Package Routing in Tranportation Networks with fixed Vehicle Schedules: Formulation, ComplexityResults and Approximation Algorithms

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-26
June 1992

Package Routing in Transportation Networks with Fixed Vehicle Schedules: Formulation, Complexity Results and Approximation Algorithms

Lloyd Greenwald* Thomas Dean*
Department of Computer Science
Brown University, Box 1910, Providence, RI 02912

June 29, 1992

Abstract

We consider a special case of a general problem involving the deployment of vehicles to transport packages in transportation networks. In this special case, the schedules of the vehicles are fixed and packages are routed by transferring them between vehicles as these vehicles make stops according to their fixed schedules. We show that this problem is hard and explore approximation algorithms for its solution. In particular, we cast this problem as a multicommodity flow problem with a mixed integer/linear program formulation. We then apply combinatorial optimization techniques based on solving the relaxed linear programming formulation of the problem to obtain provable feasibility and expected performance guarantees, where performance is measured in terms of the sum of the time in transit over all packages. We investigate the sensitivity of the performance guarantees to certain scaling factors and other limitations of this technique.

---

*This work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601 and by the Air Force and the Advanced Research Projects Agency of the Department of Defense under Contract No. F30602-91-C-0041.

1 Introduction

The general transportation problem addresses a wide range of issues facing those in government and industry responsible for the efficient and timely transportation of passengers and cargo. In this paper we explore in detail a special case of the general problem in which the schedules of the vehicles are fixed and packages are routed by transferring them between vehicles as these vehicles make stops according to their fixed schedules. We call this restricted problem the *greyhound package scheduling problem* since its fixed vehicle schedules are reminiscent of commercial bus schedules. By focusing on this restricted version of the general problem we gain insight into the general problem as well.

In addition to providing insight into the general problem, there is independent motivation for studying this restricted version of the problem. One heuristic for solving scheduling problems is to decompose the larger problem into a series of smaller scheduling problems. One such multistage scheduling approach is to solve the scheduling problem for the larger cargo and then solve the scheduling problem for the smaller cargo within the constraints imposed by the solution to the former problem. The first stage of this two stage process can be solved by algorithms such as those designed for the dial-a-ride problem [4, 10]. The solution resulting from this first process is comprised of vehicles with fixed schedules and possibly unused capacity. These are the primary features of the greyhound problem, therefore, the techniques described in this paper for solving the greyhound problem can be employed to solve the second stage of this multistage scheduling approach.

In Section 2, we briefly outline a static version of the general transportation problem. The static version of the problem considers the basic entities of the transportation network and the static scheduling problem and ignores issues such as online processing, uncertainty, decision-making structure and distributed processing. In Section 3, the greyhound problem is described and formulated as a restriction of the static transportation problem. The first result of our exploration into the greyhound problem is presented in Section 4. In this section, we prove that the greyhound problem is NP-complete by reducing 3-Satisfiability to it. This result extends to the static transportation problem and the general transportation problem and also provides motivation for exploring approximation algorithms to solve the greyhound problem. The approximation algorithms we consider are variants of the randomized rounding technique of Raghavan and Thompson [9, 8]. Since this technique has been applied to multicommodity flow problems, in Section 5 we provide a polynomial reduction from the greyhound problem to the multicommodity flow problem. By doing so we can more easily apply the randomized rounding techniques to the greyhound problem. In Section 6, we describe in detail randomized rounding and its application to the greyhound problem. Additionally, we explore the provable feasibility and expected performance guarantees that this technique provides. The initial result shows that the technique can be used to provide a solution to the greyhound problem in which the package load assigned to any vehicle does not exceed its capacity by more than a constant factor with high-probability. However, this analysis technique cannot be used to provide constant factor bounds for performance; though it does provide expected performance that is optimal. In Section 7, this technique is extended to provide deterministic algorithms that guarantee feasible solutions (no capacities exceeded) to the greyhound problem, if they exist. The basic technique and its variants have inherent limitations. These limitations are explored in the context of the greyhound problem.

2 Static Transportation Problem

The transportation problems considered in this paper involve the deployment of diverse equipment (*e.g.,* planes, ships, trucks) to transport cargo (*e.g.,* personnel and material) throughout large geographical areas for private, government, and military purposes. A full presentation of the general transportation problem, including both static and dynamic entities, is given in [4]. In this section, we present a specialized version of the static transportation problem abstracted from the more general presentation.

The static portion of the transportation problem concerns the spatial layout of locations, capabilities of transportation assets and requirements for shipping packages. The notions concerning the static structure of the transportation network are depicted graphically in Figure 1 and formalized in the following mathematical objects.

- a set of locations, L; each location $l \in L$ has a storage capacity, $cap_l \in \mathbf{R}$;
- a set of direct routes (channels)[1], R, between locations in L; each direct route $r \in R$ has a length $len_r \in \mathbf{R}$;

---

[1] Transportation problems are described in the literature using a variety of language. To standardize terminology we provide our default terms for basic entities and, parenthetically, alternative terms used for the same entities elsewhere in the literature.

- (L, R) corresponds to a directed graph with vertices, L, and edges, R;
- a set of packages (parcels, commodities), P; each package $p \in P$ has:
    - requirement (size, demand) $req_p \in \mathbf{R}$,
    - source (initial location) $src_p \in L$,
    - sink (destination) $snk_p \in L$,
    - release date (earliest pickup time) $rel_p \in \mathbf{T}$ ($\mathbf{T}$ is the set of all time points), and
    - optional deadline (latest delivery time) $dead_p \in \mathbf{T}$;
- a set of transportation assets (vehicles), A; each asset $a \in A$ has:
    - capacity $cap_a \in \mathbf{R}$,
    - speed $spd_a \in \mathbf{R}$,
    - range $rng_a \in \mathbf{R}$,
    - source (starting location, depot) $src_a \in L$,
    - optional sink (ending location, end depot) $snk_a \in L$,
    - release date (earliest movement from depot) $rel_a \in \mathbf{T}$, and
    - optional deadline (latest arrival at end depot) $dead_a \in \mathbf{T}$.

The state variables of the static transportation problem consist of $|A|$ location variables, one for each asset, and $|P|$ location variables, one for each package. We denote by S the state space formed by taking the cross product of the set of values for each of the state variables. At each point in time $t \in \mathbf{T}$, each asset $a \in A$ is in some location or direct route $loc(a, t) \in \{L, R\}$ and each package $p \in P$ is contained in some asset or location $loc(p, t) \in \{A, L\}$. There are no further restrictions on the movement of packages and assets except those of sources and sinks defined above. Specifically, each package can be loaded onto and unloaded from assets multiple times during its transit from source to sink and each asset can move freely between locations as defined by the graph (L, R). We assume that the sets of assets and packages are fixed and the properties of each asset and package are known in advance (*i.e.*, at time $t = 0$).

Each static transportation problem has performance criteria for comparing solutions to various instances of the problem. A solution to an instance of a static transportation

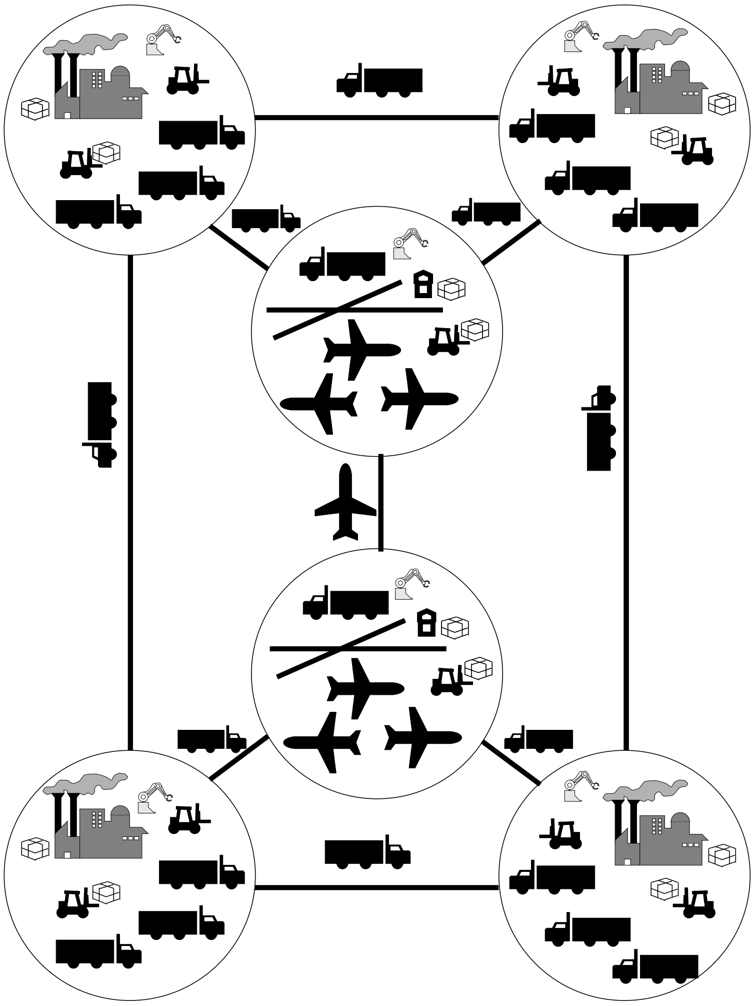


Figure 1: Transportation network

problem specifies values for the state variables defined above for each point in time. To capture this general metric for performance, we introduce the following measure:

- a function, $g : H \to \mathbf{R}$

Where H is the set of state-space histories, $H = \{h | h : T \to S\}$. Thus, the function g assigns a value to the solution of the problem based on the locations of packages and assets at each point in time. This value may or may not take into consideration the given source, sink, release date, deadline and requirement for each package and the source, sink, release date and deadline for each transportation asset. If we think of the transportation problem in terms of a linear program, the function g encompasses both the objective function and the constraints in determining the value of a state-space history.

3 Greyhound Package Scheduling Problem

A further specialization of the static transportation problem given in Section 2 is the *Greyhound Package Scheduling Problem*. In this problem, we enforce restrictions on the movement of assets in addition to specifying the sources and sinks as in the static transportation problem. Specifically, we provide a fixed schedule for each asset. Each schedule resembles a bus schedule. This important restriction simplifies the problem and allows us to focus on the complexity remaining in the unrestricted movement of packages.

To motivate this restriction, consider a situation in which a bus company has created a set of schedules for its buses that it has optimized for some performance criteria based on its passenger operations. For example, it has minimized the time a passenger has to wait before the next bus arrives given that passengers arrive at fixed times. The company then realizes that, although the schedules are optimal for its desired performance criteria, there is unused capacity available on some buses. The bus company would like to make use of this capacity by managing a package delivery business on the side. Given a set of packages to be delivered, the bus company has the new problem of determining a set of loads/unloads for each package that will maximize (minimize) some new performance criteria subject to the constraints on the packages and the fixed schedules and unused capacities of the buses. The bus schedules and unused capacities are fixed in advance by the bus company's solution to the passenger routing problem.[2]

The mathematical formulation of this problem is nearly identical to the static transportation problem of Section 2, except that the asset properties are replaced by the following. Each asset $a \in A$ has:

- capacity $cap_a \in \mathbf{R}$, and
- a fixed schedule, *i.e.,* for each $t \in \mathbf{T}$, a location $loc_{a,t} \in \{L, R\}$.

In addition, we do not include the optional deadlines on packages and assume all locations to have infinite storage capacity. It should also be clear that the fixed asset schedules not only subsume the asset properties (speed, source, sink, release date) but, also the lengths of direct routes (and, as we will see in Section 5, the direct routes themselves). The basic entities of the greyhound problem are depicted in Figure 2.

The fixed asset schedules can be summarized by a table in which only changes in asset locations corresponding to scheduled stops on routes are depicted. A fixed asset schedule table that corresponds to the example problem in Figure 2 is:

	first stop on route	second stop on route	third stop on route	fourth stop on route	fifth stop on route	sixth stop on route	cycle time*
Bus a	12:00 (A)	12:09 (C)	12:18 (B)	repeat			30.6 min
Bus b	12:00 (B)	12:24 (F)	12:39 (E)	12:54 (F)	repeat		78 min
Bus c	12:00 (E)	12:13 (D)	12:28 (C)	12:41 (B)	1:00 (A)	repeat	91 min

*Cycle times assume 3 minutes per stop for loading/unloading. In this simplification of the problem, we neglect any variance in the travel times due to traffic conditions, etc., and assume that the fixed schedules (bus schedules) are guaranteed. Also, note that, although the example problem has periodic schedules for assets, this is not required. Schedules can be terminating as well.

The example instance of the greyhound problem given in the above table and Figure 2 has the following detailed constituents:

---

[2] In practice, cargo is not stored in the passenger area but, rather, in the baggage area. The unused capacity at any time will vary though we have assumed it is fixed. A more convincing version of this problem can be framed in terms of containerized freight.

