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, USA
15 March 2007

Abstract

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 new path-enumeration method, aggregate 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 multi-constrained grid networks, road networks, and networks for aircraft routing the advantage varies, but, overall, the new algorithm is competitive with the 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 (e.g. Ahuja et al. [1], pp. 93-157). However, if each edge possesses a non-negative weight in addition to its length, and if a single side constraint is placed on the optimal path's total weight, the problem becomes NP-complete (Garey and Johnson [16], p. 214). When multiple side constraints are included, the general problem is known

as the *constrained shortest-path problem* (CSPP). This paper develops a new algorithm for solving this problem, applies it to several classes of CSPPs, and compares its computational efficiency with a state-of-the-art alternative, the label-setting algorithm of Dumitrescu and Boland [12].

CSPP is NP-complete in the weak sense for a fixed number of side constraints and admits a dynamic-programming solution procedure (Joksch [20]). However, dynamic programming (DP) can be unacceptably slow in practice even with a single side constraint; consequently, label-setting algorithms based on DP have supplanted straightforward DP implementations (e.g., Aneja et al. [2], Desrochers and Soumis [9], and Dumitrescu and Boland [12]). Other potentially useful techniques include branch and bound using a Lagrangian-based bound (Beasley and Christofides [3]), Lagrangian relaxation coupled with K-shortest-paths enumeration (Handler and Zang [18], Juttner et al. [21]), K-shortest-paths enumeration combined with dominance checks (De Neve and Van Mieghem [8], Van Mieghem et al. [33]), and heuristic algorithms (Korkmaz and Krunz [24, 23]); see also the review by Van Mieghem et al. [34].

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., Gamache et al. [15], Day and Ryan [7], Vance et al. [31]). Other important applications include minimum-risk routing of military vehicles and aircraft (e.g., Boerman [4], Latourell, et al. [25], Lee [26], Zabarankin et al. [36]), signal routing in communications networks having quality-of-service guarantees (see Van Mieghem et al. [34] and the references therein), signal compression (Nygaard et al. [29]) and numerous transportation problems (e.g., Nachtigall [28], Kaufman and Smith [22]).

Dumitrescu and Boland [12] describe a label-setting algorithm, combined with several preprocessing techniques, that may be the most efficient technique currently available for CSPP. 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 value of the (current) optimality gap.

LRE resembles the algorithm of Handler and Zang [18], but with a near-shortest-paths (NSP) algorithm replacing their K-shortest-paths algorithm. The LRE approach seems particularly attractive because we (Carlyle and Wood [6]) 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 algorithm. We discuss this issue in more detail later. LRE is also similar to 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 [12], to eliminate edges, *a priori*, that can be proven not to lie on any 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 initially optimizing the Lagrangian function. This feasibility problem is an NP-complete problem in and of itself when multiple side constraints are involved (Garey and Johnson [16], p. 214).

In addition to the theoretical development, we present a computational study of the LRE algorithm applied to CSSPs, 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 of Dumitrescu and Boland [12].

The remainder of the paper 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 [6].) 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 the NSP algorithm as well as incorporating a phase-I routine for finding initial feasible solutions when none is apparent. Finally, we present the 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 $v \in V \backslash \{s, t\}$, and let $\mathbf{g}$ denote the vector $(g_1 g_2 \cdots g_{|I|})^T$. 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. Then, CSPP may be written as this integer program (Ahuja et al. [1], p. 599):

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

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

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

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

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 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 without 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 have cycles in them, and thus this point 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 (e.g., Ahuja et al. [1], pp. 598-648), we know that for any appropriately dimensioned row vector $\boldsymbol{\lambda} \geq \mathbf{0}$,

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

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

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

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

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

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

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

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

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** (e.g., Fisher [13]). And, it is easy to construct examples in which this bound is not very close to 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., 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 (e.g., Wolsey [35], pp. 172-173). But, bisection search works well for a single side constraint (Fox and Landi [14]), as does bisection search applied in the coordinate directions for a few side constraints (e.g., DeWolfe et al. [10]): 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 even 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 given an optimal or near-optimal Lagrangian vector $\boldsymbol{\lambda}$, 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 enumeration. (The theorem is implicit in Handler and Zang [18].)

