Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems

W. Matthew Carlyle, Johannes O. Royset, R. Kevin Wood
Operations Research Department, Naval Postgraduate School, Monterey, California

The constrained shortest-path problem (CSPP) generalizes the standard shortest-path problem by adding one or more path-weight side constraints. We present a new algorithm for CSPP that Lagrangianizes those constraints, optimizes the resulting Lagrangian function, identifies a feasible solution, and then closes any optimality gap by enumerating near-shortest paths, measured with respect to the Lagrangianized length. "Near-shortest" implies ϵ-optimal, with a varying ϵ that equals the current optimality gap. The algorithm exploits a variety of techniques: a new path-enumeration method; aggregated constraints; preprocessing to eliminate edges that cannot form part of an optimal solution; "reprocessing" that reapplies preprocessing steps as improved solutions are found; and, when needed, a "phase-I procedure" to identify a feasible solution before searching for an optimal one.

The new algorithm is often an order of magnitude faster than a state-of-the-art label-setting algorithm on singly constrained randomly generated grid networks. On multiconstrained grid networks, road networks, and networks for aircraft routing the advantage varies but, overall, the new algorithm is competitive with the label-setting algorithm. **© 2008 Wiley Periodicals, Inc.* NETWORKS, Vol. 00(0), 000–000 2008**

Keywords: constrained shortest paths; near-shortest paths; Lagrangian relaxation; path enumeration; label-setting algorithm

1. INTRODUCTION

Algorithms for shortest-path problems in networks with non-negative edge lengths (or with some negative-length edges, but no negative-length cycles) are both practically and theoretically efficient [1]. But, if each edge possesses a non-negative weight in addition to its length, and just a single side constraint is placed on the optimal path's total weight, the problem becomes NP-complete [14]. The general problem, with an arbitrary number of side constraints, is known as the constrained shortest-path problem (CSPP). This article develops a new algorithm for solving this problem, applies it to several classes of network models, and compares its computational efficiency to a state-of-the-art alternative, the label-setting algorithm of Dumitrescu and Boland [10].

CSPP is NP-complete in the weak sense for a fixed number of side constraints and admits a dynamic-programming solution procedure [18]. Dynamic programming (DP) can be unacceptably slow in practice, however, even with a single side constraint. Consequently, label-setting algorithms based on DP have supplanted straightforward DP implementations [2, 8, 10]. Other potentially useful techniques include branch and bound using a Lagrangian-based bound [3], Lagrangian relaxation coupled with K-shortest-paths enumeration [16, 19], search methods employing K-shortest-paths enumeration [7, 34, 35], A* search [27], and heuristic algorithms [21, 22]; see also the review by Van Mieghem et al. [36].

CSPP arises in a number of real-world contexts. One well-known application is column-generation for generalized set-partitioning models of crew-scheduling and crew-rostering problems, especially in the airline industry (e.g., [6, 13, 33]). Other important applications include minimum-risk routing of military vehicles and aircraft (e.g., [4, 25, 26, 38]), signal routing in communications networks having quality-of-service guarantees (see [36] and the references therein), signal compression [30] and numerous transportation problems (e.g., [20, 29]).

Dumitrescu and Boland [10] describe a label-setting algorithm, combined with several preprocessing techniques, that may be the most efficient technique currently available for CSPP. Dumitrescu and Boland compare the performance of their "enhanced" label-setting algorithm with several different label-setting algorithms as well as three other algorithms on six classes of CSPPs. To the authors' knowledge, that paper provides the most extensive computational study available in the recent literature, and analyzes the largest problems. (Other recent computational studies include Korkmaz et al. [23], Kuipers et al. [24], Li et al. [27] and Ziegelmann [39].) Consequently, we use our implementation of Dumitrescu and Boland's enhanced label-setting algorithm as the benchmark for computational studies in this paper.

Received July 2005; accepted December 2007
Correspondence to: R. K. Wood; e-mail: kwood@nps.edu
DOI 10.1002/net.20247
Published online in Wiley InterScience (www.interscience.wiley.com).

We present an alternative approach, which we call Lagrangian relaxation with near-shortest-paths enumeration (LRE). This approach Lagrangianizes the side constraints, optimizes the Lagrangian function, identifies a feasible solution (often while optimizing the Lagrangian function), and closes any optimality gap by enumerating near-shortest paths. Path length is measured with respect to the Lagrangianized edge lengths, and "near shortest" means ϵ-optimal, with ϵ set to the current value of the optimality gap.

LRE resembles the CSP algorithm of Handler and Zang [16], but with a near-shortest-paths (NSP) algorithm replacing their K-shortest-paths algorithm. The LRE approach seems particularly attractive because we [5] have recently developed an extremely fast near-shortest-paths algorithm, and "near-shortest paths" proves to be a more appropriate paradigm than "K shortest paths" for the enumerative phase of the LRE algorithm. We discuss this issue in more detail later. LRE also resembles the branch-and-bound algorithm of Beasley and Christofides [3], but LRE does not reoptimize the Lagrangian lower bound at each node of the branch-and-bound enumeration tree.

Our LRE algorithm also exploits preprocessing, as in Ref. 10, to eliminate edges that can be proven, a priori, to lie on no optimal path. However, we add a number of aggregate constraints to improve the efficiency of that preprocessing, as well as to reduce subsequent enumeration effort. We also describe an auxiliary "phase-I procedure" for finding a feasible solution when none is identified while optimizing the Lagrangian function. This feasibility problem is an NP-complete problem in and of itself when multiple side constraints are involved [14].

In addition to the theoretical development, we present a computational study of the LRE algorithm applied to CSPPs, defined on artificial and real-world networks, with between one and ten side constraints. This study includes a direct comparison to our implementation of the label-setting algorithm from [10].

The remainder of the article begins by defining CSPP precisely, and by then describing the basic LRE solution approach. We then provide an overview of the NSP algorithm that makes the LRE algorithm viable. (The appendix contains the pseudo-code for this procedure. We include this for completeness because our application of NSP requires features not presented in the original development in [5].) We do not discuss optimizing the Lagrangian function in any detail, because the relevant techniques are well known. We do refine the basic LRE approach by adding aggregate constraints to the preprocessing procedure and to the NSP algorithm, as well as incorporating a phase-I procedure for finding initial feasible solutions when none is apparent. Finally, we present a computational study.

2. PROBLEM DEFINITION AND BASIC APPROACH

Let $G = (V, E)$ be a *directed graph* with vertex set V and edge set E. Each (directed) *edge* $e = (u, v) \in E$ connects distinct *vertices* $u, v \in V$, and it possesses *length* $c_e \geq 0$ and one or more *weights* $f_{ie} \geq 0$ for $i \in I$, where I indexes a set of *side constraints*. Each side constraint i has a *weight limit* $g_i \geq 0$ defined.

A *directed s-t path* E_P is an ordered set of edges of the form $E_P = \{(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, t)\}$. The path is *simple* if no vertices are repeated. Given two distinct vertices $s, t \in V$, the *constrained shortest-path problem* (CSPP) seeks a directed, simple, s-t path E_P such that $\sum_{e \in E_P} f_{ie} \leq g_i$ for all $i \in I$, and such that $\sum_{e \in E_P} c_e$ is minimized.

Let A denote the $|V| \times |E|$ vertex-edge incidence matrix for G such that if $e = (u, v)$, then $A_{ue} = 1$, $A_{ve} = -1$, and $A_{we} = 0$ for any $w \neq u, v$. Also, let $b_s = 1$, $b_t = -1$ and $b_v = 0$ for all $v \in V \setminus \{s, t\}$, and let $\mathbf{g}$ denote the vector $(g_1, g_2, \ldots, g_{|I|})^T$. For each $i \in I$, we collect the edge weights f_{ie}, $e \in E$, in the row vector $\mathbf{f}_i$. Finally, we define F as the $|I| \times |E|$-matrix having vectors $\mathbf{f}_i$ as its rows. CSPP may then be written as this integer program [1]:

$$\textbf{CSPIP} \quad z^* \equiv \min_{\mathbf{x}} \quad \mathbf{cx} \tag{1}$$

$$\text{s.t.} \quad A\mathbf{x} = \mathbf{b} \tag{2}$$

$$F\mathbf{x} \leq \mathbf{g} \tag{3}$$

$$\mathbf{x} \geq \mathbf{0} \text{ and integer,} \tag{4}$$

where $x_e^* = 1$ if edge e is in the optimal path, and $x_e^* = 0$, otherwise. Note that the problem's structure leads to binary solutions without the explicit constraints $\mathbf{x} \leq \mathbf{1}$. Equations (3) are CSPP's *side constraints*. We refer to $\hat{\mathbf{x}}$ as a "path" when it satisfies all constraints of **CSPIP** except possibly the side constraints. Strictly speaking, a path $\hat{\mathbf{x}}$ could have $\hat{x}_e = 1$ for all edges e around one or more cycles. However, there always exists an optimal solution with no cycles, because we assume $\mathbf{c} \geq \mathbf{0}$ and $\mathbf{f}_i \geq \mathbf{0}$ for all i. Furthermore, our LRE algorithm cannot generate any solutions to CSPP that incorporate cycles, and thus this issue can be safely ignored.

CSPIP would define an easy-to-solve shortest-path problem if not for the side constraints. We expect to have only a few such constraints, say one to ten, and it therefore seems reasonable to base a solution procedure on relaxing them. Using the standard theory of Lagrangian relaxation [1], we know that for any appropriately dimensioned row vector $\boldsymbol{\lambda} \geq \mathbf{0}$,

$$z^* \geq \underline{z}(\boldsymbol{\lambda}) \equiv \min_{\mathbf{x}} \quad \mathbf{cx} + \boldsymbol{\lambda}(F\mathbf{x} - \mathbf{g}) \tag{5}$$

$$\text{s.t.} \quad A\mathbf{x} = \mathbf{b} \tag{6}$$

$$\mathbf{x} \geq \mathbf{0} \text{ and integer.} \tag{7}$$

We then rewrite the objective function, and optimize the Lagrangian lower bound $\underline{z}(\boldsymbol{\lambda})$ through

$$\textbf{CSPLR} \quad \underline{z}^* \equiv \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \underline{z}(\boldsymbol{\lambda}) \tag{8}$$

$$= \max_{\boldsymbol{\lambda} \geq \mathbf{0}} \min_{\mathbf{x}} (\mathbf{c} + \boldsymbol{\lambda}F)\mathbf{x} - \boldsymbol{\lambda}\mathbf{g} \tag{9}$$

$$\text{s.t.} \quad A\mathbf{x} = \mathbf{b} \tag{10}$$

$$\mathbf{x} \geq \mathbf{0} \text{ and integer.} \tag{11}$$

For any fixed $\boldsymbol{\lambda} \geq 0$, computing $\underline{z}(\boldsymbol{\lambda})$ simply requires the solution of a shortest-path problem with Lagrangian-modified edge lengths.