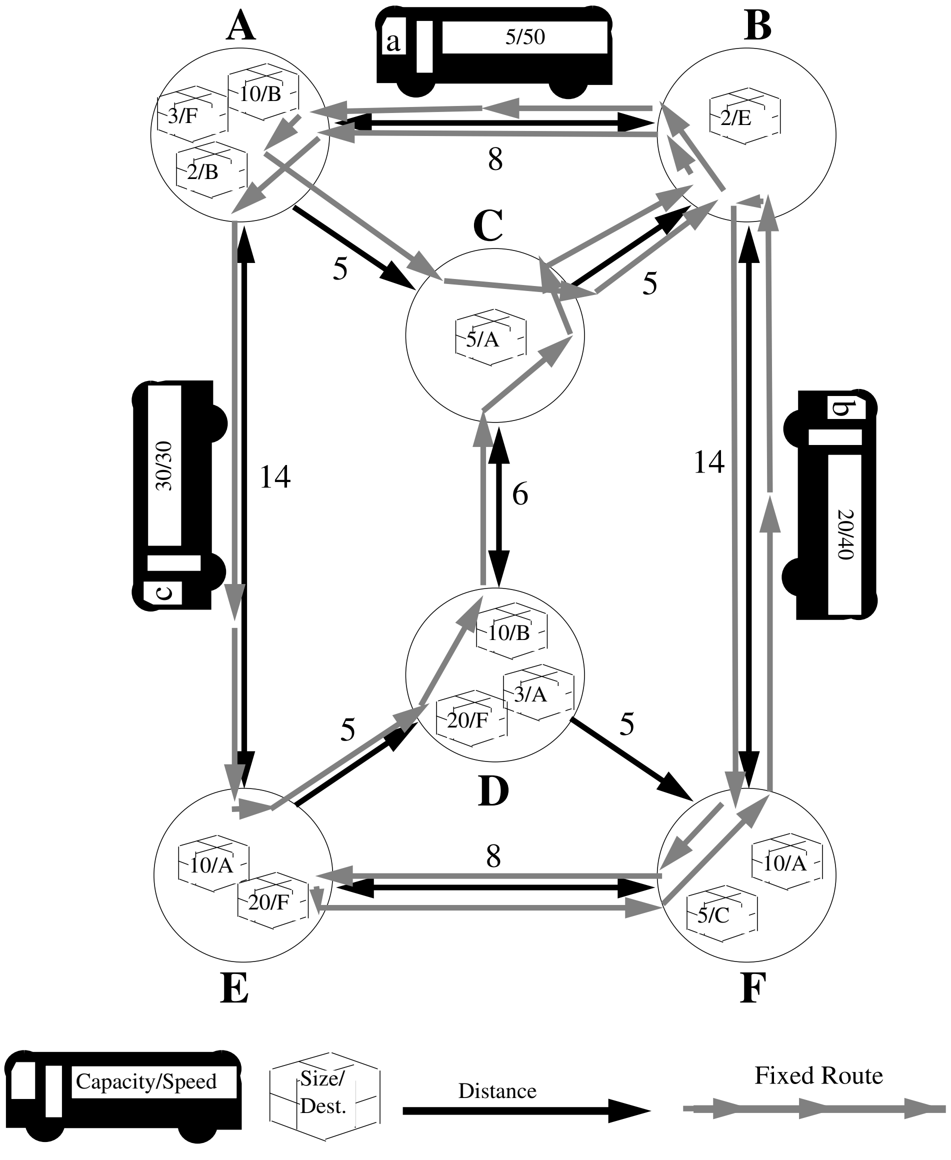


Figure 2: Greyhound Problem – Basic Entities

- $L = \{A, B, C, D, E, F\}$
  - $cap_A = cap_B = \ldots = cap_F = \infty$;
- $R = \{(A, B)$, (B, A), (A, C), (C, B), (A, E), (E, A), (C, D), (D, C), (B, F), (F, B), (E, D), (D, F), (E, F), $(F, E)\}$
  - $len_{(A,B)} = len_{(B,A)} = 8$, $len_{(A,C)} = 5$, $len_{(C,B)} = 5$, $len_{(A,E)} = len_{(E,A)} = 14$, $len_{(C,D)} = len_{(D,C)} = 6$, $len_{(B,F)} = len_{(F,B)} = 14$, $len_{(E,D)} = 5$, $len_{(D,F)} = 5$, $len_{(E,F)} = len_{(F,E)} = 8$;
- $P = \{p_1, p_2, \ldots, p_{12}\}$
  - $src_{p_1} = A$, $snk_{p_1} = B$, $req_{p_1} = 2$, $rel_{p_1} =$ 12:00,
  - $src_{p_2} = A$, $snk_{p_2} = B$, $req_{p_2} = 10$, $rel_{p_2} =$ 12:00,
  - $src_{p_3} = A$, $snk_{p_3} = F$, $req_{p_3} = 3$, $rel_{p_3} =$ 12:00,
  - $src_{p_4} = B$, $snk_{p_4} = E$, $req_{p_4} = 2$, $rel_{p_4} =$ 12:00,
  - $src_{p_5} = C$, $snk_{p_5} = A$, $req_{p_5} = 5$, $rel_{p_5} =$ 12:00,
  - $src_{p_6} = D$, $snk_{p_6} = A$, $req_{p_6} = 3$, $rel_{p_6} =$ 12:00,
  - $src_{p_7} = D$, $snk_{p_7} = B$, $req_{p_7} = 10$, $rel_{p_7} =$ 12:00,
  - $src_{p_8} = D$, $snk_{p_8} = F$, $req_{p_8} = 20$, $rel_{p_8} =$ 12:00,
  - $src_{p_9} = E$, $snk_{p_9} = A$, $req_{p_9} = 10$, $rel_{p_9} =$ 12:00,
  - $src_{p_{10}} = E$, $snk_{p_{10}} = F$, $req_{p_{10}} = 20$, $rel_{p_{10}} =$ 12:00,
  - $src_{p_{11}} = F$, $snk_{p_{11}} = A$, $req_{p_{11}} = 10$, $rel_{p_{11}} =$ 12:00,
  - $src_{p_{12}} = F$, $snk_{p_{12}} = C$, $req_{p_{12}} = 5$, $rel_{p_{12}} =$ 12:00;
- $A = \{a, b, c\}$
  - $cap_a = 5$, $loc_{a,12:00} = \ldots = loc_{a,12:03} = A$, $loc_{a,12:04} = \ldots = loc_{a,12:08} = (A, C)$, $loc_{a,12:09} = \ldots = loc_{a,12:12} = C$, $loc_{a,12:13} = \ldots = loc_{a,12:17} = (C, B)$, $loc_{a,12:18} = \ldots = loc_{a,12:21} = B$, $loc_{a,12:22} = \ldots = loc_{a,12:30} = (B, A)$, repeat;
  - $cap_b = 20$, $loc_{b,12:00} = \ldots = loc_{b,12:03} = B$, $loc_{b,12:04} = \ldots = loc_{b,12:23} = (B, F)$, $loc_{b,12:24} = \ldots = loc_{b,12:27} = F$, $loc_{b,12:28} = \ldots = loc_{b,12:38} = (F, E)$, $loc_{b,12:39} = \ldots = loc_{b,12:42} = E$, $loc_{b,12:43} = \ldots = loc_{b,12:53} = (E, F)$, $loc_{b,12:54} = \ldots = loc_{b,12:57} = F$, $loc_{b,12:58} = \ldots = loc_{b,1:17} = (F, B)$, repeat;

- $cap_c = 30$, $loc_{c,12:00} = \ldots = loc_{c,12:03} = E$, $loc_{c,12:04} = \ldots = loc_{c,12:12} = (E, D)$, $loc_{c,12:13} = \ldots = loc_{c,12:16} = D$, $loc_{c,12:17} = \ldots = loc_{c,12:27} = (D, C)$, $loc_{c,12:28} = \ldots = loc_{c,12:31} = C$, $loc_{c,12:32} = \ldots = loc_{c,12:40} = (C, B)$, $loc_{c,12:41} = \ldots = loc_{c,12:44} = B$, $loc_{c,12:45} = \ldots = loc_{c,12:59} = (B, A)$, $loc_{c,1:00} = \ldots = loc_{c,1:03} = A$, $loc_{c,1:04} = \ldots = loc_{c,1:30} = (A, E)$, repeat.

The performance criterion for the greyhound problem is to find a set of schedules for packages (*i.e.,* for each package, a series of loads/unloads onto assets such that the package begins at its source and ends at its sink), such that the total time between release date and delivery date (at sink) for all packages, subject to the capacity constraints of the assets, is minimized. Formally, we have

$$\min \sum_{p \in P} delivery(p) - rel_p$$

where $delivery(p) = \min_{t \in \mathbf{T}}\{loc(p, t) = snk_p\}$ is the time at which package p is delivered at its sink. The performance criterion is discussed more in Sections 5 and 6.1.

4 Greyhound Problem is NP-Complete

In this section, we show that the greyhound problem is NP-complete through a reduction from 3-Satisfiability (3SAT). This reduction not only provides us with a measure of complexity for the greyhound and general transportation problems, but also gives us additional insight into the structure of the greyhound problem. In Sections 5 and 6, we exploit this structure to provide approximation algorithms for restricted versions of the greyhound problem.

It is convenient for our reduction to consider the greyhound problem of Section 3 as a feasibility problem rather than an optimization problem. We can express the greyhound feasibility problem as a decision problem as follows:

INSTANCE: Directed graph $G = (L, R)$; set $T \subset \mathbf{T}$ of time points; set P of $|P|$ packages; for each $p \in P$, $req_p \in \mathbf{R}$, $src_p \in L$, $snk_p \in L$, $rel_p \in \mathbf{T}$; set A of $|A|$ assets; for each $a \in A$, $cap_a \in \mathbf{R}$ and for each $t \in \mathbf{T}$, a location $loc_{a,t} \in \{L, R\}$.

QUESTION: Is there a function $loc : P \times T \rightarrow \{A, L\}$ such that

1. for each $a \in A$ and $t \in T$, $\sum_{p \in \{q \in P | loc(q,t)=a\}} req_p \leq cap_a$ (capacity constraints);
2. for each $p \in P$ and $t \in T$, either
    - $loc(p, t_{k+1}) = loc(p, t_k)$ (package remains at location or in asset), where t_{k+1} is the time point immediately following t_k ($t_{k+1} - t_k$ may be infinitesimal, if desired),
    - $loc(p, t_{k+1}) = a$ and $loc_{a,t_k} = loc(p, t_k)$ (load), or
    - $loc(p, t_{k+1}) = l$, $loc(p, t_k) = a$ and $loc_{a,t_k} = l$ (unload),

   (conservation of flow constraints); and
3. for each $p \in P$, $loc(p, t) = snk_p$ for some $t \geq rel_p$ (requirement delivery constraints)?

The greyhound optimization problem of Section 3 consists of the above decision problem with the following modifications:

INSTANCE: as above.

QUESTION: Is there a function $loc : P \times T \rightarrow \{A, L\}$ *and a positive integer* K such that

1.,2.,3. as above; and

4. $\sum_{p \in P} delivery(p) - rel_p \leq K$, where $delivery(p) = \min_{t \in T}\{loc(p,t) = snk_p\}$ is the time at which package p is delivered at its sink?

It is clear that the NP-completeness of the greyhound feasibility problem implies the NP-completeness of the greyhound optimization problem and the NP-hardness of the general transportation problem.

3SAT is described by the following decision problem [6]:

INSTANCE Set U of variables, collection C of clauses over U such that each clause $c \in C$ has $|c| = 3$.

QUESTION Is there a satisfying truth assignment for C?

3SAT was shown to be NP-complete by Cook [3] as a transformation from Satisfiability.

Theorem 1 *Greyhound Feasibility is NP-complete.*

Proof (Sketch):

Our proof sketch reduces 3SAT to the greyhound feasibility problem. In [5], Satisfiability is reduced to a restricted version of multicommodity flow by creating a network in which, for each variable, there are two possible paths from source to sink, one corresponding to the positive truth-value for the variable and the other corresponding to its negation. The network is constructed such that flow can only take place through one or the other of these paths, but not both. Our reduction takes a similar approach but requires more attention to time-constrained routes rather than flows. Flow problems center around the flow of a commodity along one or more paths from source to sink. Transportation problems differ from this in that the notion of flow is replaced by discrete movements in time of packages within specific transportation assets along transportation routes.

Without loss of generality, we assume that the 3SAT instance is in conjunctive normal form (CNF, product of sums). Additionally, we assume that each variable is present at most once in any given clause, either negated or unnegated (if this were not the case we could create an equivalent instance in which it is so). Let $m = |C|$ be the number of clauses in our 3SAT instance. In the following reduction, we construct an instance of the greyhound feasibility problem in which there are m unit packages, m sources and m sinks, one for each clause. We then create a transportation network and a set of unit capacity transportation assets such that there are three different paths from source to sink for each clause, one corresponding to each literal in the clause. The package for a clause can be transported on exactly one of these paths. The paths are constructed such that, for any complementary pair of literals (negated and unnegated versions of the same variable) from two different clauses, the two corresponding paths have a common link. We call this common link a *contention.* The common link is a contention because each link in the transportation network can be traversed exactly once by exactly one unit capacity transportation asset. Therefore, of the two paths meeting at a contention, at most one can be used to transport a unit-sized package from its source to its sink. An important detail of the reduction concerns the timing of the transportation asset schedules so that contentions can occur between any two clauses, if necessary.