Theorem 1 *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 enumerating* $\hat{X}(\boldsymbol{\lambda}, \bar{z})$, *and by then selecting*

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

∎

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 exponentially reduce the size of $\hat{X}(\boldsymbol{\lambda}, \bar{z})$, and reduce the computational workload correspondingly.

Theorem 1 and Corollary 1 imply that we may need to enumerate each path $\hat{\mathbf{x}}$ satisfying $(\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$. That is, if $\mathbf{x}^*_{\boldsymbol{\lambda}}$ solves the shortest-path problem given the edge-length vector $\mathbf{c} + \boldsymbol{\lambda}F$ and $\underline{z}(\boldsymbol{\lambda}) = (\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}}^*_{\boldsymbol{\lambda}} - \boldsymbol{\lambda}\mathbf{g}$, then CSPP is solved by enumerating all paths $\hat{\mathbf{x}}$ such that $\underline{z}(\boldsymbol{\lambda}) \leq (\mathbf{c} + \boldsymbol{\lambda}F)\hat{\mathbf{x}} - \boldsymbol{\lambda}\mathbf{g} \leq \bar{z}$. Therefore, given edge-length vector $\mathbf{c} + \boldsymbol{\lambda}F$, and including the Lagrangian constant term $-\boldsymbol{\lambda}\mathbf{g}$ in 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 improve, 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, however (e.g., Hadjiconstantinou and Christofides [17]). Such an algorithm is meant to enumerate the K shortest paths in a network for a pre-specified integer K. But, because it enumerates paths in order of increasing length, it could be halted when path length exceeds $(\mathbf{c} + \boldsymbol{\lambda}F)\mathbf{x}^*_{\boldsymbol{\lambda}} + \epsilon$. However, enumerating paths in order of length requires unnecessary computational work, storage and algorithmic complexity. The NSP algorithm developed by Carlyle and Wood [6] 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

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 [6], 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 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 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 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 Korkmaz and Krunz [24], Liu and Ramakrishnan [27], and Dumitrescu and Boland [12].

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 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 version of "A* search" (e.g., Russell and Norvig [32], pp. 92-107). We also note that Liu and Ramakrishnan [27] use a version of 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 $d(v)$ and $d_i(v)$ represent similar lower bounds for the Lagrangianized path length and ith path weight, respectively. Each label represents a lower bound, rather than an exact value, because each is computed independently, and because the v-t subpath any label represents may include one or more vertices already on $E_P(u)$. In the latter case, the complete path would have at least one cycle, we have ruled out cycles in our definition of CSPP, and thus the label corresponds to a relaxation of CSPP.

The values $d(v)$, $d_0(v)$ 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 (Carlyle and Wood [6]).

On the other hand, solving the shortest-path problems required to maintain precise distance labels can add tremendously to the computational workload. This workload need not be as great as one shortest-path calculation for each type of distance 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 distance labels in our 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 too computationally expensive.

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

4 Algorithmic Enhancements

The basic LRE algorithm can solve many problems quickly, as we will see in Section 5. However, 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 numerous 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 there are fewer vertices and edges that must be processed (e.g., Aneja et al. [2], Dumitrescu and Boland [12]). Importantly, a smaller network may also yield a tighter Lagrangian bound. We use the following preprocessing procedure originally proposed by Aneja et al. [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 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], [12]), 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, computational effort is negligible.)

By its construction, our NSP algorithm automatically performs many of the checks that a preprocessing procedure carries out. However, empirically, we find 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 any 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" (Jaffe [19]).