The solution to the linear-programming (LP) relaxation of the inner minimization of **CSPLR** is intrinsically integer, so we know that $\underline{z}^*$ equals the optimal objective value of the LP relaxation of **CSPIP** [11]. Unfortunately, it is easy to construct examples with large duality gaps, i.e., examples for which $\underline{z}^*$ greatly underestimates z^*. Thus, the success of the LRE approach will sometimes depend on the ability to close a large duality gap quickly.

The outer maximization over $\boldsymbol{\lambda}$ can be solved in several ways, depending on the dimension of $\boldsymbol{\lambda}$, i.e., depending on the number of side constraints. Beasley and Christofides [3] describe the use of subgradient optimization for CSPPs with up to ten side constraints. A constraint-generation algorithm analogous to Benders decomposition could also be used [37]. But, bisection search works well for a single side constraint [12], as does bisection search applied in the coordinate directions for a few side constraints [9]. That is the approach we use.

In the process of optimizing $\underline{z}(\boldsymbol{\lambda})$, we may find a path $\hat{\mathbf{x}}$ that is feasible with respect to the relaxed side constraints (3). Such a solution provides an upper bound for CSPP, $\bar{z} = \mathbf{c}\hat{\mathbf{x}} \geq z^*$. Indeed, if a feasible instance of **CSPIP** possesses only a single side constraint, then for sufficiently large $\boldsymbol{\lambda}$ every optimal solution of **CSPLR** satisfies (3). Unfortunately, as the number of side constraints grows, finding a feasible solution to **CSPIP** while optimizing $\underline{z}(\boldsymbol{\lambda})$ becomes less and less likely. To overcome this difficulty, we develop and apply the phase-I routine described in Section 4.3. Note that without this subroutine, a (weak) upper bound for a feasible **CSPIP** is always $\bar{z} = (|V| - 1)c_{\max}$ where $c_{\max} \equiv \max_{e \in E} c_e$.

Now, given $\bar{z}$ and a Lagrangian vector $\boldsymbol{\lambda} \geq \mathbf{0}$, the following theorem and corollary show that we may view the problem of identifying $\mathbf{x}^*$, an optimal solution to **CSPIP**, as one of simple path enumeration. (The theorem is implicit in Handler and Zang [16].)

Theorem 1. *For some $\boldsymbol{\lambda} \geq \mathbf{0}$, let $\hat{X}(\boldsymbol{\lambda}, \bar{z})$ denote the set of feasible solutions $\hat{\mathbf{x}}$ to* **CSPLR** *with the property that $\mathbf{c}\hat{\mathbf{x}} + \boldsymbol{\lambda}(F\hat{\mathbf{x}} - \mathbf{g}) \leq \bar{z}$. Then $\mathbf{x}^* \in \hat{X}(\boldsymbol{\lambda}, \bar{z})$.*

Proof. Since $F\mathbf{x}^* \leq \mathbf{g}$ and $\boldsymbol{\lambda} \geq \mathbf{0}$, the result follows from the facts that (i) $\mathbf{c}\mathbf{x}^* + \boldsymbol{\lambda}(F\mathbf{x}^* - \mathbf{g}) \leq z^*$, and (ii) $z^* \leq \bar{z}$. ∎

Corollary 1. *If* **CSPIP** *is feasible, an optimal solution $\mathbf{x}^*$ can be identified by setting $\boldsymbol{\lambda} \geq \mathbf{0}$, enumerating $\hat{X}(\boldsymbol{\lambda}, \bar{z})$, and by then selecting*

$$\mathbf{x}^* \in \underset{\mathbf{x} \in \hat{X}(\boldsymbol{\lambda}, \bar{z})}{\operatorname{argmin}} \{\mathbf{c}\mathbf{x} | F\mathbf{x} \leq \mathbf{g}\}. \tag{12}$$

∎

Theorem 1 and Corollary 1 are valid for any $\boldsymbol{\lambda} \geq \mathbf{0}$, but it is easy to devise examples that show how an optimal or near-optimal $\boldsymbol{\lambda}$ for **CSPLR** can reduce the size of $\hat{X}(\boldsymbol{\lambda}, \bar{z})$ exponentially, and reduce the computational workload correspondingly.

Theorem 1 and Corollary 1 imply that we may need to enumerate each path $\hat{\mathbf{x}}$ satisfying $(\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$. Therefore, given the edge-length vector $\mathbf{c} + \boldsymbol{\lambda}F$, and adding the Lagrangian constant term $-\boldsymbol{\lambda}\mathbf{g}$ to the length of any path, we wish to find all ϵ-optimal (near-shortest) paths for $\epsilon \equiv \bar{z} - \underline{z}(\boldsymbol{\lambda})$. Of course, as path enumeration proceeds, better feasible solutions may be found, $\bar{z}$ and thus ϵ will be reduced, and that may in turn reduce the necessary enumeration.

From the above discussion, it appears that an NSP algorithm, which identifies ϵ-optimal paths, is a natural choice for path enumeration in the LRE solution approach to CSPP. A typical K-shortest-paths (KSP) algorithm could be used as an alternative, however (e.g., [15]). Such an algorithm is meant to enumerate the K shortest paths in a network for a prespecified integer K. But, because it enumerates paths in order of increasing length, it could be halted when it finds a path $\hat{\mathbf{x}}$ such that $(\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \geq \bar{z}$. However, enumerating paths in order of length requires undue computational effort, storage and algorithmic complexity. The NSP algorithm developed by Carlyle and Wood [5] is much simpler and faster.

3. THE LRE ALGORITHM FOR CSPP

This section outlines the basic LRE algorithm for CSPP.

LRE Algorithm for CSPP (Outline)

1. Find $\boldsymbol{\lambda}$ that optimizes, or approximately optimizes, the Lagrangian lower bound $\underline{z}(\boldsymbol{\lambda})$.
2. Let $\hat{X}$ denote the set of feasible paths identified while optimizing $\underline{z}(\boldsymbol{\lambda})$. If $\hat{X} \neq \emptyset$, set the upper bound $\bar{z} = \min_{\hat{\mathbf{x}} \in \hat{X}} \mathbf{c}\hat{\mathbf{x}}$, else set $\bar{z} = (|V| - 1)c_{\max} + \gamma$ for some $\gamma > 0$.
3. Using the NSP algorithm from [5], begin enumerating all paths $\hat{\mathbf{x}}$ such that $(\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$, with the following modifications:
   a. Use $\bar{z}$ and the side constraints to limit the enumeration when it can be projected that the current path cannot be extended to one whose (true) length improves upon $\bar{z}$ or which does not violate a side constraint.
   b. Whenever the algorithm identifies a feasible path $\hat{\mathbf{x}}$ that is shorter than the incumbent, update the incumbent to $\hat{\mathbf{x}}$ and update the upper bound to $\bar{z} = \mathbf{c}\hat{\mathbf{x}}$.
4. If no $\hat{\mathbf{x}}$ is found in Step 3, the problem is infeasible; otherwise the best solution $\hat{\mathbf{x}}$ is optimal.

The NSP algorithm upon which we base this procedure (see the Appendix) begins by

1. Computing the minimum "Lagrangian distance" $d(v)$ from each $v \in V$ back to t by solving a single, backwards, shortest-path problem starting from t, using the Lagrangianized edge lengths $\mathbf{c}' \equiv \mathbf{c} + \boldsymbol{\lambda}F$,
2. Computing analogous minimum v-to-t distances $d_0(v)$ for all $v \in V$ with respect to the edge lengths $\mathbf{c}$, and
3. For each $i \in I$, computing analogous minimum v-to-t distances $d_i(v)$ for all $v \in V$ with respect to the edge weights $\mathbf{f}_i$.

This first phase requires the solution of only $|I| + 2$ shortest-path problems. We note that several other authors have proposed similar, backward shortest-path calculations within other solution approaches; for example, see [10, 22, 28].

Let $E_P(u) = \{(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, u)\}$ denote a *directed s-u subpath*. In the second phase of the algorithm, a depth-first path-enumeration procedure commences from s, but extends subpath $E_P(u)$ along edge $e = (u, v)$ if and only if the following conditions hold:

1. $E_P(u) \cup \{e\}$ can be extended to a path whose Lagrangianized length does not exceed $\bar{z}$, i.e., $L(u) + (c_e + \sum_{i \in I} \lambda_i f_{ie}) + d(v) \leq \bar{z}$, where $L(u)$ denotes the Lagrangianized length of $E_P(u)$ and where, by convention, we define $L(s) = -\boldsymbol{\lambda}\mathbf{g}$.
2. $E_P(u) \cup \{e\}$ can be extended to a path whose true length is strictly less than $\bar{z}$, i.e., $L_0(u) + c_e + d_0(v) < \bar{z}$, where $L_0(u)$ denotes the length of $E_P(u)$.
3. For all $i \in I$, $E_P(u) \cup \{e\}$ can be extended to a path whose i-th weight does not exceed g_i, i.e., $L_i(u) + f_{ie} + d_i(v) \leq g_i$, where $L_i(u)$ denotes the i-th total weight of $E_P(u)$.
4. The path does not loop back on itself.

Computer scientists will recognize this algorithm as a non-heuristic, depth-first version of "A* search" (e.g., [32]). We note that that Li et al. [27] solve small instances of CSPP using A* search, but use a "best-first" search strategy and do not exploit Lagrangian bounds. (See also [28], who use a variant on A* search to identify multiple feasible solutions to CSPPs.)