The details of this reduction are provided here. Let an arbitrary instance of 3SAT in CNF with m clauses and n variables be $C = \{c_1, c_2, \ldots, c_m\}$, $U = \{u_1, u_2, \ldots, u_n\}$, where

$c_i = (w_{i,1}, w_{i,2}, w_{i,3})$ and $w_{i,j} = u_k$ or $\bar{u}_k$ for some k, $1 \leq k \leq n$. We construct an instance of the greyhound problem as follows:

For each $c_i \in C$, we create a source location l_{s_i}, a sink location l_{t_i} and a package p_i with $req_{p_i} = 1$, such that $src_{p_i} = l_{s_i}$, $snk_{p_i} = l_{t_i}$ and $rel_{p_i} = 0$. We then create the transportation network and transportation assets to transport each package from its source to sink. There are two types of direct-routes in our transportation network, *vertical* and *lateral* links. Vertical links provide the sources of contention between complementary pairs of literals. In other words, they allow us to ensure that two clauses with literals corresponding to a variable and its negation depend on that variable being assigned exactly one truth-value, either "true" or "false". Lateral links enable the contention on vertical links by providing a bridge between clauses. In order to ensure that all possible contentions between package paths can occur, each package must pass through exactly $\frac{m(m-1)}{2} + 2$ vertical links, one for each possible contention between literals from different clauses plus one from source and one to sink. Let $S_m = \frac{m(m-1)}{2}$. We create $S_m + 1$ locations $\{l_{i,j,1}, l_{i,j,2}, \ldots, l_{i,j,S_m+1}\}$, in addition to source and sink, corresponding to the endpoints of these links, where $j = 1, 2, 3$ corresponds to the three literals of clause c_i. The reason a package from a single clause must pass through S_m links, even though many enable contentions that do not involve that clause, is so that precise timings of contentions can be accomplished among all possible pairs of clauses. This will become clearer when we discuss the construction and timing of transportation asset schedules.

For a literal corresponding to an unnegated variable, we create a path consisting solely of vertical links in a single chain as shown in Figure 3. These links correspond to direct routes $r_{i,j,k} = (l_{i,j,k}, l_{i,j,k+1})$, $1 \leq k \leq S_m$ as well as $r_{s_i,j,0} = (l_{s_i}, l_{i,j,1})$ and $r_{t_i,j,S_m+1} = (l_{i,j,S_m}, l_{t_i})$, for $j = 1, 2, 3$. For a single clause in which all literals correspond to unnegated variables, we have the transportation network of Figure 4. If there are no literals that correspond to negated variables in the instance of 3SAT (or variables occur either negated or unnegated but not both throughout the instance), then there are no sources of contention and we have a disconnected graph as shown by the example in Figure 5.

For a literal that corresponds to a negated variable, we create a path with a combination of vertical and lateral links. In order for a package to traverse a path corresponding to a literal, it must prevent packages from traversing any paths corresponding to the complementary literal in other clauses. This is accomplished via lateral links. Specifically, consider literal $w_{i,j} = \bar{u}_k$ from clause c_i. For each clause c_l, if c_l contains a negated version of the

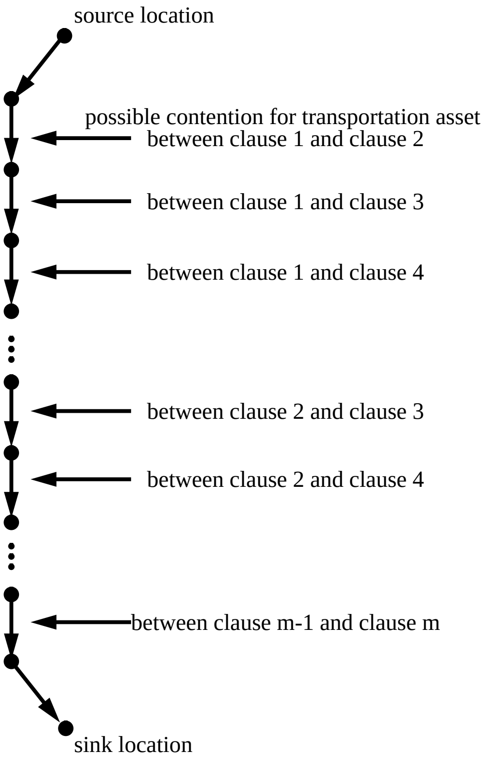


Figure 3: Packages must pass through enough vertical links to account for contentions between literals of any two clauses.

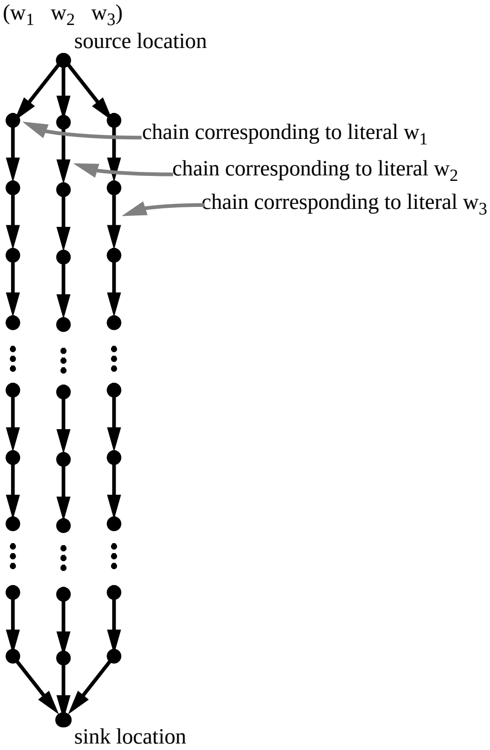


Figure 4: Three possible paths are constructed for each clause.

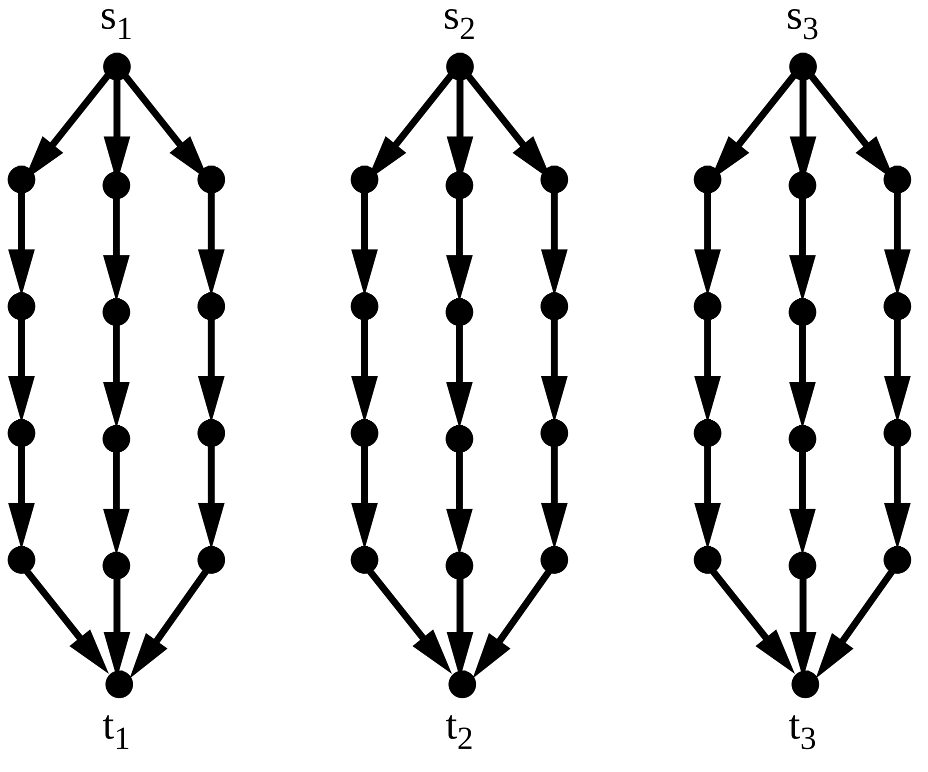


Figure 5: Disconnected graphs are immediately satisfiable, contain no lateral links and, therefore, have no contention for transportation assets.

same variable or no version of that variable, then there is no contention and we create a vertical link (direct route) $r_{i,j,l} = (l_{i,j,l}, l_{i,j,l+1})$. However, if c_l contains an unnegated version of that variable, $w_{l,m} = u_k$, then we set up a contention by constructing the lateral links (direct routes) $r_{i,j,l} = (l_{i,j,l}, l_{l,m,l})$ and $r_{i,j,l'} = (l_{l,m,l+1}, l_{i,j,l+1})$. This forces the package to traverse vertical link $r_{l,m,l} = (l_{l,m,l}, l_{l,m,l+1})$ on its path from source to sink. Therefore, in order for a package taking the path corresponding to an instance of the negated variable to reach its sink it must pass through the vertical links of all clauses with unnegated instances of the same variable. By doing so, it prevents any packages in the other clauses from using these paths. It prevents all or none. In other words, either the negated instances of a single variable block all unnegated instances for that variable in other clauses, or the unnegated instances block all negated instances for that variable.

Figure 6 depicts an example 3SAT problem and its corresponding transportation network. This figure depicts both vertical and lateral links. Note that the package from clause c_2 contends with that of clause c_1 on vertical link $r_{1,1,1}$ and the package from clause c_3 contends with that of clause c_1 on vertical link $r_{1,1,2}$. Either p_1 can traverse the path corresponding to u_1 (u_1 set to true), or p_2 and p_3 can traverse the paths corresponding to $\bar{u}_1$ (u_1 set to false). Also, note that the package from clause c_3 contends with that of clause c_1 on vertical link $r_{1,2,2}$ and that of clause c_2 on vertical link $r_{2,2,3}$.

We have, up to now, neglected the transportation assets. For each direct route created above (both vertical and lateral), we create an asset with capacity 1. We constrain the schedule of the asset to a time interval that provides the crucial synchronization of contentions in the transportation network. In particular, we interleave the travel on vertical links with the travel on lateral links so that contentions across clauses can be handled. This delay between the time one asset arrives at a location from the vertical direction and the next asset leaves in the vertical direction is bounded because at most two lateral links need to be traversed between vertical links. Furthermore, we constrain the travel on vertical links so that they must be traversed serially from top to bottom. Any two packages being transported in the network must traverse corresponding vertical links at the same time. In other words, the movement of assets corresponds to a wavefront in which the levels are reached simultaneously across all clauses. A package which pauses at a location will not be delivered to its sink. Similarly, there is no way to skip a level.

Formally, we create $|R|$ assets $a_{i,j,k}$ such that for each $r_{i,j,k} \in R$, $cap_{a_{i,j,k}} = 1$, and

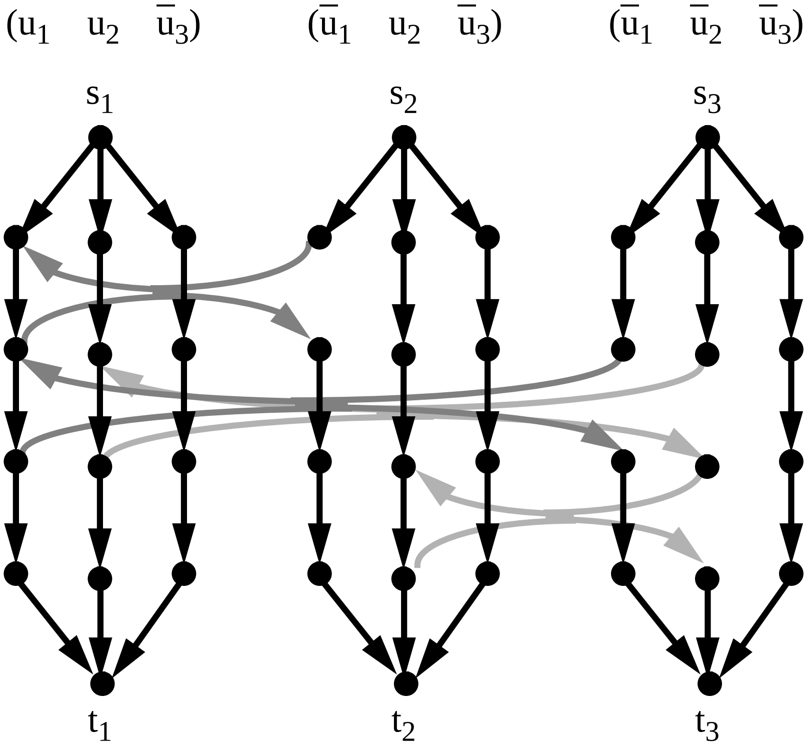


Figure 6: Example transportation network with lateral links to enable contentions across clauses.

- $loc_{a_{i,j,0},0} = l_{s_i}$
- $loc_{a_{i,j,0},1/3} = l_{i,j,1}$
- if $r_{i,j,k} = (l_{i,j,k}, l_{i,j,k+1})$ then
  - $loc_{a_{i,j,k},k} = l_{i,j,k}$
  - $loc_{a_{i,j,k},k+1/3} = l_{i,j,k+1}$
- if $r_{i,j,l} = (l_{i,j,l}, l_{l,m,l})$ then
  - $loc_{a_{i,j,l},l-1/3} = l_{i,j,l}$
  - $loc_{a_{i,j,l},l} = l_{l,m,l}$
- if $r_{i,j,l'} = (l_{l,m,l+1}, l_{i,j,l+1})$ then
  - $loc_{a_{i,j,l'},l+1/3} = l_{l,m,l+1}$
  - $loc_{a_{i,j,l'},l+2/3} = l_{i,j,l+1}$
- $loc_{a_{i,j,S_m+1},S_m+1} = l_{i,j,S_m+1}$
- $loc_{a_{i,j,S_m+1},S_m+4/3} = l_{t_i}$

The above capacities and asset schedules define the role of the transportation assets in the reduction. All packages are released at time 0 and cannot wait at the sources since the only assets to visit the source nodes visit at time 0, exclusively. Finally, we create $|T| = 3(S_m + 2)$ time points, $T = \{0, 1/3, 2/3, 1, \ldots, S_m + 5/3, S_m + 2\}$ to correspond to the asset schedules above. The time points are labeled in third units for convenience.