"LRE-P" will denote a version of the LRE algorithm that incorporates preprocessing Steps 1-3 with respect to individual side constraints and the aggregate constraint; we use only $\boldsymbol{\pi} = \mathbf{1}$. 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 upper bound $\bar{z}$, and we can add the following constraints to the problem:

$$\mathbf{cx} < \bar{z} \tag{14}$$

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

(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 be used in this 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

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

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

Aggregate versions of constraints [A], [B], and [C] may further limit path enumeration. Using empirically determined multipliers $\boldsymbol{\pi}$, we aggregate [C], each pair of [A], [B], and [C], and all three:

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

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

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

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

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

where $\boldsymbol{\pi}_1 = (1/g_1 \ldots 1/g_{|I|})$, $\pi_2 = 1$ and $\pi_3 = 1/\underline{z}(\boldsymbol{\lambda})$. For instance, checking $\pi_3[\text{B}] + \boldsymbol{\pi}_1[\text{C}]$ corresponds to checking whether or not

$$-\pi_3\boldsymbol{\lambda}\mathbf{g} + (\pi_3\mathbf{c} + \pi_3\boldsymbol{\lambda}F + \boldsymbol{\pi}_1F)\mathbf{x} \quad < \quad \pi_3\bar{z} + |I|. \tag{21}$$

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 the various versions of $\boldsymbol{\pi}$ as Lagrangian multipliers that differ from $\boldsymbol{\lambda}$.)

Checking the aggregate constraints while enumerating paths in LRE does add overhead, of course, but empirical results typically show the tradeoff in reduced enumeration to be worthwhile. When reporting computational results in Section 5, "LRE-A" denotes the LRE algorithm with aggregate constraints used to limit enumeration, and "LRE-PA" will denote the use of that along with 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 $\underline{z} = (|V| - 1)c_{\max} + \gamma$, and this 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":

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

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 enhancement only comes into play for problems with more than one side constraint where a feasible solution is not found during the optimization of $\underline{z}(\boldsymbol{\lambda})$.

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 that problem is feasible: After all, it is difficult to find a feasible solution to the original 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. (More refined, generic bounds can be used such as γ plus the sum of the $|V|-1$ shortest edge lengths, but such bounds are still unacceptably weak.)

5 Computational Results

This section reports computational experiments with the LRE algorithm and the label-setting algorithm (LS) of Dumitrescu and Boland [12] applied to problem instances defined on four classes of networks. We have implemented the LS algorithm using a two-heap data structure for labels. And, to facilitate accurate comparison between the two algorithm, we have implemented LS using the same subroutines for preprocessing (see Section 4.1), solving the dual problem $\underline{z}(\boldsymbol{\lambda})$, 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.

We solve instances of CSPP with at most ten side constraints, so repeated bisection searches in the coordinate directions (Fox and Landi [14], DeWolfe et al. [10]) suffice to maximize $\underline{z}(\boldsymbol{\lambda})$ adequately and quickly: We have verified "adequately" by solving the LP relaxation of a number of instances of CSPP using an interior-point algorithm; reported solution times verify "quickly." All versions of LRE employ the shortest-path algorithm of Pape [30] 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 gigabyte of RAM, 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 [12]. 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 $100\%(\bar{z} - \underline{z}^*)/\underline{z}^*$, where $\bar{z}$ is the upper bound found prior to initiating path enumeration. Similarly, column six gives the Lagrangian duality gap, defined as $100\%(z^* - \underline{z}^*)/\underline{z}^*$. For reference, column eight gives run times from Beasley and Christofides. Those computations were performed using FORTRAN on a CDC 7600 computer, and hence, a direct comparison of run times is impossible.