It is easy to see that the conditions above are necessary for existence of a feasible path better than $\bar{z}$, because (i) the label $d_0(v)$ represents a lower bound on the true length of any subpath from v to t that is required to complete the subpath $E_P(u) \cup \{e\}$, and (ii) because the labels $d(v)$ and $d_i(v)$ represent similar lower bounds for the Lagrangianized path length and the i-th path weight, respectively. Each label represents a lower bound from a relaxation of CSPP, rather than an exact value, because each is computed independently, and because the v-t subpath may include one or more vertices already on $E_P(u)$, which is disallowed in our version of CSPP.

The labels $d(v)$, $d_0(v)$, and $d_i(v)$ could be made sharper if, every time we extend or retract the current subpath, we recompute "shortest" paths, subject to the condition that no vertex currently on $E_P(u)$ is used. This could reduce enumeration. Indeed, when enumerating near-shortest paths with respect to a single distance measure, this recomputation ensures that only polynomial work need be expended for each path enumerated; otherwise, that work can be exponential [5].

On the other hand, solving the shortest-path problems required to maintain updated labels can add tremendously to the computational workload. This workload need not be as great as one shortest-path calculation for each type of label, for every extension or retraction of the current subpath, but it can still be prohibitive. In fact, Carlyle and Wood show empirically that, when enumerating near-shortest paths, an algorithm that does not recompute distances can be orders of magnitude faster than one that does. This holds true over a wide range of problem classes, even though the theoretical complexity is worse. Consequently, we do not recompute labels in the LRE algorithm as the current s-u subpath extends or contracts.

The reader will probably recognize that the LRE algorithm actually defines a branch-and-bound procedure that incorporates a depth-first enumeration mechanism along with feasibility checks. Branching consists of extending the current subpath by one edge. An LP-based algorithm would update the lower-bounding problem to account for the restriction imposed by a branch, and would then reoptimize the lower bound. LRE updates the bound, but does not reoptimize it. Reoptimization would require a new search over $\boldsymbol{\lambda}$, and the solution of numerous shortest-path problems which, as indicated above, is computationally onerous.

As with any branch-and-bound procedure, allowing a small but acceptable optimality gap in LRE can substantially reduce the enumeration required. The pseudo-code for the NSP algorithm, given in the Appendix, does include an "absolute-gap parameter," $\delta \geq 0$, for this purpose,

4. ALGORITHMIC ENHANCEMENTS

The basic LRE algorithm solves many problems quickly, as we shall see in Section 5. However, the three enhancements to the basic algorithm, described here, prove useful for solving more difficult problems.

4.1. Preprocessing

A preprocessing procedure for CSPP may be able to identify vertices and edges that cannot lie on any optimal path, and remove them prior to optimization. The resulting, smaller network should require less effort to solve, simply because fewer vertices and edges must be processed (e.g., [2, 10]). Importantly, a smaller network may also yield a tighter Lagrangian bound. We use the following preprocessing procedure originally proposed in [2]:

1. For all $i \in I$, and for all $v \in V$, compute a minimum-weight s-v subpath length $D_i(v)$ and a minimum-weight v-t subpath length $d_i(v)$ with respect to the weight vector $\mathbf{f}_i$.
2. Delete any edge $e = (u, v) \in E$ such that $D_i(u) + f_{ie} + d_i(v) > g_i$ for any $i \in I$.
3. Repeat Steps 1 and 2 until no new edges can be deleted.

A similar procedure for eliminating vertices can also be constructed [2, 10], but the edge-elimination procedure subsumes it. (Preprocessing first with respect to vertices and then with respect to edges could be more efficient, on average, than preprocessing with respect to edges alone. But, either way, the computational effort is negligible.)

By its construction, our NSP algorithm automatically performs many of the checks that a preprocessing procedure carries out. However, we find empirically that the preprocessing procedure described above does reduce computation

times. In an attempt to eliminate additional edges from the network, we can also preprocess with respect to the aggregate weight constraint

$$\boldsymbol{\pi} F\mathbf{x} \leq \boldsymbol{\pi}\mathbf{g} \tag{13}$$

for some row vector $\boldsymbol{\pi} \geq \mathbf{0}$ of dimension $|I|$. Because the aggregate constraint (13) considers all the weights for each edge along subpaths simultaneously, it has the potential to eliminate additional edges as the following example illustrates. Consider a three-vertex network with edges $a = (s, 2)$, $b = (2, t)$, and $c = (2, t)$; weights $f_{1a} = f_{2a} = f_{1b} = f_{2c} = 1$, $f_{1c} = f_{2b} = 2$; weight limits $g_1 = g_2 = 2$; and $\pi_1 = \pi_2 = 1$, so that the aggregate weight limit is 4. Edge a cannot be removed by tests based on weight limits g_1 or g_2 separately, because a low-weight 2-t subpath exists for each (i.e., $f_{1a} + \min\{f_{1b}, f_{1c}\} \leq g_1$, and $f_{2a} + \min\{f_{2b}, f_{2c}\} \leq g_2$). However, all 2-$t$ subpaths have an aggregate weight of 3; the aggregate weight for edge a is 2; $2 + 3 > 4$; and hence edge a can be deleted (i.e., $\pi_1 f_{1a} + \pi_2 f_{2a} + \min\{\pi_1 f_{1b} + \pi_2 f_{2b}, \pi_1 f_{1c} + \pi_2 f_{2c}\} > \pi_1 g_1 + \pi_2 g_2$). We note that this constraint-composition technique is related to "Jaffe's approximation" [17].

"LRE-P" will denote a version of the LRE algorithm that incorporates preprocessing Steps 1–3 with respect to individual side constraints and the single aggregate constraint based on $\boldsymbol{\pi} = \mathbf{1}$. If a preprocessing scan of all edges leads to the removal of at least one edge, it is possible that a subsequent scan may lead to further reductions. In practice, we let LRE-P repeat preprocessing scans until no reductions are identified, or until a maximum of 10 scans is reached.

If a feasible solution is found for CSPP, it yields an upper bound $\bar{z}$, and we can add the following constraints to the problem:

$$\mathbf{c}\mathbf{x} < \bar{z} \tag{14}$$

$$-\boldsymbol{\lambda}\mathbf{g} + (\mathbf{c} + \boldsymbol{\lambda} F) \leq \bar{z} \quad \text{for any } \boldsymbol{\lambda} \geq \mathbf{0}. \tag{15}$$

(Recall that we include the Lagrangian constant term $-\boldsymbol{\lambda}\mathbf{g}$ in the Lagrangian path length.) We do not normally preprocess with respect to these constraints, however, because their effect tends to be limited unless $\bar{z}$ is close to z^*. However, as demonstrated in Section 5.4, this extra preprocessing can be useful on difficult problems.

The following subsection describes a second use of aggregate constraints, to limit enumeration. To avoid confusion, the word "aggregate" will henceforth pertain to the second context, except where specifically noted.

4.2. Aggregate Constraints to Limit Enumeration

Once $\boldsymbol{\lambda}$ has been optimized, the path-enumeration portion of LRE repeatedly asks: Given that x_e must equal 1 for every edge e on the current s-u subpath, can this subpath be extended to a complete path $\mathbf{x}$ such that

$$\begin{array}{lrl} \text{[A]} & \mathbf{c}\mathbf{x} < \bar{z}, & \text{and} \\ \text{[B]} & -\boldsymbol{\lambda}\mathbf{g} + (\mathbf{c} + \boldsymbol{\lambda} F)\mathbf{x} \leq \bar{z}, & \text{and} \\ \text{[C]} & F\mathbf{x} \leq \mathbf{g}? & \end{array}$$

We do not extend the path along an edge, say $e' = (u, v)$, if setting $x_{e'} = 1$ would provably force $\mathbf{x}$ to violate any of the constraints [A], [B], and [C].

Aggregate versions of constraints [A], [B], and [C] may further limit path enumeration. Using empirically determined multipliers $\boldsymbol{\pi}_1 = (1/g_1, \ldots, 1/g_{|I|})$, $\pi_2 = 1$, and $\pi_3 = 1/\underline{z}(\boldsymbol{\lambda})$, we aggregate [C], each pair of [A], [B], and [C], and all three:

$$\boldsymbol{\pi}_1[\text{C}] \tag{16}$$

$$\pi_2[\text{A}] + \pi_2[\text{B}] \tag{17}$$

$$\pi_3[\text{A}] + \boldsymbol{\pi}_1[\text{C}] \tag{18}$$

$$\pi_3[\text{B}] + \boldsymbol{\pi}_1[\text{C}] \tag{19}$$

$$\pi_3[\text{A}] + \pi_3[\text{B}] + \boldsymbol{\pi}_1[\text{C}]. \tag{20}$$

For instance, checking $\pi_3[\text{B}] + \boldsymbol{\pi}_1[\text{C}]$ corresponds to checking whether or not

$$-\pi_3\boldsymbol{\lambda}\mathbf{g} + (\pi_3\mathbf{c} + \pi_3\boldsymbol{\lambda}F + \boldsymbol{\pi}_1 F)\mathbf{x} \leq \pi_3\bar{z} + |I|. \tag{21}$$

All of these checks are carried out within the LRE algorithm by defining additional edge lengths that incorporate the aggregate coefficients. (Clearly, we may view $\boldsymbol{\pi}_1$, π_2, and π_3 as Lagrangian multipliers that differ from $\boldsymbol{\lambda}$.)

Checking the aggregate constraints while enumerating paths in LRE does add computational overhead, of course, but empirical results typically show a worthwhile tradeoff in reduced enumeration. When reporting computational results in Section 5, "LRE-A" denotes the LRE algorithm with aggregate constraints used to limit enumeration, and "LRE-PA" denotes a version of LRE-A that adds the preprocessing described in Section 4.1.

4.3. Identifying a Feasible Solution

When CSPP contains multiple side constraints, a feasible solution may not be found while optimizing $\underline{z}(\boldsymbol{\lambda})$. When this happens, the basic LRE algorithm begins its path-enumeration phase with the weak upper bound $\bar{z} = (|V| - 1)c_{\max} + \gamma$, which can lead to excessive enumeration. An alternative approach, described here, first seeks to find a feasible solution and thereby a more useful upper bound.