To better understand the asset schedules derived above, consider a path from Figure 6. Consider the path $\langle l_{s_3}, l_{3,2,1}, l_{3,2,2}, l_{1,2,2}, l_{1,2,3}, l_{3,2,3}, l_{2,2,3}, l_{2,2,4}, l_{3,2,4}, l_{t_3}\rangle$ isolated in Figure 7. The transportation assets transport package p_3 along this route as follows:

1. asset $a_{3,2,0}$ loads the package at time 0 at location l_{s_3} and unloads the package at time 1/3 at location $l_{3,2,1}$,
2. the package waits at location $l_{3,2,1}$ from time 1/3 until time 1,

3. asset $a_{3,2,1}$ loads the package at time 1 at location $l_{3,2,1}$ and unloads the package at time 4/3 at location $l_{3,2,2}$,
4. the package waits at location $l_{3,2,2}$ from time 4/3 until time 5/3,
5. asset $a_{3,2,2}$ loads the package at time 5/3 at location $l_{3,2,2}$ and unloads the package at time 2 at location $l_{1,2,2}$,
6. asset $a_{1,2,2}$ loads the package at time 2 at location $l_{1,2,2}$ and unloads the package at time 7/3 at location $l_{1,2,3}$,
7. asset $a_{3,2,2'}$ loads the package at time 7/3 at location $l_{1,2,3}$ and unloads the package at time 8/3 at location $l_{3,2,3}$,
8. asset $a_{3,2,3}$ loads the package at time 8/3 at location $l_{3,2,3}$ and unloads the package at time 3 at location $l_{2,2,3}$,
9. asset $a_{2,2,3}$ loads the package at time 3 at location $l_{2,2,3}$ and unloads the package at time 10/3 at location $l_{2,2,4}$,
10. asset $a_{3,2,3'}$ loads the package at time 10/3 at location $l_{2,2,4}$ and unloads the package at time 11/3 at location $l_{3,2,4}$,
11. the package waits at location $l_{3,2,4}$ from time 11/3 until time 4,
12. finally, asset $a_{3,2,4}$ loads the package at time 4 at location $l_{3,2,4}$ and unloads the package at time 13/3 at location l_{t_3}.

In Figure 7, the location indices are indicated by triplets and the time intervals are indicated by bracketed pairs.

The reduction given above can be accomplished in polynomial time. Membership in NP is straightforward. It remains to show that this transformation produces a greyhound feasibility problem instance in which a feasible solution can be found if and only if the corresponding 3SAT instance is satisfiable. First, we summarize the important properties of the reduction:

1. contentions can only arise between complementary versions of the same variable;

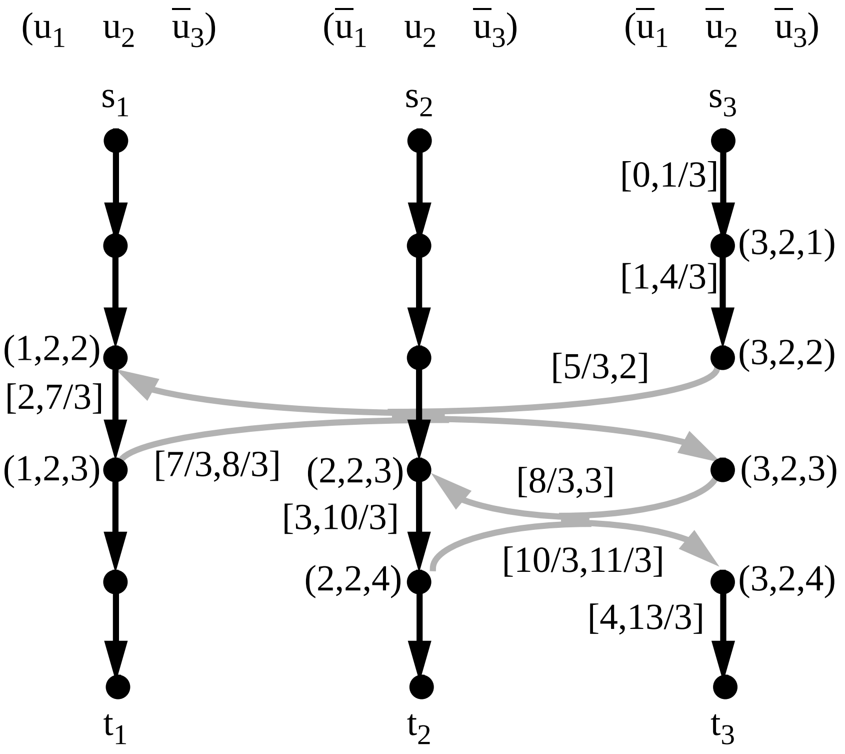


Figure 7: Example timed path for transporting package p_3 from l_{s_3} to l_{t_3}.

2. when a contention arises, only one of the two paths can be traversed;
3. all possible contentions across clauses and variables are enabled;
4. assets travel vertically in a lockstep wavefront; no cycles can occur;
5. a package has exactly three paths from source to sink; it can traverse other paths, but they will not lead it to its sink; once it embarks on a path, it cannot cross-over to one of the other two;
6. at most one pair of lateral links is traversed for any vertical link; the lateral links go from negated instance to unnegated instance and back and must traverse a vertical link at the chain corresponding to an unnegated version of the variable before it can traverse another lateral link; in other words, it can only contend with any given clause once and must traverse the vertical link for each clause before moving on.

(if) Assume that there exists a feasible solution to the greyhound problem. Then each package is delivered to its sink and no transportation asset is assigned a load that exceeds its capacity. Exactly one path is taken that corresponds to each clause of the 3SAT instance. Assign a value of “true” to each literal corresponding to a path traversed by a package in its routing from source to sink (implicitly assigning ”false” to its negation). Arbitrarily assign truth values to any variables left unassigned. By properties 1, 2 and 3 no variable is assigned both the values “true” and “false”. Therefore, there is at least one literal assigned a value of “true” for each clause and the 3SAT expression is satisfiable.

(only if) Assume that the expression is satisfiable. For each clause, select one of the literals that has been assigned the value “true” in the satisfying truth assignment. For each package, construct a route (series of loads and unloads for the package) that corresponds to traversing the path associated with this literal. Since the expression is satisfiable, the package will have no contentions along this path (if it did have a contention then both the negated and unnegated version of a variable would be assigned the value of “true” and this would not be a satisfying truth assignment). Therefore, each package is delivered to its sink, no transportation asset is assigned a load that exceeds its capacity and we have a feasible solution to the greyhound problem instance.

Thus, both requirements are met. Q.E.D.

Given the above result, there is no polynomial time optimal algorithm to solve the greyhound problem unless $P = NP$. However, there are ways to approach the greyhound problem if we are willing to relax our need for optimal solutions. We look to construct *approximation algorithms* to solve the greyhound problem. Approximation algorithms give up the notion of an "optimal solution" to an optimization problem but, in turn, guarantee that the quality of the solution of the resulting algorithm is within a factor of the optimal solution for any problem instance.

The approximation algorithms that we construct are based on randomization techniques for flow and other related problems developed by Raghavan and Thompson [9]. This work introduces the concept of *randomized rounding* in which high-probability approximations are obtained for integer problems by first solving a relaxed linear program and then using the resulting real-number values to determine integer values for the variables. (See [2] for recent results in using linear programming for solving complex transportation problems).

5 Formulating a Multicommodity Flow Problem from an Instance of the Greyhound Problem by Inducing a Flow Graph

In [5], Even, Itai and Shamir show that the *simple directed two-commodity integral flow problem* is NP-complete. This problem is described by the following decision problem:

INSTANCE Directed graph $G = (V, E)$; specified vertices s_1, s_2, t_1, and t_2; capacity $c(e) = 1$, for each $e \in E$; and requirements $R_1, R_2 \in \mathbf{Z}$.

QUESTION Are there two flow functions $f_1, f_2 : E \rightarrow \mathbf{Z}$ such that

1. for each $e \in E$, $f_1(e) + f_2(e) \leq c(e)$,
2. for each $v \in V - \{s, t\}$ and $i \in \{1, 2\}$, flow f_i is conserved at v, and
3. for $i \in \{1, 2\}$, the net flow into t_i is at least R_i?

The two-commodity version of the problem generalizes to an m-commodity problem called *integral multicommodity flow*, in which there are m commodities, each with a designated source, sink and requirement. The problem is then to produce m flow functions such that the combined flows do not violate the constraints generalized from the two-commodity case. The integral version is NP-complete [6], although the fractional version of the problem is solvable by linear programming in polynomial time. The multicommodity flow problem has been well-studied and there are several approximation algorithms for its solution. In this section, we provide a reduction from any instance of the greyhound problem to an instance of the integral multicommodity flow problem. Expressing a greyhound problem as a multicommodity flow problem allows us to apply solution techniques designed for multicommodity flow to an instance of the greyhound problem.

As noted in Section 3, the fixed asset schedules of the greyhound problem subsume the properties of the assets and the lengths of the direct routes of the transportation graph. We can take this a step further by noting that the direct routes themselves are completely subsumed by the fixed asset schedules. Using this fact we can collapse the fixed asset schedules and transportation graph from any greyhound problem instance into a single structure. In the reduction from 3SAT to greyhound problem of Section 4, we create an asset for every direct route. In other words, a direct-route has no meaning other than to support an asset. Likewise, in our transformation from greyhound problem to multicommodity flow, we can ignore direct routes and rely on the fixed asset schedules in constructing the flow graph.

In constructing the flow graph we must be certain to capture all the paths (induced by fixed asset schedules) that are possible in the transportation graph, including paths that cycle over time. Since periodic schedules lead to an infinite number of paths, we restrict our problem instance to a specified interval of time (otherwise, we potentially would have to deal with an infinite flow graph). Another difference is that packages travel in discrete time steps in a transportation graph whereas flow of a commodity is instantaneous in a flow graph. We resolve this problem by duplicating each location for each time point that corresponds to an event for that location (*i.e.,* a package release date or asset-stop at that location). Also, packages are allowed to wait (be stored) at locations to be loaded onto an asset at a future time. This is also not possible in flow graphs. To overcome this we provide edges from each instance of the same location across time. Flow along these edges corresponds to waiting at the location. Finally, since there are duplicate nodes corresponding to the same location at different times, there are no distinct termination nodes for the flow graph. We resolve this by creating distinct termination nodes with edges coming from the corresponding location for each of its duplications across time.

In Section 4, we are primarily concerned with feasibility version of the greyhound problem. In general, we would like to consider the optimization versions of the greyhound and the multicommodity flow problems as well. The performance criterion of Sections 3 and 4 concerns minimizing the total time between release date and delivery date (at sink) for all packages. Formally, $\min \sum_{p \in P} delivery(p) - rel_p$ where $delivery(p) = \min_{t \in \mathbf{T}}\{loc(p,t) = snk_p\}$ is the time at which package p is delivered at its sink. Although it doesn't change the value at all, it is convenient to explicitly decompose this value into a summation of smaller values. For any package p, $delivery(p) - rel(p)$ can be decomposed into a series of values representing the time needed to travel a particular leg of the trip (from one location to another, aboard a specific asset) or the time spent waiting to be loaded onto an asset at a particular location. We introduce the term *cost*, measured in time units, to capture this notion. Since the asset schedules are fixed, the cost of using a particular asset to deliver a package from one location to another, along a direct route, starting at a particular time, is fixed in advance. Also, the cost of waiting for a particular asset to arrive at a location at a particular time is fixed in advance. These costs are implicit in the problem instance and are made explicit in the transformation to a multicommodity flow problem below. As mentioned above, waiting at a node is not relevant in the multicommodity flow formulation, so the cost of waiting at a node is transferred to the cost of traversing an edge in the flow graph from a location to the same location, across time. Therefore, cost is associated with edges only, in the flow graph.

Formally, given an instance of the greyhound problem, including transportation graph, fixed asset schedules and packages, we induce a flow graph as follows:

- pick some time interval;
- for each asset $a \in A$:
    - for each scheduled stop $i = stop_{a,t} \in L$ within the time interval ($stop_{a,t} = loc_{a,t}$ such that $loc_{a,t} \in L$ and $loc_{a,t-\epsilon} \in R$, *i.e.,* asset a arrives at location $loc_{a,t}$ at time $t \in \mathbf{T}$):

      asset stop node: create a node $l_{i,t}$;

      asset travel edge: create a directed edge $(l_{i,t}, l_{j,t'})$ with capacity $cap_{(l_{i,t},l_{j,t'})} = cap_a$ and cost $cost_{(l_{i,t},l_{j,t'})} = t' - t$, where $j = stop_{a,t'} \in L$ is the next stop on asset a's fixed schedule ($t' > t$);

- for each package $p \in P$:

  package release node: create a node l_{src_p,rel_p} such that package p is released at source node $src_p \in L$ at release date $rel_p \in \mathbf{T}$;

- for each time step $t \in \mathbf{T}$ within the time interval, if $l_{i,t}$ corresponds to a node created above (either an asset stop node or a package release node) then:

  package wait (storage) edge: create a directed edge $(l_{i,t}, l_{i,t'})$ with infinite capacity $cap_{(l_{i,t},l_{i,t'})} = \infty$ and cost $cost_{(l_{i,t},l_{i,t'})} = t' - t$; where t' is minimum time step greater than t corresponding to node created above for location $i \in L$;

- for each $snk \in \{snk_p | p \in P\}$

  package sink node: create a node l_{snk};

  package sink edges: for each asset stop node or package release node $l_{i,t}$ created above, if $i = snk$ then create a directed edge $(l_{i,t}, l_{snk})$ with infinite capacity $cap_{(l_{i,t},l_{snk})} = \infty$ and zero cost $cost_{(l_{i,t},l_{snk})} = 0$.