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

Problem	$\|V\|$	$\|E\|$	$\|I\|$	Initial gap (%)	Duality gap (%)	Run Time LRE (sec.)	Run Time B&C (sec.)	Presolve D&B
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

Table 1: Problem statistics and run times for the basic LRE algorithm applied to CSPPs from Beasley and Christofides [3]. Run times on a 3.8 GHz desktop computer exclude problem-generation time; problems are solved to optimality. BC14 is infeasible and the time reported there is for proving infeasibility. Columns five and six report initial optimality gap and duality gap, respectively. (If $\bar{z}$ denotes the objective value for the first feasible solution, then the "initial gap" is $100\% \times (\bar{z} - \underline{z}^*)/\underline{z}^*$, and the "duality gap" is $100\% \times (z^* - \underline{z}^*)/\underline{z}^*$.) 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.. The second-to-last column lists the run time, on a CDC 7600 computer, reported by Beasley and Christofides (B&C). The last column, labeled "Presolve D&B," specifies whether or not Dumitrescu and Boland [12] solve the problem with preprocessing alone.

Weight Limit	Edges pre-processed	Initial gap (%) No pre.	Pre.	Duality Gap (%) No pre.	Pre.	Run Time (sec.) 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

Table 2: Computational results for solving CSPPs to plan the movement of a military convoy through a road network. Problems are solved to optimality. The table reports percentage of edges removed by preprocessing, initial gap, duality gap, and run times for the various versions of LRE and for LS-P. The instance marked with "†" is infeasible because the weight limit of 240 is too small.

Let the weight $f_e = f_{1e}$ associated with edge $e = (u, v)$ represent the time required to traverse road segment e, and let length c_e reflect the risk of being attacked while traversing e. 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, and 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 LRE, with and without various enhancements. The data represent roads in Maryland, Virginia, and Washington, D.C. with speed limits of 30 miles per hour and higher [6]. The resulting graph has 3,670 vertices and 9,876 edges. Road segments with speed limits 65, 55, 50, 45, and 30 miles per hour are categorized as major highways, minor highways, and so on, respectively. Table 2 displays results for a range of hypothesized time limits. Note that it is impossible for the convoy to reach its destination in less than 240 minutes, and no reduction in risk accrues beyond 360 minutes.

The second column of Table 2 displays the percentage of edges that preprocessing re-

moves. 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. For $g = 300$, tightening the lower bound also reduces run time significantly: Compare run times for LRE to those for LRE-P, and compare times for LRE-A and LRE-PA.

A similar 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 [12], 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 for CSPP. 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, with a separate source vertex s and sink vertex t external to the grid. The source s connects to each vertex in the leftmost column of the grid, and each vertex in the rightmost column connects to 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. For each $i \in I$, weight limits

Grid	Weight Limit	$\|V\|$	$\|E\|$	Run Time (sec.) LRE-PA	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

Table 3: Comparison of LRE-PA to the label-setting algorithm, LS-P, of Dumitrescu and Boland [12]. Test problems are singly constrained CSPPs on grid networks of problem class 4-L, Type 2, and problem class 4-M, Type 2 from [12]. Each row represents a single problem instance, solved to optimality. "Speedup" is the apparent improvement of LRE-PA over LS-P, computed as 100% × (LS-P sec. − LRE-PA sec.) / (LS-P sec.)

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 ten singly constrained problems also solved and reported by Dumitrescu and Boland [12]. 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 [11]). The first five come from "problem class 4-L, Type 2" in Dumitrescu and Boland [12], and the second five come from "problem class 4-M, Type 2."

Columns five and six of Table 3 shows run times for LRE-PA and LS-P, respectively; all problems are solved to optimality. On average, LRE-PA solves these problems 82% faster than LS-P (see column seven).

Grid	Weight Limit	Run Time (sec.) LRE-PA avg.	LRE-PA s.d.	LS-P avg.	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	$40.7^{\dagger}$	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

Table 4: Comparison of run times for LRE-PA and LS-P when applied to randomly-generated, singly constrained CSPPs on grid networks. Problems are solved to optimality. The table reports averages ("avg.") and standard devations ("s.d."), over 20 instances for each problem type. One problem instance in the group marked by "†" takes 789 seconds to solve.

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). There, the algorithm finds the optimal solution and proves it to be within 0.5% of optimality in 0.3 seconds, but requires 789 seconds 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.