If a feasible path exists, one can be identified by selecting an arbitrary side constraint indexed i', and by then solving this "phase-I, feasibility integer program:"

$$\begin{array}{lll} \textbf{FIP} & \min\limits_{\mathbf{x}} & \mathbf{f}_{i'}\mathbf{x} \\ & \text{s.t.} & A\mathbf{x} = \mathbf{b} \\ & & \mathbf{f}_i\mathbf{x} \leq g_i \quad \forall i \in I \setminus \{i'\} \\ & & \mathbf{x} \geq \mathbf{0} \quad \text{and integer.} \end{array}$$

Any feasible solution of **FIP** with objective value no greater than $g_{i'}$ is feasible for **CSPIP**, and hence the corresponding path yields an upper bound for **CSPIP**. We solve **FIP** using LRE as if the problem were just a CSPP, but terminate as

soon as a feasible solution to the original problem is found (if one exists).

We include this phase-I subroutine in all our "enhanced" LRE algorithms, i.e., LRE-P, LRE-A, and LRE-PA. However, this particular enhancement comes into play only for a problem with at least two side constraints and for which no feasible solution is found while optimizing $\underline{z}(\boldsymbol{\lambda})$.

FIP has only one fewer side constraint compared to the original CSPP. This is a significant reduction only for small $|I|$ and cannot account for the improvements seen in testing. The benefit of using **FIP**, in tightly constrained problems, clearly derives from the fact that $g_{i'}$ tends to be a fairly tight upper bound on the optimal objective value for **FIP** if the original CSPP is feasible: After all, it is difficult to find a feasible solution to the CSPP because the g_i are rather small. In contrast, the crude upper bound $\bar{z} = (|V| - 1)c_{\max} + \gamma$ is extremely weak for the original CSPP.

5. COMPUTATIONAL RESULTS

This section reports computational experiments with the LRE algorithm and the label-setting algorithm (LS) described by Dumitrescu and Boland [10] applied to problem instances defined on four classes of networks. We have implemented the LS algorithm using a standard two-heap data structure for labels. To facilitate accurate comparison between the two algorithms, we have implemented LS using the same subroutines for preprocessing (see Section 4.1), for maximizing the Lagrangian function $\underline{z}(\boldsymbol{\lambda})$, and determining an initial feasible solution (see Section 4.3). We let "LS-P" denote the label-setting algorithm with all these enhancements. All information available from the preliminary calculations are made available to the label-setting and path-enumeration stages of LS and LRE, respectively. Note that LS-P applies aggregate constraints only in its preprocessing stage, as we have not found such constraints to be of value within the main algorithm.

Because we solve instances of CSPP with at most ten side constraints, repeated bisection searches in the coordinate directions suffice to maximize $\underline{z}(\boldsymbol{\lambda})$ adequately and quickly. (See [9, 12].) We have verified "adequately" by solving the LP relaxation of a number of instances of CSPP using an interior-point algorithm; the reported solution times verify "quickly." All versions of LRE employ the shortest-path algorithm of Pape [31] as a subroutine. This algorithm has exponential worst-case complexity, but performs consistently well on all problem classes studied here.

We carry out computational experiments on a desktop computer with a 3.8 GHz Intel Pentium IV processor, 3 gigabytes of RAM, and the Microsoft Windows XP Professional operating system. Both LRE and LS programs are written and compiled using Microsoft Visual C++ Version 6.0.

5.1. Small-Scale Problem Instances

The first class of test problems consists of the 24 problem instances first investigated by Beasley and Christofides [3], and subsequently by Dumitrescu and Boland [10]. Our purpose is to demonstrate the efficiency of the "basic LRE algorithm," which does not use preprocessing (Section 4.1), aggregate constraints (Section 4.2), or the phase-I subroutine (Section 4.3). Each problem instance has either one or ten side constraints, and is solved to optimality using the basic LRE algorithm. Table 1 displays problem and solution statistics.

Column five of Table 1 lists the "initial optimality gap" defined as $100\% \times (\bar{z} - \underline{z}^*)/\underline{z}^*$, where $\bar{z}$ is the upper bound found prior to initiating path enumeration. An initial gap of ∞ indicates that no feasible solution is identified while optimizing $\underline{z}(\boldsymbol{\lambda})$, and the crude upper bound of $(|V| - 1)c_{\max} + \gamma$ is used. Similarly, column six gives the Lagrangian duality gap, defined as $100\% \times (z^* - \underline{z}^*)/\underline{z}^*$. For reference, column eight ("B&C") gives run times from Beasley and Christofides. Those computations were performed using FORTRAN on a CDC 7600 computer, and hence, no direct comparison of run times is possible.

Dumitrescu and Boland do not report run times for their solutions of these problems. However, they do solve most of them using only preprocessing: Column nine of Table 1 indicates whether or not the preprocessing suffices to solve the instance.

Even though most of these problems have large duality gaps, and may therefore require substantial path enumeration, the table shows that they present no computational challenge to LRE, even without enhancements.

5.2. Routing Military Units Through a Road Network

Our second class of test problems derives from planning the movement of a military unit through a road network. Consider a small convoy that must move from junction s in the network to junction t, in a limited amount of time. Planners wish to select a route that meets the time limit, but minimizes the risk of an attack (for example, from an ambush by ground forces or by the detonation of an improvised explosive device). We formulate this problem as a CSPP with one side constraint and use it to illustrate the effect of preprocessing (see Section 4.1) and aggregate constraints (see Section 4.2).

Let the weight $f_e = f_{1e}$ associated with edge $e = (u, v)$ represent the time required to traverse road segment e, and let the length c_e reflect the risk associated with that traversal. The convoy will travel with civilian traffic and obey speed limits, so f_e equals the physical length of e divided by its speed limit. We assume that larger roads, which happen to have higher speed limits, are riskier; we therefore set $c_e = f_e\beta_e$, where $\beta_e = 5.0$, 2.0, 1.0, 0.5, and 0.2 when e is a major highway, a minor highway, a major expressway, a minor expressway, or a local road, respectively. Clearly, the optimal route will traverse small, slow-speed roads as much as possible, given the time limit.

Table 2 presents computational results for various versions of LRE and for LS-P. Over a range of hypothesized time limits, the table reports percentage of edges removed by preprocessing, initial gap, duality gap, and run time. The data represent roads in Maryland, Virginia, and Washington, D.C. with speed limits of 30 miles/h and higher [5]. The

TABLE 1. Problem statistics and run times for the basic LRE algorithm applied to CSPPs from Beasley and Christofides [3].

Problem	$\|V\|$	$\|E\|$	$\|I\|$	Initial gap (%)	Duality gap (%)	Run time		Presolve D&B
						LRE (sec.)	B&C (sec.)	
BC1	100	955	1	60	47	0.00	1.9	Yes
BC2	100	955	1	45	34	0.00	0.9	Yes
BC3	100	959	1	33	33	0.00	1.9	Yes
BC4	100	959	1	0	0	0.00	1.0	Yes
BC5	100	990	10	21	21	0.02	4.6	Yes
BC6	100	990	10	16	16	0.02	4.6	Yes
BC7	100	999	10	142	62	0.02	4.4	Yes
BC8	100	999	10	∞	227	0.02	6.3	No
BC9	200	2,040	1	18	18	0.00	2.0	Yes
BC10	200	2,040	1	0	0	0.00	2.0	Yes
BC11	200	1,971	1	0.1	0.1	0.00	4.0	Yes
BC12	200	1,971	1	0.1	0.1	0.00	3.9	Yes
BC13	200	2,080	10	133	100	0.05	5.2	Yes
BC14	200	2,080	10	infeas.	infeas.	0.08	9.3	Yes
BC15	200	1,960	10	∞	61	0.05	9.2	Yes
BC16	200	1,960	10	∞	120	0.05	12.1	No
BC17	500	4,858	1	41	34	0.00	10.6	Yes
BC18	500	4,858	1	32	25	0.00	10.5	Yes
BC19	500	4,978	1	0.0	0.0	0.00	11.1	Yes
BC20	500	4,978	1	0	0	0.02	6.4	Yes
BC21	500	4,847	10	33	33	0.09	13.6	Yes
BC22	500	4,847	10	25	25	0.09	13.1	Yes
BC23	500	4,868	10	22	22	0.08	26.3	Yes
BC24	500	4,868	10	36	36	0.08	26.3	Yes

Run times exclude problem-generation time; problems are solved to optimality. BC14 is infeasible and the time reported there is for proving infeasibility.

TABLE 2. Computational results for solving CSPPs to plan the movement of a military convoy through a road network.

Time lim. g (min.)	Edges pre-processed (%)	Initial gap (%)		Duality gap (%)		Run time (sec.)				
		No pre.	Pre.	No pre.	Pre.	LRE	LRE-P	LRE-A	LRE-PA	LS-P
240†	100	NA	NA	NA	NA	0.02	0.00	0.02	0.02	0.02
250	92	2.1	<1.0	2.1	<1.0	0.02	0.00	0.02	0.03	0.75
260	83	1.7	1.7	<1.0	<1.0	0.02	0.02	0.02	0.02	0.75
270	77	16.6	16.6	<1.0	<1.0	75.0	75.2	16.1	16.4	0.77
280	73	15.0	14.2	6.2	5.2	11.9	7.27	3.09	1.86	0.77
290	70	35.3	11.7	6.5	3.8	3.77	0.25	1.09	0.08	0.77
300	68	64.4	33.2	9.2	3.3	1775	179	440	42.1	9.16
310	64	<1.0	<1.0	<1.0	<1.0	0.02	0.00	0.02	0.02	0.77
320	59	12.9	12.9	1.9	1.9	0.45	0.45	0.22	0.24	0.77
330	49	<1.0	<1.0	<1.0	<1.0	0.02	0.00	0.02	0.02	0.77
340	45	1.1	1.1	<1.0	<1.0	0.03	0.03	0.03	0.03	0.77
350	40	<1.0	<1.0	<1.0	<1.0	0.03	0.03	0.03	0.02	0.78
360	37	1.2	1.2	<1.0	<1.0	0.81	0.83	0.17	0.19	0.78

Problems are solved to optimality. The instance marked with "†" is infeasible because the time limit of 240 is too small.

resulting graph has 3,670 vertices and 9,876 edges. Road segments with speed limits 65, 55, 50, 45, and 30 miles/h are categorized as major highways, minor highways, and so on, respectively. Note that the convoy cannot reach its destination in 240 min, and no reduction in risk accrues by allowing a transit time greater than 360 min.