The edges and nodes created above form the flow graph $G = (V, E)$. It is sufficient to set the package-sink-edge and package-wait-edge capacities equal to the sum of all package requirements $\sum_{p \in P} req_p$, rather than infinity.

In addition to the induced flow graph, we complete the transformation from a greyhound problem to a multicommodity flow problem with the following entities: for each package $p \in P$ we have:

- $s_p = src_p$,
- $t_p = snk_p$, and
- $R_p = req_p$

(rel_p is already incorporated into the graph).

An induced flow graph corresponding to our example problem of Figure 2 is given in Figure 8. In this example the time interval is $T = \{12{:}00, \ldots, 1{:}32\}$; location $l_{i,t}$ is represented by a node in row i, column t; and edge $(l_{i,t}, l_{j,t'})$ is represented by a directed arc from $l_{i,t}$

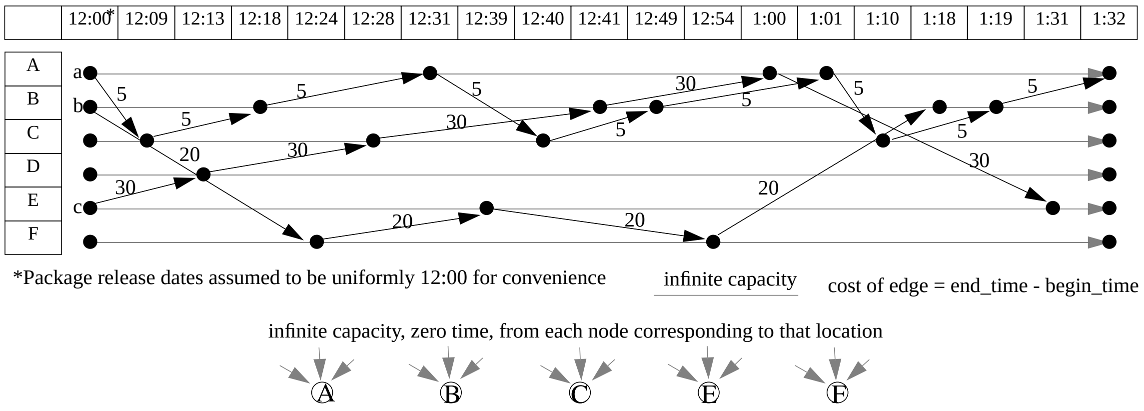


Figure 8: Greyhound Problem – Induced Graph

to $l_{j,t'}$ with its capacity $cap_{(l_{i,t},l_{j,t'})}$ labeled. Unlabeled edges have infinite capacity. Cost for edges, $cost_{(l_{i,t},l_{j,t'})}$, are implicit in the distance across time of each edge in the graph. Note that capacities for edges created from the same transportation asset are the same throughout, and the edges are contiguous. The source nodes are $l_{A,12:00}, l_{B,12:00}, \ldots, l_{F,12:00}$ and the sink nodes are $l_A, \ldots, l_F$ excluding D since no commodity has D as a sink.

6 Application of Randomized Rounding to the Greyhound Problem

Given an instance of the greyhound problem as described in Section 3, we can induce a flow graph as described in Section 5 and solve the problem as a multicommodity flow. While the integral multicommodity flow problem is NP-complete, there are existing polynomial approximation algorithms providing provable performance bounds. One such method is the *randomized rounding* technique of Raghavan and Thompson [9, 8]. In this section, we describe the randomized rounding technique and apply it to the greyhound problem. We also discuss the performance guarantees that this method provides. However, this method is not without its limitations. In Section 7 we describe several variations of the randomized rounding technique, each with its own limitations. These limitations take the form of restrictions on the instances of the greyhound problem that can be solved with any particular variation of randomized rounding, as well as features of problem instances that weaken the provable feasibility and expected performance guarantees.

In this section, we discuss each of the following phases in the randomized rounding technique:

1. Formulate and solve linear programming relaxation of integral multicommodity flow problem,
2. Perform *path-stripping* to change the relaxed solution decomposition from edge-based flow to path-based flow, and
3. Convert the path-based relaxed solution to an integer solution using randomized rounding.

Finally, we discuss the provable feasibility and expected performance guarantees provided by the randomized rounding technique.

6.1 Linear Programming Formulation Phase

Although integral multicommodity flow is NP-complete, fractional multicommodity flow is polynomially equivalent to linear programming [6]. Linear programming is solvable in polynomial time [7], therefore, fractional multicommodity flow is solvable in polynomial time. The first phase of the randomized rounding procedure is to solve the linear program relaxation of the multicommodity flow version of the greyhound problem. This solution serves as a basis for subsequent phases. In this section, we show how to construct a linear program from the multicommodity flow problem constructed in Section 5.

The linear programming formulation below captures the restrictions on flow through the induced graph, including capacity constraints on edges. The restrictions are those introduced for the two-commodity flow problem in Section 5, generalized for the m-commodity case. The linear programming formulation also captures the performance criterion of the greyhound problem as a function of costs on edges, as discussed in Section 5. In the following, constants cap_e represent the capacity of edge $e \in E$ in the induced graph and constants $cost_e$ represent the cost of edge $e \in E$, in terms of time, in the induced graph. The constants s_p, t_p and R_p represent the source node, sink node and requirement, respectively, of package $p \in P$.

For all p and e, let $\hat{X}_{p,e} \in [0,1]$ be a variable describing the fractional flow from package p assigned to edge e (*i.e.,* if package p could be divided into arbitrarily small pieces, the fraction that would be assigned to any given edge in the flow graph). We then would like to solve the following linear program:

$$\min \hat{Z} = \sum_{p \in P} \sum_{e \in E} cost_e \hat{X}_{p,e} \text{ (performance criterion/objective function)}$$

such that

for each $e \in E$

$$\sum_{p \in P} R_p \hat{X}_{p,e} \leq cap_e \text{ (capacity constraints)}$$

for each $p \in P$, $v \in V - \{s_p, t_p\}$

$$\sum_{(u,v) \in E} \hat{X}_{p,(u,v)} = \sum_{(v,u) \in E} \hat{X}_{p,(v,u)} \text{ (conservation of flow constraints)}$$

for each $p \in P$

$$\sum_{(u,t_p) \in E} \hat{X}_{p,(u,t_p)} - \sum_{(t_p,u) \in E} \hat{X}_{p,(t_p,u)} = 1 \text{ (requirement delivery constraints)}$$

with sign restrictions

for each $e \in E, p \in P$ $\hat{X}_{p,e} \geq 0$

for each $e \in E, p \in P$ $\hat{X}_{p,e} \leq 1$

A solution to the relaxed linear programming formulation above describes a flow of packages through the flow graph such that all the constraints are satisfied. A fractional flow value between 0 and 1 is assigned to each of the $|P| \times |E|$ variables of the linear program. The flow value is the fraction of each package assigned to each edge in the flow graph. This corresponds to the fraction of each package that is assigned to a particular asset for a leg of its travel or is designated to wait at a location for an asset to arrive, in the original greyhound problem. This solution minimizes the performance criterion. Thus, assuming the packages can be broken up into arbitrarily small pieces and then reassembled at the sinks, this solution is optimal. The path-stripping and randomized rounding phases of the randomized rounding procedure of Sections 6.2 and 6.3 address the fact that packages cannot be split up this way.

To demonstrate the linear programming formulation of a multicommodity flow problem, we apply it to a subset of our example multicommodity flow graph of Figure 8. As mentioned in Section 5, we may use the sum of all package requirements instead of infinity on edge constraints, where necessary. Let $SR = \sum_{p \in P} req_p$ be this sum. The performance criterion for this problem is:

$$\begin{array}{rcl}
\min \hat{Z} & = & 31\hat{X}_{p1,(l_{A,12:00},l_{A,12:31})} + 9\hat{X}_{p1,(l_{A,12:00},l_{C,12:09})} + \ldots + \\
 & & 24\hat{X}_{p1,(l_{F,12:54},l_{B,1:18})} + 0\hat{X}_{p1,(l_{E,1:31},l_E)} + \\
 & & 31\hat{X}_{p2,(l_{A,12:00},l_{A,12:31})} + 9\hat{X}_{p2,(l_{A,12:00},l_{C,12:09})} + \ldots + \\
 & & 24\hat{X}_{p2,(l_{F,12:54},l_{B,1:18})} + 0\hat{X}_{p2,(l_{E,1:31},l_E)} \\
 & & + \ldots + \\
 & & 31\hat{X}_{p12,(l_{A,12:00},l_{A,12:31})} + 9\hat{X}_{p12,(l_{A,12:00},l_{C,12:09})} + \ldots + \\
 & & 24\hat{X}_{p12,(l_{F,12:54},l_{B,1:18})} 0\hat{X}_{p12,(l_{E,1:31},l_E)}
\end{array}$$

The capacity constraints are:

$$\begin{array}{lcl}
2\hat{X}_{p1,(l_{A,12:00},l_{A,12:31})} + 10\hat{X}_{p2,(l_{A,12:00},l_{A,12:31})} + \ldots + 5\hat{X}_{p12,(l_{A,12:00},l_{A,12:31})} & \leq & SR \\
2\hat{X}_{p1,(l_{A,12:00},l_{C,12:09})} + 10\hat{X}_{p2,(l_{A,12:00},l_{C,12:09})} + \ldots + 5\hat{X}_{p12,(l_{A,12:00},l_{C,12:09})} & \leq & 5 \\
\ldots & & \\
2\hat{X}_{p1,(l_{F,12:54},l_{B,1:18})} + 10\hat{X}_{p2,(l_{F,12:54},l_{C,1:18})} + \ldots + 5\hat{X}_{p12,(l_{F,12:54},l_{C,1:18})} & \leq & 20 \\
2\hat{X}_{p1,(l_{E,1:31},l_E)} + 10\hat{X}_{p2,(l_{F,1:31},l_E)} + \ldots + 5\hat{X}_{p12,(l_{F,1:31},l_E)} & \leq & SR
\end{array}$$

Similarly for the other constraints and sign restrictions.

Some observations can be made of any feasible solution to this problem. For example, $\hat{X}_{p12,(l_{A,12:00},l_{A,12:31})}$ must be assigned flow zero or else it would violate the conservation of flow constraints (package p_{12} cannot get to location $l_{A,12:00}$ since, in the greyhound problem, it is at location F at time 12:00; *i.e.,* there is a zero flow into $l_{A,12:00}$ for package p_{12} and, thus, there must be a zero flow out of this node as well). Also, $\hat{X}_{p5,(l_{C,12:00},l_{C,12:09})}$ must

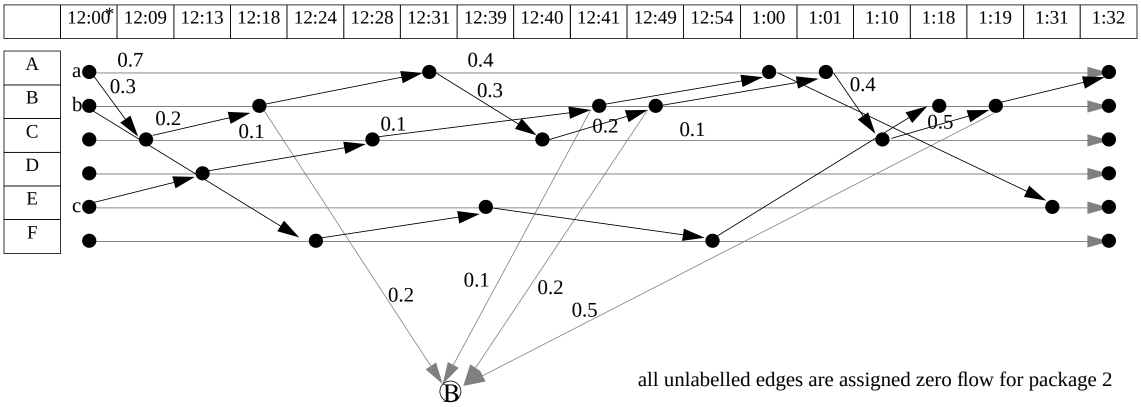


Figure 9: Edge Flow Values Assigned by Feasible Linear Program Solution - Package p_2 Only

be assigned flow 1 (all the flow of package p_5) or else there would be no way to satisfy the requirement delivery constraints for package p_5. Requirement delivery constraints in conjunction with conservation of flow constraints guarantee that all the requirement for a package flows from the source as well as flows to the sink, in any feasible solution.