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 [12] do not solve any multi-constrained 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 might be preferred over the other in some 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 minutes, along with the average run time and standard deviation for the successfully solved problems. Since the averages and standard deviations are computed only over the solved problems, these statistics must be considered in view of the number of problems actually solved. 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 within the time limit of 30 minutes, respectively. 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. Table 5 does show some large standard deviations in run times for both LS-P and LRE-PA, however, which indicates that 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 LS-P and LRE-PA can now solve more problems within the time limit: Now, LS-P achieves 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 on problems with the high (H) weight limit instead of medium. 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 is faster. 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

Grid	Statistics	$\|I\|$=2		$\|I\|$=3		$\|I\|$=4		$\|I\|$=5		$\|I\|$=10	
		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

Table 5: Run-time statistics for LRE-PA and LS-P solving multi-constrained CSPPs on grid networks with $|I|$ side constraints and with medium (M) weight limits. Problems are solved to optimality. The table reports the average (avg.) and standard deviation (s.d.) of the run times over 20 randomly generated instances for each grid size. "No. solved" indicates the number of instances solved within 30 minutes. Only instances solved within 30 minutes are included in the average and standard deviation calculations.

Grid	Statistics	$\|I\|$=2		$\|I\|$=3		$\|I\|$=4		$\|I\|$=5		$\|I\|$=10	
		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 6: Run-time statistics for solving the same CSPPs as in Table 5, except that the optimality tolerance is 1% here.

Grid	Statistics	$\|I\|$=2		$\|I\|$=3		$\|I\|$=4		$\|I\|$=5		$\|I\|$=10	
		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

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). (Problems are solved to optimality.)

increase to 1%. In this case LS-P and LRE-PA perform equally well, with 99.8% of the problem instances solved within 30 minutes.

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 five seconds 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 solved within 30 minutes over all grid sizes. The total number of instances for each case is 100. (5 network sizes $\times$ 20 randomly generated instances = 100 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 and a 0% optimality tolerance, and see $|I|$=10 with high weight limits and a 0% optimality tolerance).

Grid	Statistics	$\|I\|$=2		$\|I\|$=3		$\|I\|$=4		$\|I\|$=5		$\|I\|$=10	
		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 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%.

Algorithm	Weight Limit	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

Table 9: Summary run-time statistics for computational study of multi-constrained grid networks (from Tables 5-8). This table reports total number of problem instances solved (out of 100 instances) within 30 minutes over all grid sizes as well as total percentage solved over all grid sizes and number of side constraints.

5.4 Routing Aircraft on Dense Network

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 are differ substantially from the grid networks examined in Section 5.3, which are sparse and 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. 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 F/A-18 and similar aircraft will maintain a constant and 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 nm by 296 nm with the southwest corner being the origin in a Cartesian coordinate system, measured in nautical miles. We assume that the strike group enters the AO at its western edge, at x-y coordinates (0,104), and 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 eight different edge structures (A-H in Table 10), which 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 all vertices v that are between 8 nm and 12 nm away, but only those that are no further west than u. (The general east-to-west travel of the strike group makes westward travel unlikely, so none of the structures contain edges with a west-bound vector components.) 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.

		Edge length			Run times for various fuel limits g					
Struct.	$\|E\|$	min	max	Algo.	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

Table 10: Run-time statistics for solving aircraft-routing CSPPs with various fuel constraints and network structures. All problems are solved to optimality. Each vertex u is connected with edges (u, v) where v lies between "min edge" and "max edge" nautical miles distant, but is no further west than the tail vertex. The last six columns specify the run times for LRE-PA, LRE-PA with reprocessing, denoted "LRE-PAR," and LS-P.