The second column of Table 2 displays the percentage of edges that preprocessing removes. Most edges are eliminated by preprocessing in tightly constrained problems, but even modestly successful preprocessing can tighten the Lagrangian lower bound substantially. Columns 3–6 in the table establish this fact, by displaying the initial gap and the duality gap, without preprocessing ("No pre.") and with preprocessing ("Pre."). For $g = 300$, tightening the lower bound also reduces run time significantly: Compare the run times for LRE to those for LRE-P, and compare the times for LRE-A and LRE-PA.

Another comparison illustrates the beneficial effect of the aggregate constraints described in Section 4.2. For the cases $g = 270$ and $g = 300$, these constraints reduce run times significantly: Compare columns seven and nine. We observe

similar reductions by comparing LRE with preprocessing (LRE-P), to the complete algorithm with preprocessing and aggregate constraints (LRE-PA): Compare columns eight and ten. Overall, Table 2 indicates that LRE, particularly with enhancements, does solve the CSPP on this real-world network quite efficiently. For the sake of comparison, column 11 of Table 2 lists the run times for LS-P. LS-P is usually slower than LRE-PA, except in some cases with large initial gaps. A more comprehensive comparison between these algorithms follows.

5.3. Grid Networks

Grid networks, with the same structure as those studied by Dumitrescu and Boland [10], comprise the third set of test problems. Our purpose with this computational study is to provide a comprehensive comparison of the relative efficiencies of the LRE and LS algorithms. As in other tests, to make comparisons as objective as possible, both algorithms use identical ancillary routines.

The test networks, denoted "Grid(a, b)," derive from a rectangular grid, a vertices tall and b vertices wide. A source vertex s, external to the grid, connects to each vertex in the leftmost column of the grid, while each vertex in the rightmost column connects to an external sink t. Each vertex u within the grid has (up to) three edges (u, v) directed out of it, up, down, and from left to right, as long as the vertex v exists in the grid. Edge lengths and weights are uniform, randomly generated integers in the range [1,10] for vertical edges, and in the range [80,100] for horizontal edges and for external edges. For each $i \in I$, the weight limits are $g_i = \alpha g_{\max,i} + (1-\alpha) g_{\min,i}$, where $g_{\min,i}$ denotes the total weight of the minimum-weight path with respect to i, and $g_{\max,i}$ denotes the total weight, with respect to i, of the shortest path (with respect to $\mathbf{c}$). As in Dumitrescu and Boland, we examine α set to the low (L), medium (M), and high (H) values of 0.05, 0.50, and 0.95, respectively: The "L-instances" are tightly constrained, the "H-instances" are loosely constrained, and the "M-instances" are somewhere in between.

5.3.1. Singly Constrained CSPPs on Grid Networks. Table 3 shows the run times for 10 singly constrained problems also solved and reported by Dumitrescu and Boland [10]. The data for these instances (only) were obtained from one of those authors who indicates that, for a given setting of a, b, and α, each represents the most computationally challenging instance extracted from a large set of randomly generated instances (Dumitrescu, private communication, 2005). The first five come from "problem class 4-L, Type 2" in [10], and the second five come from "problem class 4-M, Type 2" in the same article. All problems are solved to optimality using both LRE-PA and LS-P. "Speedup" is the apparent improvement of LRE-PA over LS-P, computed as 100% × (LS-P sec. − LRE-PA sec.)/(LS-P sec.). On average, LRE-PA solves these problems 82% faster than LS-P.

Table 4 further investigates the behavior of the two algorithms for CSPP by examining the average and standard deviation of run times over 20 randomly generated instances from the problem classes used in Table 3's comparisons. LRE-PA solves all instances to optimality quickly, with the exception of one instance of Grid(450, 300) with the medium weight limit (marked with "†" in Table 4). In this instance, the algorithm finds the optimal solution and proves it to be within 0.5% of optimality in 0.3 s, but requires 789 s to prove optimality.

Table 4 indicates that, with one exception, average run times for these problem classes are consistent with the results in Table 3. And, with one exception, standard deviations are reasonably small. Thus, LRE-PA seems to perform well, with good but imperfect consistency.

TABLE 3. Comparison of LRE-PA to the label-setting algorithm LS-P of Dumitrescu and Boland [10].

Grid	Weight limit	$\|V\|$	$\|E\|$	Run time (sec.) LRE-PA	Run time (sec.) LS-P	Speedup (%)
(30, 100)	L	3,002	8,830	0.06	0.08	21
(100, 100)	L	10,002	29,990	0.08	0.38	79
(200, 200)	L	40,002	119,800	0.25	2.39	90
(350, 200)	L	70,002	209,950	0.38	4.92	92
(450, 300)	L	135,002	404,850	0.99	13.5	93
(30, 100)	M	3,002	8,830	0.03	0.14	78
(100, 100)	M	10,002	29,990	0.05	0.36	87
(200, 200)	M	40,002	119,800	0.20	2.34	91
(350, 200)	M	70,002	209,950	0.27	4.84	95
(450, 300)	M	135,002	404,850	0.74	13.4	95

Test problems are singly constrained CSPPs on grid networks of problem class 4-L, Type 2, and problem class 4-M, Type 2 from [10].

TABLE 4. Comparison of run times for LRE-PA and LS-P for singly constrained CSPPs on grid networks.

Grid	Weight limit	Run time (sec.) LRE-PA avg.	Run time (sec.) LRE-PA s.d.	Run time (sec.) LS-P avg.	Run time (sec.) LS-P s.d.	Average speedup (%)
(30, 100)	L	0.02	0.01	0.04	0.01	50
(100, 100)	L	0.04	0.01	0.14	0.02	71
(200, 200)	L	0.23	0.14	1.27	0.53	82
(350, 200)	L	0.44	0.18	2.65	0.87	83
(450, 300)	L	1.18	1.50	8.99	3.65	87
(30, 100)	M	0.01	0.01	0.04	0.02	75
(100, 100)	M	0.04	0.01	0.17	0.04	76
(200, 200)	M	0.35	0.60	1.56	0.34	78
(350, 200)	M	0.29	0.19	3.24	1.03	91
(450, 300)	M	40.7†	172	10.8	0.06	−277
(30, 100)	H	0.01	0.01	0.02	0.02	50
(100, 100)	H	0.02	0.01	0.10	0.08	80
(200, 200)	H	0.09	0.04	1.00	0.77	91
(350, 200)	H	0.13	0.06	1.66	1.74	92
(450, 300)	H	0.31	0.14	4.96	5.26	94

Each row represents 20 randomly generated instances of the indicated problem type. (See the text regarding the time annotated by "†.")

5.3.2. Multi-Constrained CSPPs on Grid Networks. Tables 5–9 report results for LS-P and LRE-PA when solving the CSPPs of Table 4, but with two to ten side constraints instead of one. Other than the small problems from Beasley and Christofides [3], Dumitrescu and Boland [10] do not solve any multiconstrained CSPPs. The goal here is simply to explore the behavior of the two algorithms over a wider range of problems and optimality tolerances, and to see if one algorithm dominates the other in certain situations.

For each grid size and number of side constraints $|I|$ in Table 5, and for both algorithms, we attempt to solve 20 randomly generated problem instances with medium (M) weight limits. We report the number of instances solved successfully in less than 30 min, along with the average run time and standard deviation for the successfully solved problems. (These statisics should be viewed warily, of course.)

Table 5 shows that LS-P is faster than LRE-PA when $|I| = 2$. LS-P and LRE-PA solve 100 and 93 of these instances, respectively, within the time limit of 30 min. However, for $|I| > 2$, LRE-PA appears to be at least as fast as LS-P. We note that for $|I| = 10$, both algorithms have identical run times because all these instances are proven to be infeasible through the preprocessing and phase-I subroutines, which the algorithms have in common. Some large standard deviations in run times do appear in the table, for both LS-P and LRE-PA. Thus, neither algorithm is free from data-induced instabilities. Overall, LRE-PA solves 72% of the problems within the 30-minute time limit, while LS-P solves 65%.

Table 6 displays statistics analogous to those in Table 5, but with problem instances solved to a 1% optimality tolerance rather than to optimality. Naturally, both algorithms can now solve more problems within the time limit: LS-P achieves

TABLE 5. Run-time statistics for LRE-PA and LS-P when solving 20, randomly generated, multi-constrained CSPPs on grid networks with $|I|$ side constraints and with medium (M) weight limits.

		$\|I\| = 2$		$\|I\| = 3$		$\|I\| = 4$		$\|I\| = 5$		$\|I\| = 10$	
Grid	Statistics	LRE	LS	LRE	LS	LRE	LS	LRE	LS	LRE	LS
(30,100)	avg. (sec.)	0.68	0.11	0.93	7.54	31.9	256	112	87.6	1.62	1.62
	s.d. (sec.)	1.21	0.06	1.46	16.2	100	462	197	193	4.90	4.90
	no. solved	20	20	20	20	19	14	15	6	20	20
(100,100)	avg. (sec.)	3.08	0.37	24.9	41.4	189	173	97.8	187	0.66	0.66
	s.d. (sec.)	9.85	0.10	72.5	134	305	329	175	402	1.93	1.93
	no. solved	20	20	19	19	19	15	19	9	20	20
(200,200)	avg. (sec.)	135	4.28	214	218	619	–	4.90	4.90	0.10	0.10
	s.d. (sec.)	305	6.70	358	434	679	–	0	0	0.01	0.01
	no. solved	19	20	17	16	5	0	1	1	20	20
(350,200)	avg. (sec.)	5.97	5.22	214	413	93.7	28.3	283	7.97	2.56	2.56
	s.d. (sec.)	8.68	0.95	412	551	155	10.5	275	0	10.4	10.4
	no. solved	18	20	14	15	7	2	2	1	20	20
(450,300)	avg. (sec.)	54.5	16.9	364	55.2	605	127	16.8	16.8	5.02	5.02
	s.d. (sec.)	121	5.10	449	44.0	511	0	0	0	20.2	20.2
	no. solved	16	20	9	5	4	1	1	1	20	20

Problems are solved to optimality.