6.2 Path-Stripping Phase

A solution to the relaxed linear programming formulation of a greyhound problem assigns a fractional flow value between 0 and 1 to each variable. For example, one feasible (though, not necessarily optimal) solution for the assignment of package p_2's flow may include: $\hat{X}_{p2,(l_{A,12:00},l_{A,12:31})} = 0.7$, $\hat{X}_{p2,(l_{A,12:00},l_{C,12:09})} = 0.3$, $\hat{X}_{p2,(l_{C,12:09},l_{B,12:18})} = 0.2$, $\hat{X}_{p2,(l_{C,12:09},l_{C,12:28})} = 0.1$, $\hat{X}_{p2,(l_{C,12:28},l_{B,12:41})} = 0.1$, $\hat{X}_{p2,(l_{A,12:31},l_{A,1:01})} = 0.4$, $\hat{X}_{p2,(l_{A,12:31},l_{C,12:40})} = 0.3$, $\hat{X}_{p2,(l_{C,12:40},l_{B,12:49})} = 0.2$, $\hat{X}_{p2,(l_{C,12:40},l_{C,1:10})} = 0.1$, $\hat{X}_{p2,(l_{A,1:01},l_{C,1:10})} = 0.4$, $\hat{X}_{p2,(l_{C,1:10},l_{B,1:18})} = 0.5$, $\hat{X}_{p2,(l_{B,12:18},l_B)} = 0.2$, $\hat{X}_{p2,(l_{B,12:41},l_B)} = 0.1$, $\hat{X}_{p2,(l_{B,12:49},l_B)} = 0.2$, and $\hat{X}_{p2,(l_{B,1:18},l_B)} = 0.5$. All other variables are assigned zero flow. This partial solution is shown in Figure 9.

The flow value of a variable is the fraction of a package assigned to an edge in the flow graph. We ultimately want to assign a single path in the flow graph to each package. As a step toward this goal, we re-interpret the flow values for each package as being assigned to a set of paths, rather than a set of edges. Subsequently, in Section 6.3, we choose a single path for the package from this set of paths. In this section, we describe an efficient procedure for re-interpreting the edge-based flows as path-based flows.

The example solution of Figure 9 shows a set of edge-based flow assignments for package p_2. Consider the package flowing through this graph. Edge $(l_{A,12:00}, l_{A,12:31})$ is assigned a flow of 0.7. Similarly edges $(l_{A,12:00}, l_{C,12:09})$, $(l_{C,12:09}, l_{B,12:18})$, $(l_{C,12:09}, l_{C,12:28})$ and $(l_{B,12:18}, l_B)$ are assigned flows of 0.3, 0.2, 0.1 and 0.2, respectively. Rather than looking at these edge-flows in isolation, we can look at them as the splitting of a package as it flows through the graph. At location $l_{A,12:00}$ the package is split into two pieces, one of requirement 0.7 and one of requirement 0.3. Then, at location $l_{C,12:09}$ the 0.3 portion of the package is again split into two pieces, one of requirement 0.2 and the other of requirement 0.1. Given this interpretation, we notice that the path $\langle(l_{A,12:00}, l_{C,12:09}), (l_{C,12:09}, l_{B,12:18}), (l_{B,12:18}, l_B)\rangle$ carries 0.2 of the package from source to sink. The path-stripping procedure of Raghavan and Thompson [9] explicitly constructs this interpretation by assigning a flow value to each possible path (with nonzero flow) of each package in the flow graph. This path is assigned a value corresponding to the largest possible common flow across all edges of the path (*i.e.,* the minimum flow on any one edge in the path). In this way, the path stripping technique merely relabels the solution from an edge-based flow to a path-based flow. The solution is not changed.

This procedure is more formally defined as follows. Given the induced graph $G = (V, E)$ and the fractional flows $\{\hat{X}_{p,e}\}$ derived from the linear program we create a set of paths Γ_p for each package $p \in P$ with the algorithm (sketch) below. For each $\gamma \in \Gamma_p$, $\Pr\{\gamma\}$ is the assigned probability that p will traverse γ.

```
for each package $p \in P$ with source $s_p$ and sink $t_p$
  $G_p = (V, E_p)$, where $E_p = \{e \mid e \in E \text{ and } \hat{X}_{p,e} \neq 0\}$
  $\Gamma_p = \emptyset$
  while $\gamma = findpath(s_p, t_p, G_p)$
    $\hat{X}_m = \min_{e \in \gamma} \hat{X}_{p,e}$
    $\Pr\{\gamma\} = \hat{X}_m$
    $\Gamma_p = \Gamma_p \cup \gamma$
    for each $e \in \gamma$
      $\hat{X}_{p,e} = \hat{X}_{p,e} - \hat{X}_m$
      if $\hat{X}_{p,e} = 0$ then $E_p = E_p - \{e\}$
```

The above process terminates since *findpath* can find at most $|E_p|$ paths per package, as at least one edge is removed per iteration of the while loop. Notice that, in the above procedure, the flow assigned to a path is called its *probability*. The reasons for this are explained in Section 6.3.

The results of path stripping for package p_2 are shown in Figure 10. In this figure, we begin with 15 edge-based flows and end with five path-based flows. The five paths contained in Γ_{p_2} comprise all possible paths (with non-zero flow) that package p_2 could take, given the edge flow values assigned by the linear program. These paths have the assigned path-flows (probabilities) $0.4, 0.2, 0.1, 0.2, 0.1$ which naturally sum to one (if they did not sum to one then the requirement delivery constraint of the linear program would have been violated). In Section 6.3, these path-flows (probabilities) are used to select one path from the five that will transport all of package p_2.

6.3 Randomized Rounding Phase

The randomized rounding phase concludes the randomized rounding procedure. In the path-stripping phase of Section 6.2, the edge-based flows that resulted from the relaxed linear programming solution are re-interpreted as path-based flows. Since the values of the flows on all paths for a given package sum to one, these flows can be considered probabilities. The randomized rounding phase chooses a single path for each package (independently from other packages), based on these probabilities.

Formally, we can define this procedure as follows: Given a set of paths Γ_p for each package $p \in P$ and a corresponding flow value (probability) $\Pr\{\gamma\}$ for each path $\gamma \in \Gamma_p$; for each package $p \in P$:

1. Perform a probability experiment with $|\Gamma_p|$ choices (such as throwing a $|\Gamma_p|$-faced die), one corresponding to each $\gamma \in \Gamma_p$;
2. Select one choice based on the probabilities $\{\Pr\{\gamma\}\}$ and call this choice γ_p; this choice γ_p is the path selected for p.

This procedure selects one and only one path for each $p \in P$, γ_p, to transport all of package p's requirement R_p from source s_p to sink t_p. In other words, we set $X_{p,e} = 1$ for each $e \in \gamma_p$ and $X_{p,e} = 0$ for each $e \notin \gamma_p$.

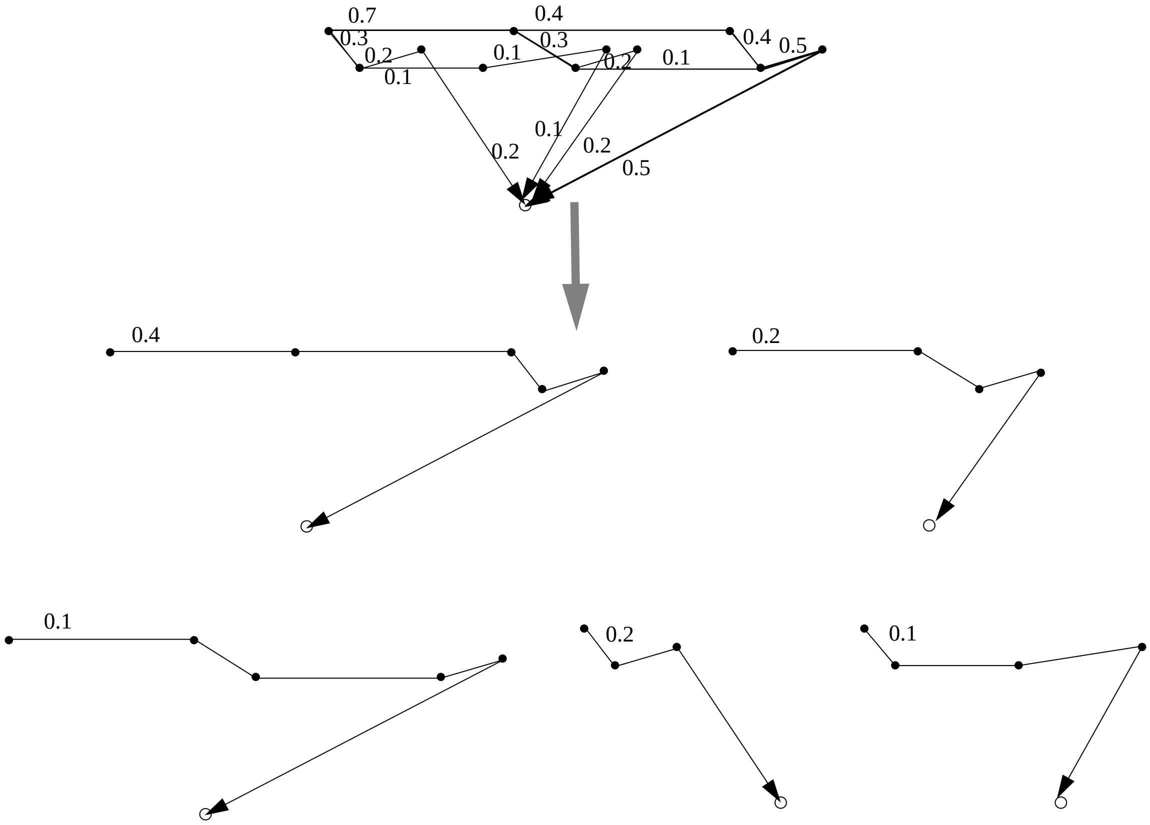


Figure 10: Path Probabilities Assigned after Path Stripping for Package p_2

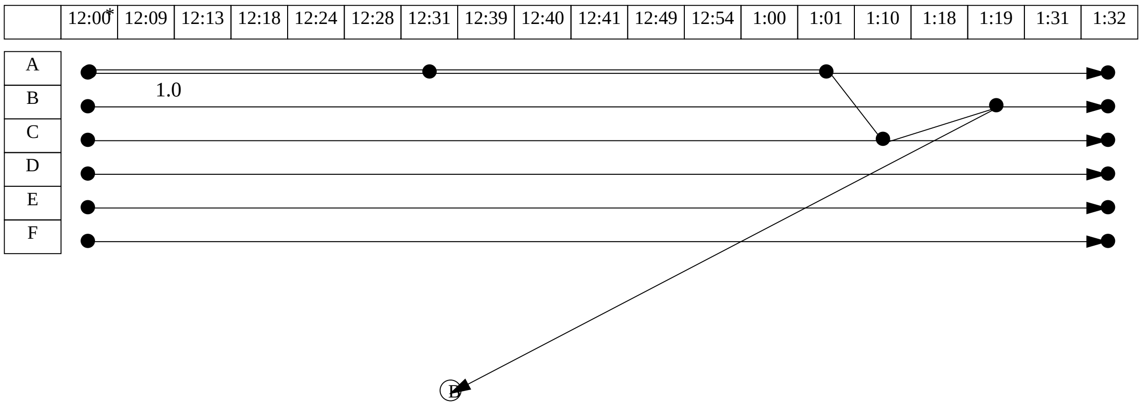


Figure 11: One Possible Path Chosen for Package p_2 after Randomized Rounding

For our example, the results of path stripping for package p_2 are given in Figure 10. From these results one possible selection from the randomized rounding phase is the path given in Figure 11.

Each phase of the randomized rounding procedure can be performed in polynomial time. Therefore, the entire procedure can be performed in polynomial time. Thus, we have approximated an NP-complete problem, multicommodity flow (greyhound problem), with a polynomial time algorithm. It remains to see how much the quality of the solution of this approximation algorithm differs from the optimal solution. Section 6.4 provides provable feasibility and expected performance guarantees for the randomized rounding procedure.

6.4 Provable Feasibility and Expected Performance Guarantees

The randomized rounding procedure produces an approximate integral solution to the greyhound problem based on the optimal fractional solution. Although we have approximated an NP-complete problem with a polynomial time algorithm, we have not shown that the resulting solution possesses any desired qualities. In fact, we have not shown that the integral solution is even feasible (*i.e.,* does not violate any constraints), much less near-optimal. In this section, we provide provable feasibility and expected performance guarantees for the randomized rounding procedure.

We demonstrate the quality of the randomized rounding solution in two parts. First, we explore the solution in view of the constraints. This provides our feasibility results. The path stripping and randomized rounding phases guarantee that the conservation of flow and requirement delivery constraints are met. To see this, notice that path stripping defines a set of paths for each package, with each path beginning at the source and ending at the sink for that package and each edge along a path carrying identical flow. Therefore, every one of these paths satisfies the conservation of flow constraints. Also, the randomized rounding phase selects exactly one of these paths to transport the entire requirement, thus, the requirement delivery constraint is met. However, the satisfaction of the capacity constraints is not as straight-forward and is explored in detail below. Second, we explore the solution in view of the performance criterion. This provides our expected performance results. In particular, we relate the value of the integral solution to the optimal fractional solution.

In order explore the feasibility of the integral solution (*i.e.,* capacity constraints are not violated), given a feasible fractional solution, we introduce the general analysis technique of Raghavan and Thompson [9, 8]. In summary, their technique employs a Chernoff-type bound to bound how far the flow assigned to each edge (load) can exceed the edge capacity, for any edge, in the integral solution. Unfortunately, as is discussed later in this section, this technique cannot be applied to bounding how far the value of the integral solution (performance) can vary from the optimal fractional solution.

Let $a_1, a_2, \ldots, a_r$ be reals in $(0, 1]$. Let $X_1, X_2, \ldots, X_r$ be independent Bernoulli trials with $E[X_j] = p_j$. Look at the random variable $\Psi = \sum_{j=1}^{r} a_j X_j$.