The last six columns of Table 10 show the run times for LS-P, LRE-PA, and LRE-PA with "reprocessing" (LRE-PAR), which will be 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 seconds: This problem instance 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 seconds: This instance 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 scan of reprocessing, applied each time the upper bound improves, can reduce run times significantly over LRE-PA. However, reprocessing does add overhead, and it may reduce the network only modestly, or not all, when executed. We find it more efficient to execute a reprocessing scan 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 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. (However, reprocessing does not reduce run times substantially in the grid network problems of Section

5.3, because those problems have small duality gaps and few updates of the upper bounds occur.) In principle, LS-P could also incorporate reprocessing, but the effect is likely to be minimal, or even deleterious, because the run times for LS-P are modest to begin with here, and do not exhibit the variability seen with LRE-PA.

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 can also be extended to "cannot lie on any optimal path" if the preprocessing mechanism identifies a feasible solution.) 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. Executing additional preprocessing scans within the algorithm's enumeration phase (called "reprocessing") to take advantage of improving upper bounds from improving feasible solutions, and

3. When the process of optimizing the Lagrangian lower bound does not yield a 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.

The first two enhancements, even when not required, do not add significantly to computational overhead, so we recommend them as standard additions to the basic algorithm. The third enhancement may involve substantial computational effort but computational tests show that that effort is almost always worthwhile when an initial feasible solution is not immediately 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-minute time limit. However, LS does solve some problems substantially faster than LRE, especially in the presence of large duality gaps.

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), (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 pass through exactly one vertex in some easily identifiable subset of vertices $V' = \{v_1, v_2, \ldots, v_k\}$. Then, the CSPP's solution may be found by solving, for $i = 1, \ldots, k$, the presumably simpler CSPPs defined on G with vertices $V' \backslash \{v_i\}$ deleted. (In fact, such a decomposition reduces LRE's 40.7 second average solution time in Table 4, indicated by "†," to less than four seconds.)

More complicated ideas may prove useful, too. In particular, we believe that 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 (c) 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.

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 possibile variants and hybrids is enormous.

Acknowledgments

The authors thank Professor Irina Dumitrescu for providing data for some of our computational tests. 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 paper.

References

[1] Ahuja, R.K., Magnanti, T.L., and J.B. Orlin, (1993), *Network Flows*, Prentice Hall, Englewood Cliffs, New Jersey.

[2] Aneja, Y., Aggarwal, V., and Nair, K., (1983), "Shortest Chain Subject to Side Conditions," *Networks*, 13, pp. 295–302.

[3] Beasley, J. and Christofides, N., (1989), "An Algorithm for the Resource Constrained Shortest Path Problem," *Networks*, 19, pp. 379–394.

[4] Boerman, D., (1994), "Finding an Optimal Path Through a Mapped Minefield," Master's Thesis, Naval Postgraduate School, Monterey, California, March.

[5] Byers, T.H. and Waterman, M.S., (1984), "Determining Optimal and Near-Optimal Solutions When Solving Shortest Path Problems by Dynamic Programming," *Operations Research*, 32, pp. 1381–1384.

[6] Carlyle, W.M. and Wood, R.K., (2003), "Near-Shortest and K-Shortest Simple Paths," *Networks*, 46, pp. 98–109.

[7] Day, P.R. and Ryan, D.M., (1997), "Flight Attendant Rostering for Short-Haul Airline Operations," *Operations Research*, 45, pp. 649–661.

[8] De Neve, H. and Van Mieghem, P., (2000), "TAMCRA: A Tunable Accuracy Multiple Constraints Routing Algorithm," *Computer Communications*, 23, pp. 667–679.

[9] Desrochers, M. and Soumis, F., (1988), "A Generalized Permanent Labeling Algorithm for the Shortest Path Problem with Time Windows," *Information Systems and Operational Research*, 26, pp. 191–212.

[10] DeWolfe, D., Stevens, J. and Wood, K., (1993), "Setting Military Reenlistment Bonuses," *Naval Research Logistics*, 40, pp. 143–160.

[11] Dumitrescu, I., (2005), Private communication, 20 April 2005.

[12] Dumitrescu, I. and Boland, N., (2003), "Improved Preprocessing, Labeling and Scaling Algorithm for the Weight-Constrained Shortest Path Problem," *Networks*, 42, pp. 135–153.