TABLE 6. Run-time statistics for solving the same CSPPs as in Table 5, except that the optimality tolerance is 1%.

		$\|I\| = 2$		$\|I\| = 3$		$\|I\| = 4$		$\|I\| = 5$		$\|I\| = 10$	
Grid	Statistics	LRE	LS	LRE	LS	LRE	LS	LRE	LS	LRE	LS
(30,100)	avg. (sec.)	0.01	0.02	0.04	0.04	7.87	14.3	127	61.2	1.62	1.62
	s.d. (sec.)	0.01	0.02	0.07	0.04	28.0	40.0	339	158	4.90	4.90
	no. solved	20	20	20	20	20	20	20	16	20	20
(100,100)	avg. (sec.)	0.03	0.05	0.69	0.35	10.4	33.8	19.1	3.40	0.66	0.66
	s.d. (sec.)	0.02	0.07	1.94	0.21	24.9	136	57.1	9.16	1.93	1.93
	no. solved	20	20	19	20	19	19	20	15	20	20
(200,200)	avg. (sec.)	0.11	0.11	1.94	0.73	174	3.83	454	155	0.10	0.10
	s.d. (sec.)	0.04	0.04	6.25	0.98	357	6.36	494	300	0.01	0.01
	no. solved	20	20	20	20	16	17	11	14	20	20
(350,200)	avg. (sec.)	0.18	0.18	1.26	1.77	83.8	78.8	321	61.0	2.56	2.56
	s.d. (sec.)	0.09	0.09	1.76	2.35	161	315	540	144	10.4	10.4
	no. solved	20	20	20	20	18	20	9	10	20	20
(450,300)	avg. (sec.)	0.35	0.35	1.18	1.91	53.8	28.6	6.19	34.3	5.02	5.02
	s.d. (sec.)	0.20	0.20	0.81	3.77	156	68.1	5.53	46.6	20.2	20.2
	no. solved	20	20	20	20	17	19	5	14	20	20

TABLE 7. Run-time statistics for solving the same CSPPs as in Table 5, except that the weight limit on side constraints is high (H).

		$\|I\| = 2$		$\|I\| = 3$		$\|I\| = 4$		$\|I\| = 5$		$\|I\| = 10$	
Grid	Statistics	LRE	LS	LRE	LS	LRE	LS	LRE	LS	LRE	LS
(30,100)	avg. (sec.)	0.02	0.05	0.06	0.09	0.14	0.18	0.24	0.49	12.6	43.1
	s.d. (sec.)	0.01	0.03	0.05	0.05	0.15	0.16	0.33	0.96	25.1	68.0
	no. solved	20	20	20	20	20	20	19	20	18	14
(100,100)	avg. (sec.)	0.05	0.20	0.24	0.30	2.44	0.47	71.2	1.98	33.2	130
	s.d. (sec.)	0.02	0.10	0.40	0.15	9.72	0.09	297	3.76	72.8	203
	no. solved	20	20	20	20	20	20	19	20	15	18
(200,200)	avg. (sec.)	43.9	1.63	1.32	2.63	32.9	8.66	19.6	15.1	148	343
	s.d. (sec.)	187	0.88	1.73	0.83	62.1	10.1	46.3	31.1	242	526
	no. solved	20	20	20	20	19	20	18	20	15	10
(350,200)	avg. (sec.)	35.6	4.02	4.78	4.88	41.0	38.6	114	25.4	220	231
	s.d. (sec.)	150	1.27	8.26	2.34	128	115	374	30.0	380	247
	no. solved	20	20	20	20	18	20	17	20	12	11
(450,300)	avg. (sec.)	25.5	11.3	19.4	16.6	110	29.4	68.8	38.2	446	86.1
	s.d. (sec.)	103	4.60	34.5	6.55	204	15.8	129	37.3	499	51.2
	no. solved	20	20	19	20	17	20	12	15	6	3

Problems are solved to optimality.

a small advantage over LRE-PA by solving 93% of the instances within 30 minutes, compared to 91% for LRE-PA.

Table 7 shows statistics analogous to those in Table 5, but using problems with the high (H) weight limit. We observe that the relaxed weight limit results in easier problems with improved performance for both algorithms. LS-P is slightly faster than LRE-PA for $|I| = 2$ and $|I| = 3$, and this advantage becomes more substantial for $|I| = 4$ and $|I| = 5$. For $|I| = 10$, however, LRE-PA dominates. Overall, the algorithms exhibit similar performances, with LS-P solving 90% of the problem instances, and LRE-PA solving 89%. Table 8 shows statistics analogous to those of Table 7, but with the optimality tolerance set at 1%. In this case, the two algorithms perform equally well, with 99.8% of the problem instances solved within 30 min.

Statistics for problems with low (L) weight limits are not listed because: (i) All 500 randomly generated instances are infeasible, and (ii) the preprocessing routines, which the two algorithms hold in common, prove this in less than 5 s for each instance.

Table 9 summarizes the computational study of multi-constrained grid networks (detailed in Tables 5–8). This table displays the total number of problem instances that solve within the 30-min, individual-problem limit. For each value of $|I|$, the total number of instances tested is 100 (5 network sizes $\times$ 20 randomly generated instances). The last column of Table 9 gives the percentage of instances solved within the time limit, over all grid sizes and numbers of side constraints. In relatively easy cases, where both algorithms perform well, LS-P tends to be faster than LRE-PA: See cases $|I| = 3$ and $|I| = 4$ with high weight limits and a 0% optimality tolerance, and with medium weight limits and a 1% tolerance. In difficult cases, however, LRE-PA appears to outperform LS-P: See cases $|I| = 4$ and $|I| = 5$ with medium weight limits

TABLE 8. Run-time statistics for solving the same CSPPs as in Table 5, except the weight limit is high (H) and the optimality tolerance is 1%.

		$\|I\| = 2$		$\|I\| = 3$		$\|I\| = 4$		$\|I\| = 5$		$\|I\| = 10$	
Grid	Statistics	LRE	LS	LRE	LS	LRE	LS	LRE	LS	LRE	LS
(30,100)	avg. (sec.)	0.00	0.00	0.01	0.00	0.01	0.00	0.01	0.01	2.36	0.04
	s.d. (sec.)	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	10.2	0.11
	no. solved	20	20	20	20	20	20	20	20	20	20
(100,100)	avg. (sec.)	0.01	0.01	0.02	0.01	0.14	0.04	0.05	0.05	0.23	0.25
	s.d. (sec.)	0.01	0.01	0.00	0.01	0.52	0.09	0.06	0.12	0.54	0.58
	no. solved	20	20	20	20	20	20	20	20	19	19
(200,200)	avg. (sec.)	0.05	0.04	0.07	0.06	0.09	0.07	0.10	0.09	0.16	0.16
	s.d. (sec.)	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02
	no. solved	20	20	20	20	20	20	20	20	20	20
(350,200)	avg. (sec.)	0.08	0.07	0.11	0.10	0.14	0.12	0.18	0.15	0.32	0.32
	s.d. (sec.)	0.01	0.01	0.02	0.01	0.02	0.01	0.03	0.02	0.16	0.16
	no. solved	20	20	20	20	20	20	20	20	20	20
(450,300)	avg. (sec.)	0.19	0.15	0.25	0.21	0.30	0.25	0.38	0.31	0.57	0.57
	s.d. (sec.)	0.02	0.01	0.03	0.02	0.01	0.02	0.04	0.03	0.05	0.05
	no. solved	20	20	20	20	20	20	20	20	20	20

TABLE 9. Summary statistics for a computational study of multi-constrained grid networks (from Tables 5–8).

Algorithm	Weight limits	Optimality tolerance	Problems solved over all grid sizes					Total % solved
			$\|I\| = 2$	$\|I\| = 3$	$\|I\| = 4$	$\|I\| = 5$	$\|I\| = 10$	
LRE-PA	Medium	0	93	79	54	38	100	72.4
LS-P	Medium	0	100	75	32	18	100	65.0
LRE-PA	High	0	100	99	94	85	66	88.8
LS-P	High	0	100	100	100	95	56	90.2
LRE-PA	Medium	1	100	99	90	65	100	90.8
LS-P	Medium	1	100	100	95	69	100	92.8
LRE-PA	High	1	100	100	100	100	99	99.8
LS-P	High	1	100	100	100	100	99	99.8

and a 0% optimality tolerance, and see $|I| = 10$ with high weight limits and a 0% optimality tolerance.

5.4. Routing Aircraft on Dense Networks

Our fourth and final set of test problems examines the performance of LRE and LS on a class of networks arising in the routing of military aircraft. This class involves fairly dense graphs with potentially large duality gaps. Hence, these networks differ substantially from the grid networks examined in Section 5.3, which are sparse and which have relatively small duality gaps.

When routing military aircraft, the goal is to identify a fuel-constrained, minimum-risk route from an entry point in an area of operations (AO), through enemy airspace, to a fixed destination. We consider an F/A-18 strike group, typically comprising two to ten aircraft of various types (e.g., electronic warfare, fighter, strike), whose mission is to destroy or disable some ground or naval target. Each aircraft in the group risks being shot down by enemy surface-to-air missiles (SAMs).

We formulate this routing problem as a singly constrained CSPP on a two-dimensional network consisting of a highly connected grid of vertices. The edge length c_e measures the risk of traveling along e. (The AO contains 15 SAM threats that generate various risk values for the edges.) Edge e's weight $f_e = f_{1e}$ represents fuel consumption along e, with the Euclidean length of the edge used as a surrogate. Current doctrine specifies that the F/A-18, and similar aircraft, will maintain a constant, fuel-efficient altitude of about 36,000 feet, so a two-dimensional grid suffices to model the relevant airspace. We cover the airspace with a 26×38 rectangular grid of vertices (i.e., $|V| = 988$), with a spacing of eight nautical miles (nm). The grid covers an AO of 200 by 296 nm with the southwest corner being the origin in a Cartesian coordinate system measured in nautical miles. The strike group enters the AO at its western edge at x-y coordinates (0,104); the destination lies directly east at coordinates (296,104).