Theorem 2 *[8] Let $\delta > 0$, and $m = E[\Psi] = \sum_{j=1}^{r} a_j p_j > 0$. Then*

$$\Pr[\Psi > (1+\delta)m] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^m$$

This result roughly states that the probability that a weighted sum of independent Bernoulli trials exceeds the weighted sum of the expected values of the same trials by more $(1+\delta)$ times (probability of a *bad* event or, in other words, a *failure*), decreases with increasing δ. If we know the expected values of the independent Bernoulli trials, then, by choosing the appropriate δ, we can bound the value of the weighted sum of those trials with high probability. In a sense, δ bounds the penalty we pay (deviation from feasible solution)

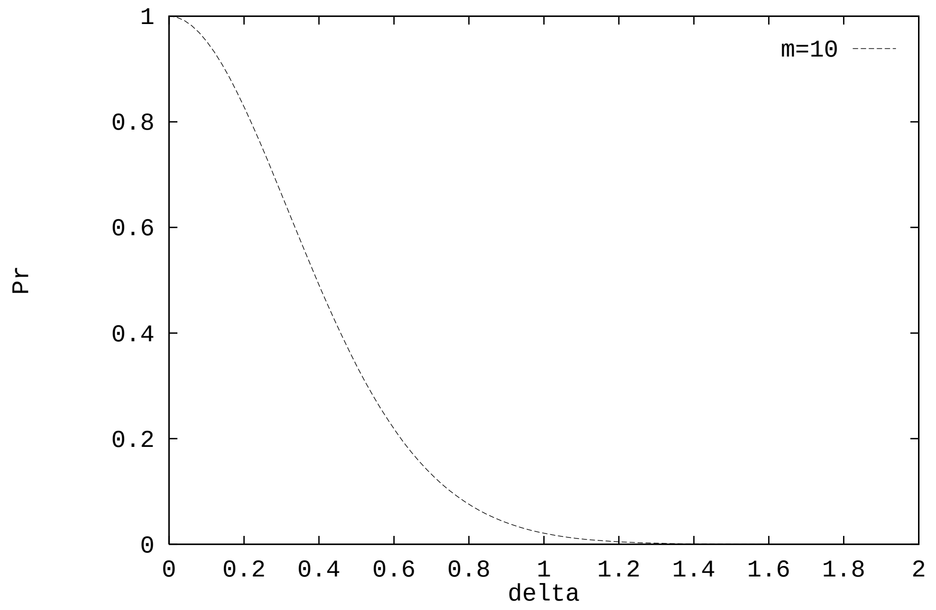


Figure 12: Probability versus δ for fixed mean $m = 10$, $[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}]^{10}$

by enforcing an integer solution. The more confidence (higher probability) we need that we won't pay an unexpectedly high penalty, the weaker our feasibility bound.

The character of the bound of Theorem 2 can be seen by the graph in Figure 12. This graph plots probability of a bad event versus δ for a fixed mean $m = 10$. It can be seen that as we relax our bounds (allow δ to become greater) the probability that the bounds will fail (a *bad* event) is reduced. On the other hand, if we insist on tight bounds (small δ) then the probability that a bad event will occur increases (similarly, the probability that only good events will occur, $1 - \mathrm{Pr}$, decreases).

This result applies to the randomized rounding technique as follows. After the path stripping phase, for any given package p, we have a set of paths Γ_p through which the flow of that package may be assigned. The expected fraction of flow from this package assigned to any given one of those paths $\gamma \in \Gamma_p$, after the randomized rounding phase, is equal to the probability assigned to that path $\mathrm{Pr}\{\gamma\}$ during path stripping. For any given package and edge within the flow graph, the expected fraction of flow from this package assigned to that edge is the sum of the expected fraction of flows assigned to each path $\gamma \in \Gamma_p$ that includes the edge (which is the same as the flow assigned to that edge before path stripping); *i.e.,* for each $p \in P$ and $e \in E$, $E[X_{p,e}] = \sum_{\{\gamma \in \Gamma_p | e \in \gamma\}} \Pr\{\gamma\} = \hat{X}_{p,e}$.

For each package $p \in P$ a flow path for that package is chosen independent of any other package. Each of these decisions can be considered an independent Bernoulli trial. By the above argument, we know the expected flow values for each edge in the flow graph resulting from each of these trials. Therefore, we can bound any weighted sum of these trials using Theorem 2 and the appropriate δ. We use this fact to bound the capacity constraints.

If the relaxed linear program is infeasible then the integral problem is infeasible. However, even if the fractional problem is feasible, the integral problem may still be infeasible. We use the tools above to analyze this situation in reference to capacity constraints. Since the relaxed linear programming solution is optimal, we know that all the constraints are satisfied in that solution. Consider any edge $e \in E$ in the flow graph. From the linear program we know that $\sum_{p \in P} R_p \hat{X}_{p,e} \leq cap_e$ (the fractional load assigned to edge e does not exceed the capacity of the edge). From the above arguments, we can see that, for each $p \in P$ $E[X_{p,e}] = \hat{X}_{p,e}$. Therefore, $E[\sum_{p \in P} R_p X_{p,e}] = \sum_{p \in P} R_p \hat{X}_{p,e} \leq cap_e$ (the expected load assigned to edge e by the integral solution does not exceed the capacity of the edge). Let $\Psi = \sum_{p \in P} R_p X_{p,e}$ (actual load assigned to edge e by the integral solution); by Theorem 2,

$$\Pr[\Psi > (1+\delta)cap_e] < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{cap_e}.$$

Therefore, the actual load assigned to any edge by the randomized rounding solution does not exceed the capacity of the edge by more than $(1 + \delta)$ times, with high probability, for selected δ. In Section 7, we show a variation of randomized rounding that guarantees that the integral load will not exceed the capacity, deterministically, with some sacrifice in performance.

The above arguments show that for any edge, the expected load is the same as the optimal fractional load and, therefore, does not exceed the capacity of the edge. In addition, we have shown that, if the actual load does exceed the capacity, this violation is provably bounded from above, with high-probability. While we can make these strong statements about the feasibility of the integral solution obtained by randomized rounding, we can only extend some of these claims to the performance of the algorithm. In particular, we show that the expected performance of the integral solution is the same as the optimal performance; however, we cannot provide tight bounds on the actual performance.

We know that the solution to the relaxed linear program gives us an optimal value for our performance criterion $\hat{Z}$. As before, for each $p \in P$ $E[X_{p,e}] = \hat{X}_{p,e}$. Therefore, $E[Z] = \sum_{p \in P} \sum_{e \in E} cost_e \hat{X}_{p,e} = \hat{Z}$ (the expected total cost of the integral solution equals the optimal value of the fractional solution).

Previously, we pointed out the properties needed in order to apply Theorem 2 to bound a quantity. Included is that the quantity involved be the sum of independent Bernoulli trials. We now show that this is not the case for the performance criterion. In our multicommodity flow formulation (and the original greyhound problem) the costs in the performance calculation are associated with each edge. Namely, the algorithm is assessed the cost of an edge once for each package that traverses that edge. As discussed above, for any edge $e \in E$ selecting whether a particular package $p \in P$ will traverse that edge is independent of any other package. Therefore, we can treat the total cost associated with any one edge, over all packages, as a weighted sum of independent Bernoulli trials. However, the performance criterion does not calculate the cost of a single edge, rather, it calculates the total cost over all edges and packages. While selecting whether a particular package will traverse a particular edge is independent of all other packages, it is not independent of all other edges. Therefore, the performance criterion is not a sum of independent Bernoulli trials, and we cannot use Theorem 2 to bound the performance of the integral solution in terms of the optimal fractional solution. In Section 8 we touch upon variations of analysis techniques similar to Theorem 2, as well as other possibilities, for overcoming this problem.

In Figure 12, we demonstrate the bounds for a fixed mean. The quality of the bounds changes as the mean changes. In Figure 13 it can be seen that as the mean increases the quality of the bound increases. For example, for a low mean such as $m = 0.1$ in Figure 13.i., in order to get a high-probability bound such as $\Pr\{\text{bad event}\} = 0.2$, we can only guarantee that the actual integer load will be within $1 + 10$ times the capacity of the edge. If we desire that the actual load be within $1 + 1$ times the capacity of the edge, we can only guarantee this with $\Pr\{\text{bad event}\} = 0.9$. For large means the result is just the opposite. In Figure 13.vi., we see that for $m = 10000$ we can guarantee with probability $1 - 0.2$ that the actual load will be within $1 + 0.02$ times the capacity. This is a much stronger result. This result is consistent with intuition; rounding has a larger effect on smaller values than on larger values. Thus, the quality of this bound is dependent upon the scale of the capacities in the particular greyhound instance being studied.

The right hand side of Theorem 2 can be a little difficult to deal with. Especially when trying to determine questions such as "how large need δ be for $\Pr[\Psi > (1+\delta)m]$ to be less than x?" For a desired probability bound, Raghavan [8] provides a technique for estimating the deviation δ to use in Theorem 2. Namely, denote $\Pr[\Psi > (1+\delta)m]$ by $B(m,\delta)$. Then, given any m and desired probability x we can derived the required $\delta = D(m,x)$ such that $B(m, D(m,x)) = x$ by the following equations:

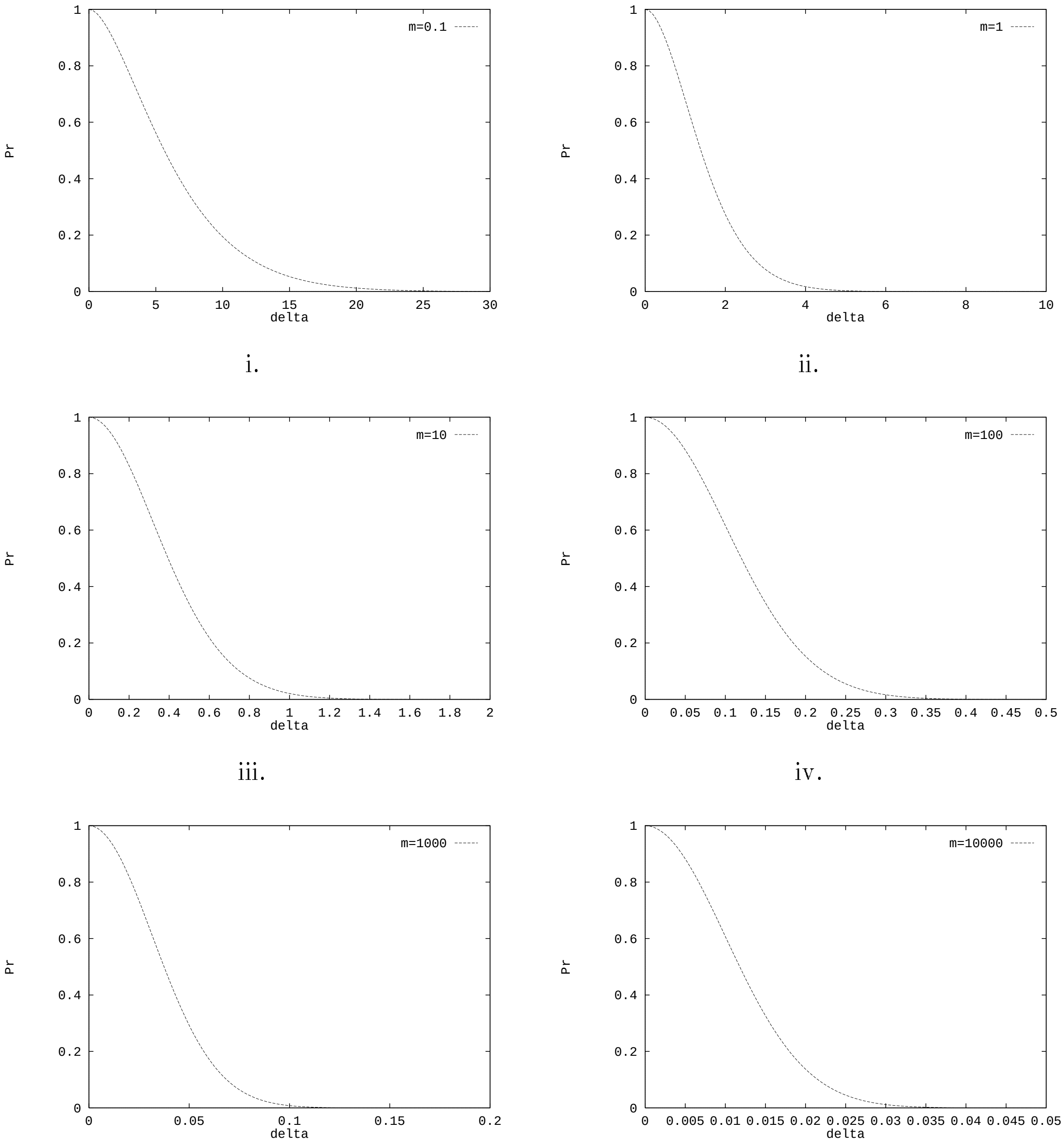
$m > \ln(1/x)\ \ D(m,x) \leq (e-1)[\frac{\ln(1/x)}{m}]^{1/2}$

$m \leq \ln(1/x)\ \ D(m,x) \leq \frac{e\ln(1/x)}{m\ln[(e\ln(1/x))/m]}$

For instance, if $m = 1000$ and we desire a solution with probability at least 0.9 (*i.e.,* solution exceeds mean by more than $1+\delta$ with probability $x = 0.1$), we can find a satisfactory $\delta = D(1000, 0.1) = 0.082$; indicating that load will be within 8.2% of the capacity (expected load) with probability ≥ 0.9. Similarly, for $m = 1000$ and desired probability at least 0.99, $\delta = D(1000, 1/100) = 0.12$, indicating that load will be within 12% of the capacity with probability ≥ 0.99. Finally, for $m = 1$ and desired probability 0.99, $\delta = D(1, 1/100) = 4.95$, indicating that load will be within 495% of the capacity with probability ≥ 0.99. These examples indicate that this technique provides much better bounds when the mean is greater. These estimates are consistent with the exact curves given in Figure 13.