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

[14] Fox, B.L. and Landi, D.M., (1970), "Searching for the Multiplier in One-Constraint Optimization Problems," *Operations Research*, 18, pp. 253–262.

[15] Gamache, M., Soumis, F., Marquis, M. and Desrosiers, J., (1999), "A Column Generation Approach for Large-Scale Aircrew Rostering Problems," *Operations Research*, 47, pp. 247–263.

[16] Garey, M.R. and Johnson, D.S., (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco.

[17] Hadjiconstantinou, E. and Christofides, N., (1999), "An Efficient Implementation of an Algorithm for Finding K Shortest Simple Paths," *Networks*, 34, pp. 88–101.

[18] Handler, G. and Zang, I., (1980), "A Dual Algorithm for the Constrained Shortest Path Problem," *Networks*, 10, pp. 293–310.

[19] Jaffe, J.M., (1984), "Algorithms for Finding Paths with Multiple Constraints," *Networks*, 14, pp. 95–116.

[20] Joksch, H., (1966), "The Shortest Route Problem with Constraints," *Journal of Mathematical Analysis and Application,* 14, pp. 191–197.

[21] Juttner, A., Szviatovszki, B., Mecs, I., and Rajko, Z., (2001), "Lagrangian Relaxation Based Method for the QoS Routing Problem," *Proceedings of the IEEE INFOCOM 2001 Conference, Vol. 2*, Anchorage, Alaska, 22-26 April, pp. 859–868.

[22] Kaufman, D.E. and Smith, R.L., (1993), "Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Applications," *IVHS Journal*, 1, pp. 1–11.

[23] Korkmaz, T. and Krunz, M., (2001), "A Randomized Algorithm for Finding a Path Subject to Multiple QoS Constraints," *Computer Networks*, 36, pp. 251–268.

[24] Korkmaz, T. and Krunz, M., (2001), "Multi-Constrained Optimal Path Selection," in *Proceedings of the IEEE INFOCOM 2001 Conference, Vol. 2*, Anchorage, Alaska, April 22-26, pp. 834–843.

[25] Latourell, J.L., Wallet, B.C. and Copeland, B., (1998), "Genetic Algorithm to Solve Constrained Routing Problems with Applications for Cruise Missile Routing," in *Applications and Science of Computational Intelligence*, *Proceedings of SPIE, Vol. 3390*, Society of Photo-optical Instrumentation Engineers, Bellingham, Washington, USA, pp. 490–500.

[26] Lee, S.H.K., (1995), "Route Optimization Model for Strike Aircraft," Master's Thesis, Naval Postgraduate School, Monterey, California, September.

[27] Liu, G. and Ramakrishnan, K.G., (2001), "An Algorithm for Finding k-Shortest Paths Subject to Multiple Constraints," in *Proceedings of the IEEE INFOCOM 2001 Conference, Vol. 2*, Anchorage, Alaska, April 22-26, pp. 743–749.

[28] Nachtigall, K., (1995), "Time Depending Shortest-path Problems with Applications To Railway Networks," *European Journal of Operational Research*, 83, pp. 154–166.

[29] Nygaard, R., Melnikov, G. and Katsaggelos, A.K., (2001), "A Rate Distortion Optimal ECG Coding Algorithm," *IEEE Transactions on Biomedical Engineering,* 48, pp. 28–40.

[30] Pape, U., (1974), "Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem," *Mathematical Programming*, 7, pp. 212–222.

[31] Vance, P.H., Barnhart, C., Johnson, E.L. and Nemhauser, G.L., (1997), "Airline Crew Scheduling: A New Formulation and Decomposition Algorithm," *Operations Research*, 45, pp. 188–200.

[32] Russell, S. and Norvig, P., (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall, Upper Saddle River, NJ.

[33] Van Mieghem, P., De Neve, H., and Kuipers, F.A., (2001), "Hop-by-Hop Quality of Service Routing," *Computer Networks*, 37, pp. 407–423.