The simplest discretization of the AO might connect nearest-neighbor vertices, including diagonals, with edges. The resulting network would be sparse and the computational burden low, but it could lead to unrealistically jagged flight paths. On the other hand, modeling straight-line flight segments between every vertex pair would yield a dense, complete network with about 10^6 edges, and a high computational burden. Consequently, we explore the eight edge structures labeled A-H in Table 10: These are much denser than the grid and road networks examined above, but sparser than a complete network. For instance, Structure A connects each vertex u to each vertex v that lies at a distance of between 8 nm ("Edge length, min") and 12 nm ("Edge length, max"), but only if v lies no further west than u. (The general west-to-east travel of the strike group makes westward travel unlikely, so none of the structures contain edges with a west-bound vector component.) We justify models with no short edges (see F, G, and H in Table 10) by the fact that solutions to these models cannot exhibit much zig-zagging, which is desirable from a pilot's perspective. Note that the minimum possible fuel consumption for the strike group is 296.

The last six columns of Table 10 show the run times for LS-P, LRE-PA, and LRE-PA with "reprocessing" (LRE-PAR), which is described below. Results indicate that LRE-PA is substantially faster than LS-P for most instances with network structures A, B, and G. The results are mixed for structures C, D, E, F, and H: LRE-PA is often faster than LS-P for these five cases but run times exhibit substantial variability, and LRE-PA can be much slower on occasion. The long run times correspond to problems with large duality gaps. For example, LRE-PA solves the "D-problem," with fuel limit 310, in 153 s. This problem has an initial optimality gap of 264%, and a duality gap of 117%. On the other hand, LRE-PA solves the same problem, with a fuel limit of 320, in only 0.16 s. This problem has an initial optimality gap of 41% and a duality gap of only 4%.

To improve the robustness of LRE-PA, we have experimented with application of the preprocessing routines within the enumeration phase of the algorithm. In this phase, LRE-PA typically finds a sequence of improving solutions, i.e., upper bounds. Each time a new upper bound is found, another "preprocessing" scan (see Section 4.1) may shrink the network further and reduce enumeration. We refer to this application of the preprocessing routines as "reprocessing."

Preliminary numerical testing on this class of problems indicates that a single reprocessing scan, applied each time the upper bound improves, can reduce run times significantly over LRE-PA. However, reprocessing does add computational overhead, and it may reduce the network only

TABLE 10. Run-time statistics for solving aircraft-routing CSPPs with various fuel constraints and network structures.

		Edge length			Run times for various fuel limits g					
Structure	$\|E\|$	min	max	Algorithm	300	310	320	330	340	350
A	4,712	8	12	LRE-PA	0.02	0.02	0.06	0.02	0.02	0.02
				LRE-PAR	0.02	0.02	0.02	0.02	0.02	0.02
				LS-P	0.41	0.41	0.41	0.41	0.41	0.42
B	11,048	8	18	LRE-PA	0.06	0.50	0.08	0.08	1.05	1.56
				LRE-PAR	0.00	0.00	0.00	0.00	0.05	0.02
				LS-P	0.41	0.42	0.44	2.20	1.72	1.17
C	22,222	8	30	LRE-PA	3.23	0.13	0.03	0.02	0.42	1.11
				LRE-PAR	0.03	0.02	0.03	0.02	0.42	0.02
				LS-P	0.42	0.44	0.42	0.42	0.44	0.42
D	123,166	8	80	LRE-PA	0.16	153	0.16	0.33	0.23	23.1
				LRE-PAR	0.16	0.47	0.30	0.16	0.25	0.14
				LS-P	0.56	1.92	0.56	0.56	0.59	0.59
E	228,042	8	120	LRE-PA	0.31	269	0.31	0.47	0.52	39.6
				LRE-PAR	0.30	0.55	0.47	0.31	0.59	0.31
				LS-P	0.72	2.34	0.75	0.78	0.80	0.80
F	223,330	16	120	LRE-PA	0.28	3.03	0.30	0.31	0.31	0.50
				LRE-PAR	0.22	0.45	0.44	0.25	0.31	0.50
				LS-P	0.69	0.92	0.70	0.72	0.74	0.75
G	195,110	40	120	LRE-PA	0.23	0.25	0.25	0.25	0.25	0.25
				LRE-PAR	0.19	0.38	0.24	0.22	0.25	0.25
				LS-P	0.66	0.67	0.67	0.66	0.67	0.69
H	118,454	16	80	LRE-PA	0.14	1.67	0.14	0.17	0.14	0.27
				LRE-PAR	0.13	0.25	0.28	0.13	0.14	0.28
				LS-P	0.56	0.69	0.56	0.56	0.58	0.56

All problems are solved to optimality. "LRE-PAR" denotes the LRE-PA algorithm with reprocessing.

modestly, or not all. We find it more efficient to reprocess only after the upper bound has improved a suitable amount compared to the last time reprocessing was executed. Without extensive numerical experimentation, we adopt this empirical rule: Execute one reprocessing scan of all the edges whenever the upper bound reduces to 90% of the value found after the last scan.

Table 10 reports computational results for reprocessing in the rows marked "LRE-PAR." Clearly, LRE-PAR is almost always faster than LRE-PA, and it is substantially faster for difficult instances. Hence, reprocessing appears to be a valuable technique. (Reprocessing does not substantially reduce average run times in the grid-network problems of Section 5.3, however, because most of those problems have only a small duality gap, and few updates of the upper bounds occur.)

We note that reprocessing seems unlikely to be useful for LS-P, except for unusual network topologies. This is true because any label-setting algorithm for CSPP will usually find a sequence of improving solutions only near the end of the solution procedure, i.e., after expending the bulk of the work required to solve the problem. The breadth-first nature of LS-P's search mechanism produces this behavior (which differs greatly from the behavior induced by the depth-first search embedded in all LRE variants). We have verified this conjecture by examining the challenging problem instances "D" and "E" with fuel limit 310 (see Table 10). For both network structures, LRE-PA finds an optimal solution in less than 0.5% of its total run time, but spends the remaining time verifying that solution's optimality. Reprocessing definitely helps here, as indicated by the reduced run times for LRE-PAR. In contrast, LS-P spends almost 100% of its run time on each of these problems before it finds a significantly improved feasible solution. Thus, no opportunity arises for reprocessing to speed up LS-P. (A hybrid algorithm comes to mind for future research, however: Run LRE-PAR until reprocessing has a chance to reduce the network, and then switch to LS-P.)

6. CONCLUSIONS

We have described a new, highly effective algorithm for solving the constrained shortest-path problem (CSPP), which seeks a shortest *s*-*t* path in a directed network that satisfies one or more side constraints with respect to edge weights. Our basic "LRE algorithm" Lagrangianizes all side constraints, optimizes the resulting Lagrangian function, defines new edge lengths through the Lagrangian function, and enumerates all near-shortest paths in order to close any remaining optimality gap. This enumeration defines a specialized branch-and-bound algorithm, with a depth-first enumeration tree, that updates but does not reoptimize the Lagrangian lower bound at each node in the tree.

The basic LRE algorithm solves many problems quickly, but standard preprocessing is fast and can improve performance. ("Standard preprocessing" eliminates vertices and edges that cannot lie on any feasible path using simple bounding arguments based on edge weights. This is extended to

"cannot lie on any optimal path" when a feasible solution is available.) We also find the following enhancements useful:

1. Adding aggregate constraints to improve the effectiveness of preprocessing and to reduce effort in the path-enumeration phase of the algorithm.
2. When the process of optimizing the Lagrangian lower bound does not yield an initial feasible solution, solving a phase-I problem to find one. This problem is a variant of the original CSPP that moves one of the side constraints into the objective function.
3. Executing additional preprocessing scans within the algorithm's enumeration phase (called "reprocessing") to take advantage of improving upper bounds from improving feasible solutions.

The first and third enhancements, even when not required, do not add significantly to computational overhead. The second enhancement may involve substantial computational effort, but that effort seems to be worthwhile when an initial feasible solution is not available.

Testing on a variety of instances with up to ten side constraints indicates that LRE is competitive with a state-of-the-art label-setting algorithm (LS) and can be significantly faster in certain cases. Specifically, on singly constrained grid networks, LRE is typically 5–10 times faster than LS. On the most difficult group of problem instances considered (multi-constrained grid networks with medium weight limits), LRE solves up to twice as many instances as LS within a 30-min time limit.

LS can be faster than LRE, especially in the presence of large duality gaps. However, at least in the dense networks tested, reprocessing added to LRE (along with the preprocessing and constraint aggregation) completely overcomes the computational penalty imposed by large duality gaps. Indeed, LS dominates LRE without reprocessing in those dense networks, but the situation reverses with reprocessing added to LRE. We note that reprocessing will be of little value to LS, because that algorithm typically finds its first feasible solution only after completing the bulk of its computation.

We see several avenues of additional research that may lead to even faster algorithms. Simple ideas may prove useful, for example, extending preprocessing to investigate pairs of adjacent edges, (u, v) and (v, w), in order to determine if vertex v cannot lie on a feasible path. Various "decomposition schemes" may also prove useful. For instance, suppose a network's topology implies that an optimal path in G must first pass through exactly one vertex in some easily identifiable subset of vertices $V' = \{v_1, v_2, \ldots, v_k\}$. The CSPP's solution may then be found by solving, for $i = 1, \ldots, k$, the presumably simpler CSPPs defined on G with vertices $V' \setminus \{v_i\}$ deleted. In fact, such a decomposition reduces LRE's 40.7 s average solution time in Table 4, indicated by "†," to less than four seconds.

More complicated ideas may prove useful, too. In particular, a hybrid LS/LRE algorithm could reduce some of the variability seen in solution times for CSPP, especially since the two pure algorithms seem to have complementary behavior. That is, when one algorithm exhibits especially poor performance, the other often does not.

After preprocessing, one hybrid algorithm we envisage would (i) use the label-setting paradigm to compute labels starting backwards from t, (ii) stop when some limit on time or number of labels is reached, and (iii) finish solving the problem through path enumeration starting at s. This approach would also help reduce the potential for excessive computer-memory requirements associated with a label-setting algorithm, which may need to store a huge number of labels at any one time.