7 Variations and Limitations of Applying Randomized Rounding to the Greyhound Problem

In Section 6, we introduce the randomized rounding technique of Raghavan and Thompson and apply it to the greyhound problem. We show that the expected load on an edge in the integral solution is less than or equal to the capacity of the edge and that the expected performance of the integral solution is equal to the optimal fractional performance. We also provide the analysis techniques that allow us to bound the feasibility of the solution provided by the approximation algorithm on instances of the greyhound problem, but do not allow us to bound the actual performance of the integral solution. This is a significant limitation on the use of Theorem 2. We also mentioned that larger expected loads on edges



i. ii.

iii. iv.

v. vi.

Figure 13: Probability versus δ for fixed means: i. $m = 0.1$, ii. $m = 1$, iii. $m = 10$, iv. $m = 100$, v. $m = 1000$, vi. $m = 10000$

provide better bounds. In this section, we explore additional limitations of this approach. In particular, we show that some restricted instances of the greyhound problem can obtain provably tighter bounds than others. In conjunction with the restrictions on instances we also discuss variations of the randomized rounding technique that address other issues.

We discuss three classes of approximation algorithms for the greyhound problem, based on randomized rounding. These classes differ in the type of bounds provided:

1. High-Probability Bounds,
2. Deterministic Bounds within Guaranteed Range, and
3. Deterministic Upper Bounds.

Each successive bounds introduces further limitations and restrictions on the solvable instances of the greyhound problem.

7.1 Limitations of High-Probability Bound Randomized Rounding Algorithm

The basic randomized rounding algorithm presented in Section 6 is an approximation algorithm with high-probability bounds. In this section, we explore the limitations of this algorithm in more detail. The first limitation of the basic algorithm is that the high-probability bounds imply that there is a non-zero probability that the guarantees will not hold for a particular integer solution. Although the probability of a *bad* event (*i.e.,* exceeding the capacity of an edge by more than δ times) is very low, its non-zero probability can be an unfortunate limitation for the greyhound problem. One way around this is to make repeated runs of the algorithm until a solution that satisfies the bounds is achieved. With high-probability, a solution in which no edge capacities are exceeded by more than δ times will be found, but, again, this technique is only probabilistic. In Section 7.2, we discuss a deterministic variation to solve this problem.

A second limitation, both of the basic algorithm and the deterministic algorithm of Section 7.2, is that the load assigned to an edge by the integer solution is only guaranteed to be within a certain δ factor of the capacity of the edge. This may not be sufficient for the greyhound problem. In Section 7.3, we discuss a deterministic variation to overcome this deficiency.

Other limitations are derived from the application of Theorem 2 to the greyhound problem. As we pointed out in Section 6.4, this theorem only applies to the weighted sum of independent Bernoulli trials. In particular, it doesn't apply to dependent sums like the performance criterion. Also, larger expected loads (*i.e.,* capacities) on edges provide better bounds. Additional limitations concern the weights. The theorem can only be applied when weights are reals in the interval $(0, 1]$. Since the weighted sum that this theorem is applied to is $\sum_{p \in P} R_p X_{p,e}$, we are restricted to instances in which package requirements (R_p) are in the interval $(0, 1]$. In many problems this limitation may not be serious but, in the greyhound problem this is a severe limitation. There is a naive solution to getting around this limitation. If we can bound the maximum package requirement in advance we can normalize all the values to this interval. Let $R_{\max}$ be the largest package requirement. The package requirements for the normalized problem would be $R'_p = R_p / R_{\max}$. In addition, we would also be forced to normalize the edge capacities by the maximum package requirement as well, $cap'_e = cap_e / R_{\max}$. However, as discussed in Section 6.4, smaller capacities lead to weaker bounds. Furthermore, as discussed in Section 7.3, further extensions of the randomized rounding technique require that the minimum edge capacity be greater than $\ln n$, where n is the number of edges. Therefore, this naive solution introduces additional limitations.

7.2 Deterministic Variation of Randomized Rounding, Providing Bounds within Guaranteed Range

As mentioned in Section 7.1, high-probability bounds imply that there is a non-zero probability that the guarantees will not hold for a particular integer solution. In other words, although Theorem 2 guarantees with high-probability that the load assigned to any edge will not exceed the edge's capacity by more that $1 + \delta$, there may be a rare case with very low, but non-zero, probability in which the load assigned to an edge by a particular integer solution exceeds this guarantee. For the greyhound problem, this is an important limitation. Luckily, Raghavan [8] provides a deterministic method for guaranteeing that the integer solution will not exceed the fractional solution by more than the guaranteed bound.

This method can be outlined as follows. Let n be the number of edge constraints (edges). Let δ_i be the bound provided for edge i, $1 \leq i \leq n$, given a particular probability goal. If we can guarantee that for any edge the probability that a *bad* event occurs on that edge (the integer load exceeds the edge capacity by more than the guaranteed δ_i bound) is less than $1/n$, then we are guaranteed that there exists with non-zero probability an integer solution in which good events occur on all edges, simultaneously. Specifically, for the i^{th} bad event $B(m_i, D(m_i, 1/n)) < 1/n$. Since there are n of these events the probability that the integer solution violates any of these bounds is < 1. Thus, it is possible to find a good solution. Once we have guaranteed that this integer solution exists, we deterministically find that solution in polynomial time. Raghavan first shows how to derive the δ_i such that such a solution exists, then provides a deterministic method called *pessimistic estimators* for finding that solution. It is important to note that the method of pessimistic estimators can be computed efficiently.

One problem with this solution is that, since we must guarantee that no edge load will exceed its capacity by more than a factor of $1 + \delta_i$ with probability $< 1/n$, we are further restricting the problem instances in which a satisfactory δ_i can be found. For example consider an edge i in which the expected load (capacity) is $m = 10$. In Figure 12, we see that we can limit the probability of a bad event to less than 0.2 by choosing a $\delta_i = 0.6$. Thus, we can get a fairly tight bound. Now, suppose there are $n = 10$ edges, all of capacity $m = 10$. Then, we must limit the probability to less than $1/10$ and get a δ_i no better than 0.7. The more edges we add, the weaker the bound becomes. This problem is worse for edges with smaller capacities. For example, in Figure 13.i., we see that if our problem contains 10 edges, one of which has capacity 0.1, the best overall bound we can provide is $\delta = 12$ (where $\delta = \max_{1 \leq i \leq n} \delta_i$).

With this method the load assigned to any edge is guaranteed to be within $1 + \delta$ times the capacity, deterministically. Not with high-probability. However, this may not be good enough for the greyhound problem. It may be the case that the only acceptable solution is for $\delta = 0$. A way to achieve this result is given in Section 7.3.

7.3 Deterministic Variation of Randomized Rounding, Providing Guaranteed Upper Bounds

The solutions provided in Sections 7.1 and 7.2 guarantee that the load assigned to any edge is within $1+\delta$ times the capacity of the edge, either with high-probability or deterministically. In other words, the solutions do not guarantee that no load will exceed the capacity of its edge. Raghavan and Thompson [8, 9] provide a solution for this based on *scaling*. The technique involves finding a scaling factor $v \in (0,1)$ such that $B(vk, \frac{1-v}{v}) < 1/n$, where k is the minimum edge capacity and n is the number of edges. This scaling factor is then multiplied by each fractional flow before rounding. By doing so, the fractional value of the loads on edges is scaled down by a factor of v. Let's look at how this effects our bounds. It is clear that the expected value of each load after rounding is $\leq v \times cap_e$. Whereas in the unscaled version we are bounding the probability that $\Psi > (1+\delta)cap_e$, in this scaled version we are bounding the probability that $\Psi > (1+\frac{1-v}{v})v \times cap_e$. Working this out we get $\Psi > cap_e$. Thus, our bad event now occurs when the actual load exceeds the capacity, rather than some multiple of the capacity. As in the pessimistic estimators method of Section 7.2, the choice of v ensures that each bad event occurs with probability $< 1/n$. So there exists an integer solution that does not exceed any capacities and it can be found deterministically as in Section 7.2.

One limitation of this technique is that such a scaling factor can be found only when $k > \ln n$. In other words, the minimum capacity of any edge must be greater than the natural log of the number of edges. Note that v is chosen so that the minimum capacity of any edge, k, satisfies the desired probability bound, $1/n$. By doing so, the condition is simultaneously satisfied for all the edges. Note also that as the number of edges, n, becomes larger (with minimum capacity remaining fixed), the scaling factor v becomes smaller. Thus, the larger the problem the weaker the bound provided by the scaling technique. Finally, the larger the minimum capacity, k, (with size of problem fixed) the tighter the bound.

In addition to the above limitations, there is a further tradeoff for the strict bounds provided by scaling. Since the flows are scaled before rounding, the flows for all the paths of a package no longer sum to one. This introduces the probability that the integer solution will not satisfy all package requirements. In the greyhound problem we not only need to guarantee that the capacities are not exceeded, but we also must guarantee that all packages are delivered (i.e. all requirements are met). Although the deterministic algorithm with scaling is guaranteed to find a solution which does not exceed any capacities, it does not guarantee that the solution will satisfy all requirements. Without scaling, all requirements are guaranteed to be met. It is clear that we cannot deterministically guarantee both capacities and requirements, simultaneously.

8 Discussion of Applicability of Randomized Rounding to the Greyhound Problem

In the above discussions we have presented three variations of the randomized rounding technique:

1. High-Probability Bounds,
2. Deterministic Bounds within Guaranteed Range, and
3. Deterministic Upper Bounds.

Notice that in each subsequent algorithm the obtainable bounds become weaker and additional limitations creep in. The following are characteristics of restricted greyhound problem instances which weaken the bounds in the above algorithms:

- small mean fractional load (capacity) on edges
- scaling of package requirements into the interval $(0, 1]$
- for deterministic bounds, larger problem sizes (number of edges)
- for upper bounds, smaller minimum capacities (smaller scaling factor)

An equally important limitation of the randomized rounding technique is the inability to use the analyses of Theorem 2 to bound the performance of the integer solution. There is some hope to overcoming this deficiency. Although the variables in the performance criterion are not independent, their dependency graph has a particular structure that may be exploited. An open question involves how to exploit this structure in the analyses. Promising research along these lines includes the work of Berger and Rompel [1] and Spencer [11].

9 Conclusion

This paper provides insight into the structure of restricted transportation problems and their solution through approximation algorithms. By exploring the greyhound problem in detail we develop intuition about the difficulties involved in solving problems within the transportation domain. The greyhound problem is shown to be NP-complete and, therefore, amenable to approximations as opposed to exact solutions. In this paper, we explore the use of approximation techniques based on solving integer programs through first solving the relaxed linear programs and then applying rounding techniques. We also demonstrate one procedure for converting randomized algorithms into deterministic algorithms. These techniques provide positive basic results but, the limitations cast doubts on its application to practical transportation problems. Further work is anticipated both in modifying the approach to remove some of the limitations and applying similar approaches to practical problems to extend our intuitions.

Acknowledgement

Work on the multicommodity flow formulation was done jointly with Philip Klein. The authors would also like to thank Philip Klein, R. Ravi, Moises Lejter and Shieu-Hong Lin for valuable discussions and feedback on an earlier draft of this paper.

References

[1] Berger, Bonnie and Rompel, John, Simulating ($\log^c n$)-wise Independence in NC, *Proceedings of the Thirtieth Annual IEEE Symposium on Foundations of Computer Science*, 1989, 1–7.

[2] Bixby, R., Gregory, J., Lustig, I., Marsten, R., and Shanno, D., *Very Large-Scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods*, Technical Report 34-91, Rutcor Research Report, New Brunswick, NJ, June 1991.

[3] Cook, S. A., The complexity of theorem-proving procedures, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, New York, 1971, 151–158.

[4] Dean, Thomas and Greenwald, Lloyd, *A Formal Description of the Transportation Problem*, Technical Report CS-92-14, Brown University Department of Computer Science, Providence, RI, March 1992.

[5] Even, S., Itai, A., and Shamir, A., On the Complexity of Timetable and Multi-Commodity Flow Problems, *SIAM J. Comput.*, **5**(4) (1976) 691–703.

[6] Garey, Michael R. and Johnson, David S., *Computers and Intractibility: A Guide to the Theory of NP-Completeness*, (W. H. Freeman and Company, New York, 1979).

[7] Karmarkar, N., A new polynomial-time algorithm for linear programming, *Combinatorica*, **4** (1984) 373–396.

[8] Raghavan, Prabhakar, Probabilistic Construction of Deterministic Algprithms: Approximating Packing Integer Programs, *Computer and Systems Sciences*, **37** (1988) 130–143.

[9] Raghavan, Prabhakar and Thompson, Clark, Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs, *Combinatorica*, **7**(4) (1987) 365–374.

[10] Schank, John, Mattock, Michael, Sumner, Gerald, Greenberg, Irwin, Rothenberg, Jeff, and Stucker, James, *A Review of Strategic Mobility Models and Analysis*, Technical Report R-3926-JS, Rand National Defense Research Institute, 1991.

[11] Spencer, Joel, *Ten Lectures on the Probabilistic Method*, (SIAM, Philadelphia, 1987).