[34] Van Mieghem, P., Kuipers, F.A., Korkmaz, T., Krunz, M., Curado, M., Monteiro, E., Masip-Bruin, X., Sole-Pareta, J., and Sanchez-Lopez, S., (2003), "Quality of Service Routing," in *Quality of Future Internet Services, COST Action 263 Final Action Report*, M. Smirnov et al., eds., Lecture Notes in Computer Science, Vol. 2856, Springer, Berlin, Germany, pp. 80–117.

[35] Wolsey, L.A, (1998), *Integer Programming*, John Wiley & Sons, New York.

[36] Zabarankin, M., Uryasev, S. and Pardalos, P., (2001), "Optimal Risk Path Algorithms" in *Cooperative Control and Optimization*, R. Murphey and P. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, pp. 273–299.

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

{
  $\mathbf{c}' \leftarrow \mathbf{c} + \boldsymbol{\lambda}F$;
  /* Add a "0-th side constraint" to limit enumeration based on $\mathbf{c}$ */
  $I^+ \leftarrow I \cup \{0\}$; $\mathbf{f}_0 \leftarrow \mathbf{c}$; $g_0 \leftarrow \bar{z}$;
  /* The following requires just one backwards shortest-path calculation */
  for (all $v \in V$) { $d(v) \leftarrow$ minimum distance, in terms of $\mathbf{c}'$, from v to t; }
  for (each side constraint $i \in I^+$){
    /* Solve a backwards shortest-path problem using edge "lengths" $\mathbf{f}_i$ */
    for (all $v \in V$) { $d_i(v) \leftarrow$ minimum weight, in terms of $\mathbf{f}_i$, from v to t; }
  }
  for(all $v \in V$) { nextEdge(v) $\leftarrow$ firstEdge(v); }
  $L(s) \leftarrow -\boldsymbol{\lambda}\mathbf{g}$; /* Initialize path length with the Lagrangian constant term */
  for(all $i \in I^+$) { $L_i(s) \leftarrow 0$; } /* Initial path weight with respect $\mathbf{f}_i$ is 0 */
  theStack $\leftarrow s$; onStack(s) $\leftarrow$ true; onStack(v) $\leftarrow$ false $\forall\ v \in V \backslash \{s\}$;
  while (theStack is not empty){
    $u \leftarrow$ vertex at the top of theStack;
    if (nextEdge(u) $\neq \emptyset$) {
      $e \leftarrow$ the edge pointed to by nextEdge(u); /* $e = (u, v)$ */
      increment nextEdge(u);
      if ((onStack(v) = false) and ($L(u) + c'_e + d(v) < \bar{z} - \delta$)
        and ($L_i(u) + f_{ie} + d_i(v) \leq g_i\ \forall\ i \in I^+$)) {
        if ($v = t$) { /* An improved solution has been found */
          Represent the feasible path encoded as theStack $\cup\ \{t\}$ through
          its edge-incidence vector $\hat{\mathbf{x}}$;
          $\bar{z} \leftarrow \mathbf{c}\hat{\mathbf{x}}$; $g_0 \leftarrow \bar{z}$; $\mathbf{x}^* \leftarrow \hat{\mathbf{x}}$;
          /* Preemptive termination is possible in the following step */
          if ($\bar{z} - \underline{z}(\boldsymbol{\lambda}) \leq \delta$) **goto** Finish;
        } **else** {
          push v on theStack; onStack(v) $\leftarrow$ true;
          $L(v) \leftarrow L(u) + c_e$;
          for (all $i \in I^+$) { $L_i(v) \leftarrow L_i(u) + f_{ie}$; }
        }
      }
    } **else** {
      Pop u from theStack; onStack(u) $\leftarrow$ false;
      nextEdge(u) $\leftarrow$ firstEdge(u);
    }
  }
  Finish: If $\mathbf{x}^*$ is empty Print (Problem is infeasible), otherwise Print ($\mathbf{x}^*$);
}