Another hybrid algorithm would run LRE, using all options enabled, until reprocessing has a chance to reduce the network. Assuming some useful reprocessing reductions have been achieved, the hybrid would then switch to LS. This hybrid would bring the benefits of reprocessing to LS, which do not accrue to the nominal algorithm.

In essence, both LRE and LS are branch-and-bound procedures for CSPP in which the local lower bound is updated, but not reoptimized. The number of possible variants and hybrids is enormous.

Acknowledgments

The authors thank Professor Irina Dumitrescu for providing data for some of our computational tests, and thank two anonymous referees for their helpful suggestions. The authors also thank the Air Force Office of Scientific Research, the Office of Naval Research and the Naval Postgraduate School sabbatical program for funding this research. Additionally, Kevin Wood thanks the University of Auckland, Department of Engineering Science, for providing support in the preparation of this manuscript.

APPENDIX

Path-Enumeration Subroutine for LRE Algorithm to Solve CSPP

INPUT: A directed graph $G = (V, E)$ in adjacency-list format, s, t, edge length vector $\mathbf{c} \geq \mathbf{0}$, side-constraint data for $F\mathbf{x} \leq \mathbf{g}$ with $\mathbf{f}_i \geq 0$, optimal or near-optimal Lagrangian vector $\boldsymbol{\lambda}$ for **CSPLR**, upper bound $\bar{z}$, lower bound $\underline{z}(\boldsymbol{\lambda})$ and optimality tolerance $\delta \geq 0$.
OUTPUT: A δ-optimal shortest path $\mathbf{x}^*$ satisfying $F\mathbf{x}^* \leq \mathbf{g}$, if such a path exists.
NOTE: "firstEdge(v)" points to the beginning of a linked list of edges directed out of v

```
{
  c' ← c + λF;
  /* Add a "0-th side constraint" to limit enumeration based on c */
  I+ ← I ∪ {0}; f0 ← c; g0 ← z̄;
  /* The following requires just one backwards shortest-path calculation */
  for ( all v ∈ V ) { d(v) ← minimum distance, in terms of c', from v to t; }
  for ( each side constraint i ∈ I+ ){
    /* Solve a backwards shortest-path problem using edge "lengths" fi */
    for ( all v ∈ V ) { di(v) ← minimum weight, in terms of fi, from v to t; }
  }
  for ( all v ∈ V ) { nextEdge(v) ← firstEdge(v); }
  L(s) ← −λg; /* Initialize path length with the Lagrangian constant term */
  for ( all i ∈ I+ ) { Li(s) ← 0; } /* Initial path weight with respect fi is 0 */
  theStack ← s; onStack(s) ← true; onStack(v) ← false ∀ v ∈ V\{s};
  while ( theStack is not empty ){
    u ← vertex at the top of theStack;
    if ( nextEdge(u) ≠ ∅ ) {
      e ← the edge pointed to by nextEdge(u); /* e = (u, v) */
```

```
        increment nextEdge(u);
        if (     (onStack(v) = false) and (L(u) + c'_e + d(v) < z̄ − δ)
           and (L_i(u) + f_ie + d_i(v) ≤ g_i ∀ i ∈ I^+) ) {
           if ( v = t ) { /* An improved solution has been found */
              Represent the feasible path encoded as theStack ∪ {t} through
              its edge-incidence vector x̂;
              z̄ ← cx̂; g_0 ← z̄; x* ← x̂;
              /* Preemptive termination is possible in the following step */
              if ( z̄ − z(λ) ≤ δ ) goto Finish;
           } else {
              push v on theStack; onStack(v) ← true;
              L(v) ← L(u) + c_e;
              for ( all i ∈ I^+ ) { L_i(v) ← L_i(u) + f_ie; }
           }
        }
     } else {
        Pop u from theStack; onStack(u) ← false;
        nextEdge(u) ← firstEdge(u);
     }
  }
  Finish: If x* is empty Print ( Problem is infeasible ), otherwise Print ( x* );
}
```

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network flows: Theory, algorithms and applications, Prentice Hall, Englewood Cliffs, 1993, pp. 95–157, 598–648.

[2] Y. Aneja, V. Aggarwal, and K. Nair, Shortest chain subject to side conditions, Networks 13 (1983), 295–302.

[3] J. Beasley and N. Christofides, An algorithm for the resource constrained shortest path problem, Networks 19 (1989), 379–394.

[4] D. Boerman, Finding an optimal path through a mapped minefield, Thesis, Naval Postgraduate School, Monterey, California, 1994.

[5] W.M. Carlyle and R.K. Wood, Near-shortest and K-shortest simple paths, Networks 46 (2003), 98–109.

[6] P.R. Day and D.M. Ryan, Flight attendant rostering for short-haul airline operations, Oper Res 45 (1997), 649–661.

[7] H. De Neve and P. Van Mieghem, TAMCRA: A tunable accuracy multiple constraints routing algorithm, Comput Commun 23 (2000), 667–679.

[8] M. Desrochers and F. Soumis, A generalized permanent labeling algorithm for the shortest path problem with time windows, Inform Sys and Oper Res 26 (1988), 191–212.

[9] D. DeWolfe, J. Stevens, and K. Wood, Setting military reenlistment bonuses, Naval Res Logist 40 (1993), 143–160.

[10] I. Dumitrescu and N. Boland, Improved preprocessing, labeling and scaling algorithm for the weight-constrained shortest path problem, Networks 42 (2003), 135–153.

[11] M.L. Fisher, The Lagrangian relaxation method for solving integer programming problems, Management Sci 27 (1981), 1–18.

[12] B.L. Fox and D.M. Landi, Searching for the multiplier in one-constraint optimization problems, Oper Res 18 (1970), 253–262.

[13] M. Gamache, F. Soumis, M. Marquis, and J. Desrosiers, A column generation approach for large-scale aircrew rostering problems, Oper Res 47 (1999), 247–263.

[14] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W.H. Freeman, San Francisco, 1979, p. 241.

[15] E. Hadjiconstantinou and N. Christofides, An efficient implementation of an algorithm for finding K shortest simple paths, Networks 34 (1999), 88–101.

[16] G. Handler and I. Zang, A dual algorithm for the constrained shortest path problem, Networks 10 (1980), 293–310.

[17] J.M. Jaffe, Algorithms for finding paths with multiple constraints, Networks 14 (1984), 95–116.

[18] H. Joksch, The shortest route problem with constraints, J Math Anal Appl 14 (1966), 191–197.

[19] A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko, Lagrangian relaxation based method for the QoS routing problem, Proc IEEE INFOCOM 2001, Vol. 2, pp. 859–868.

[20] D.E. Kaufman and R.L. Smith, Fastest paths in time-dependent networks for intelligent vehicle-highway applications, IVHS J 1 (1993), 1–11.

[21] T. Korkmaz and M. Krunz, A randomized algorithm for finding a path subject to multiple QoS constraints, Comput Networks 36 (2001), 251–268.

[22] T. Korkmaz and M. Krunz, Multi-constrained optimal path selection, Proc IEEE INFOCOM 2001, Vol. 2, pp. 834–843.

[23] T. Korkmaz, M. Krunz, and S. Tragoudas, An efficient algorithm for finding a path subject to two additive constraints, Comput Commun 25 (2002), 225–238.

[24] F. Kuipers, T. Korkmaz, M. Krunz, and P. Van Mieghem, Performance evaluation of constraint-based path selection algorithms, IEEE Network 18 (2004), 16–23.

[25] J.L. Latourell, B.C. Wallet, and B. Copeland, Genetic algorithm to solve constrained routing problems with applications for cruise missile routing, Proc SPIE 1998, Vol. 3390, pp. 490–500.

[26] S.H.K. Lee, Route optimization model for strike aircraft, Thesis, Naval Postgraduate School, Monterey, California, 1995.

[27] Y. Li, J. Harms, and R. Holte, Fast exact multiconstraint shortest path algorithms, IEEE Int Conf Commun (2007), pp. 123–130.

[28] G. Liu and K.G. Ramakrishnan, An algorithm for finding k-shortest paths subject to multiple constraints, Proc IEEE INFOCOM 2001, Vol. 2, pp. 743–749.

[29] K. Nachtigall, Time depending shortest-path problems with applications to railway networks, Eur J Oper Res 83 (1995), 154–166.

[30] R. Nygaard, G. Melnikov, and A.K. Katsaggelos, A rate distortion optimal ECG coding algorithm, IEEE Trans Biomedical Eng 48 (2001), 28–40.

[31] U. Pape, Implementation and efficiency of Moore-algorithms for the shortest route problem, Math Prog 7 (1974), 212–222.

[32] S. Russell and P. Norvig, Artificial intelligence: A modern approach, Prentice Hall, Upper Saddle River, 1995, pp. 92–107.

[33] P.H. Vance, C. Barnhart, E.L. Johnson, and G.L. Nemhauser, Airline crew scheduling: A new formulation and decomposition algorithm, Oper Res 45 (1997), 188–200.

[34] P. Van Mieghem, H. De Neve, and F.A. Kuipers, Hop-by-hop quality of service routing, Comput Networks 37 (2001), 407–423.

[35] P. Van Mieghem and F.A. Kuipers, Concepts of exact QoS routing algorithms, IEEE/ACM Trans Network 12 (2004), 851–864.

[36] P. Van Mieghem, F.A. Kuipers, T. Korkmaz, M. Krunz, M. Curado, E. Monteiro, X. Masip-Bruin, J. Sole-Pareta, and S. Sanchez-Lopez, "Quality of service routing," Quality of Future Internet Services, COST Action 263 Final Action Report, M. Smirnov, et al. (Editors), Lect Notes Comput Sci, Vol. 2856, Springer, Berlin, 2003, pp. 80–117.

[37] L.A. Wolsey, Integer programming, Wiley, New York, 1998, pp. 172–173.

[38] M. Zabarankin, S. Uryasev, and P. Pardalos, "Optimal risk path algorithms," Cooperative control and optimization, R. Murphey, P. Pardalos, (Editors), Kluwer, Dordrecht, Netherlands, 2001, pp. 273–299.

[39] M. Ziegelmann, Constrained shortest paths and related problems, Naturwissenschaftlich-Technischen Fakultät der Universität des Saarlandes, Germany, 2001.