of *locally balanced* tours, that is, tours

that preserve the number of trailers at each terminal stop, always picking up as many trailers as they drop off. If the set of tours is so restricted, one need not add any explicit empty balancing constraints to (61).

Furthermore, it should also be possible to prove that only few of the z_j's, namely those representing simple "out and back" tours involving the break and just one EOL, need ever take a value greater than one. Thus, one can constrain most of the decision variables in (61) to be binary. However, because there are still variables, constraint matrix elements, and right-hand side entries that can be greater than 1, the remaining integer program is different from the set covering problems that arise from the "traditional" VRP, and its properties must be investigated.

The LP relaxation of (61) could be solved by column generation procedure in which the column generator could be a dynamic program with a manageably sized state space. However, success rests on solving the *integer* program by exploiting special structure. Here, one might be able to draw on earlier work done for the VRP. Though an exact method may prove out of reach, the set covering reformulation may prove useful in constructing a good heuristic amenable to side constraints, and possibly form the basis for a method based on interactive optimization similar to that in [CTR] or [SP].

7.4 LP-Based Methods

Another plan of attack we have not yet tried is an approach based on an LP relaxation, rather than a Lagrangean one. In particular, we have not yet attempted to solve CFP problems using a "commercial" quality integer programming code. If one used a restricted network rather than a complete one, and augmented such a code with some special routines, such as incumbent generators, similar to those already developed for our Lagrangean method, one might conceivably be able to construct a practical method for realistic-sized problems. This type of approach certainly merits some investigation, even though the prospects are unclear. Its chances for success rest on exact LP lower bounds on the objective being significantly sharper than their approximations via the Lagrangean. If one could show this will *not* be the case, then the method would warrant no further investigation.

8. Conclusion

Although we have identified some possible improvements and alternative approaches to the Lagrangean branch and bound method, our computational results remain somewhat disappointing. This may reflect the unfortunate reality that Lagrangean methods are not a panacea for integer programming problems. For a Lagrangean method to succeed, a happy synchrony must exit between all its components. That is, the multiplier updating procedure, the incumbent generator, and the enumeration scheme must all work together to both raise the overall lower bound on the optimum solution, and bring down the upper bound represented by the incumbent. In our case, the computational burden seems to fall excessively on the enumeration and incumbent-finding components; the subgradient step seems unable to provide rapid gains in the lower bound or to quickly point the the incumbent finder in the right direction.

At present, it is not well understood what conditions are conducive to establishing the subtle and fortunate state of affairs in which all the parts of a Lagrangean relaxation method cooperate smoothly. All we know is that the convexified problem (in our case, the LP relaxation) solved by the Lagrangean should provide a good lower bound on the integer objective. This is in itself no guarantee of success, so unless one can demonstrate that these bounds will *not* be good and hence that the method is unlikely to work, one must simply test empirically. It may happen, as in the research described here, that a great deal of "custom" programming effort may be expended to arrive an unspectacular result. In the future, new analytical tools may perhaps make it easier to assess the chances for success, or there may be programming environments that will make it much cheaper to test prospective methods.

Fortunately, we have a little more to show for our pains than a mixed computational experience. We have developed what is apparently a quite effective heuristic method (cycle splicing), identified possible improvements to it, and proposed another, similar approach more receptive to side constraints (trip matching). Also, we should not disregard the possibility of refining the current approach into a workable state, or of constructing a successful set covering or LP-based method armed with the experience gained here.

Last but not least, we have acquired the beginnings of an understanding of the CFP class of problems that we have defined. When one takes the viewpoint of a carrier

both picking up and delivering freight containers, as well as balancing empties, rather than that of a "vertically integrated" concern that moves freight primarily in one direction, vehicle routing problems take on a decidedly different flavor. The concepts of slack flow, the fanning property, simple cycle decomposition, and local improvement via splicing may prove useful in further analyzing CFP problems.

References

[AS] A. A. Assad, "Modeling Rail Freight Management", Ph D. Thesis, Sloan School of Management, MIT, 1978

[BELL] W. J. Bell *et. al.*, "Improving the Distribution of Industrial Gases with an On-line Computerized Routing and Scheduling Optimizer", *Interfaces* 13:6, pp. 4-23 (1983)

[BOD] L. Bodin *et. al.*, "Routing and Scheduling of Vehicles and Crews: The State of the Art", *Computers and Operations Research* 10, pp. 63-211 (1983)

[BQ] M. L. Balinski and R. E. Quandt, "On an Integer Program for a Delivery Problem", *Operations Research* 12, pp. 300-304 (1964)

[CTR] F. H. Cullen, J. J. Travis, and H. D. Ratliff, "Set Partitioning Heuristics for Interactive Optimization", *Networks* 11, pp. 125-143 (1981)

[FISH1] M. L. Fisher, "The Lagrangian Relaxation for Solving Integer Programming Problems", *Management Science* 27, pp. 1-18 (1981)

[FISH2] M. L. Fisher, "An Applications Oriented Guide to Lagrangian Relaxation", *Interfaces* 15:2, pp. 10-21 (1985)

[FISH3] M. L. Fisher *et. al.*, "Real-Time Scheduling of a Bulk Delivery Fleet: Practical Application of the Lagrangian Relaxation", Decision Sciences Working Paper 82-10-11, The Wharton School, University of Pennsylvania (1982)

[GG] B. Gavish and S. C. Graves, "The Traveling Salesman and Related Problems", Working Paper OR 078-78, Operations Research Center, MIT (1978)

[GM] B. L. Golden and T. L. Magnanti, Sloan School of Management Course Notes, Course 15.082, MIT (1985)

[MAG] T. L. Magnanti, "Combinatorial Optimization and Vehicle Fleet Planning: Perspectives and Prospects", *Networks* 11, pp. 179-213 (1981)

[MW] T. L. Magnanti and R. T. Wong, "Accelerating Benders Decomposition: Algorithmic Enhancements and Model Selection Criteria", *Operations Research* 29, pp. 464-484 (1981)

[MYE] T. Myers, United Parcel Service, Paramus, N.J., Personal Communication

[SHAP] J. Shapiro, *Mathematical Programming: Structures and Algorithms*, John Wiley (1979, New York)

[SP] Y. Sheffi and W. Powell, "Interactive Optimization for LTL Network Design", Center for Transportation Studies Report 85-17, MIT (1985)