CORC REPORT 2003-10

A Scalable Algorithm for the Minimum Expected Cost Restorable Flow Problem

Lisa Fleischer, Adam Meyerson, Iraj Saniee, Bruce Shepherd, Aravind Srinivasan*

Presented at the DIMACS Mini-Workshop on Quality of Service Issues in the Internet, February 2001

Abstract

We introduce a model for the computation of a multicommodity flow together with back-up flows for each commodity after any of a given set of failure states occurs in a network. We call such a total routing strategy a *restorable multicommodity flow*. One advantage of restorable flows over classical network protection methods, such as disjoint paths, is that the commodities may effectively share capacity in the network for their back-up flows. We consider the problem of finding a minimum expected-cost restorable flow, under a given probability distribution on network element failures. The underlying stochastic optimization problem can be modelled as a large-scale linear programming (LP) problem that explicitly incorporates the network failure scenarios. The size of the LPs grows swiftly, however, in the size of network and number of failure states. Our focus is on developing a scalable combinatorial algorithm for this problem. This leads us to specialize the problem to the case where traffic flows for each commodity are restricted to a (pre-computed) collection of disjoint paths. This restriction also makes it easier to restore traffic after network failures; specifically, restoration can be performed by the commodity's endpoints without rerouting any of the traffic which was not disturbed by the network failure. In this setting, devise an approximation scheme for the feasibility problem that is based only on repeatedly pushing flow along cheapest pairs of paths. The algorithms are easily described and readily implementable. We give computational results relating how our approach scales for large networks in comparison with general-purpose LP solvers. An earlier version of this paper, with identical results, appeared in [7].

*L. Fleischer is with the Graduate School of Industrial Administration, Carnegie-Mellon University, *lkf@andrew.cmu.edu*. A. Meyerson is with the Dept. of Computer Science, Carnegie Mellon University, *awm@cmu.edu*. I. Saniee, B. Shepherd are with Bell Labs, Lucent Technologies, {*iis,bshep*}*@research.bell-labs.com*. Aravind Srinivasan is with the Dept. of Computer Science and Institute for Advanced Computer Studies, University of Maryland at College Park, *srin@cs.umd.edu*; his work was done mainly while at Bell Labs, Lucent Technologies.

Keywords – **Network design, routing and restoration, network flows, linear programming, approximation scheme.**

1 Introduction

There is considerable demand for network design and routing strategies which are resilient to failures, or attacks, on a network infrastructure. At the same time, operators are focused more than ever on achieving the highest utilization of their network's existing bandwidth. Traditional resilient capacity allocation strategies, such as in SONET rings, require that total capacity is essentially doubled. For instance in $1 + 1$ routing, for each (unit/wavelength) demand there is capacity reserved on two disjoint (or diverse) paths between the source and destination. The first is used for the primary flow, and the second is the backup path in case of failures on the primary path. Even in the case where traffic can not be split over multiple paths, this is an expensive alternative to guarantee the ability to recover traffic flows after a network failure. This is because primary flow paths may themselves be disjoint, and so the failure of a single network element would only affect one of them at a time. This suggests the paradigm of sharing of back-up capacity. This has received more attention recently, for instance, in [11] it is shown that (under high congestion) 60% of the network's capacity is used for the back-up flows in the $1+1$ setting, whereas only 40% is used in the shared capacity model.

We consider a model for the design of resilient traffic routing which follows this principle of shared backup capacity. Specifically, we propose an extension to minimum cost multicommodity flow where we are given a collection of failure states and for each state, we compute a multicommodity flow which will be used in the event of this failure. We call such a collection of flows a *restorable flow*.

There are three problems related to restorable flows. The first is that of feasibility: given a capacitated network does it admit such a restorable flow for a given set of demands. The second is more a network design problem. Compute a minimum cost subnetwork (without integrality constraints) which admits a restorable flow. The third is a minimum cost restorable flow problem. In order to give this meaning, we must decide on a measure of cost for a restorable flow. In any case, it is the second problem and not the third, which exploits shared backup capacity since we may pay less for the designed subnetwork by sharing the "extra" capacity used by the backup flows. This is in contrast with the minimum cost flow version, where the backup flows pay individually on a per-unit of traffic flow basis.

The cost function we adopt for the mincost restorable flow problem is designed to approximate the expected long-term cost of running the network using a restoration strategy based on the restorable flow. Specifically, we are given a set of network failure scenarios which are to be addressed. We assume that we are also equipped with probability distributions for the failure of network elements (e.g., the network links), from historical records for instance. These then induce probabilities of occurrence for each of the failure events. Our objective

is to find a restorable flow which minimizes the expected long-term operational cost of the network (under certain independence assumptions).

The resulting stochastic optimization problem can be formulated as a linear program which grows dramatically with the increase in network size and number of failure scenarios being considered. Thus, as with many optimization problems for routing and maximal sharing of protection paths [18], this LP is sometimes too large to solve in practice for realistic-sized networks. Furthermore, we have had a need to carry out such optimizations repeatedly in operational networks. We thus address the need for fast and scalable approximations with a *guaranteed* performance. One present aim is to design simple, efficient and scalable combinatorial algorithms for the restorable flow problems. To achieve this, we introduce a further restrictions on our model. In particular, we assume that the flow for each commodity is routed along paths in a given pre-computed collection of disjoint paths for that commodity. Flows with this additional structure also appear to be operationally simpler with regard to rerouting flow at the time of restoration after a failure. The initial collections of disjoint paths may themselves be part of a preprocessing phase which, say, tries to solve the primary problem so as to minimize the maximum load, or overall load. Our algorithms are based on adapting recent ϵ-approximation algorithms for multicommodity flow [2, 8, 10] and more generally mixed packing and covering linear programs. Our methods yield algorithms for the feasibility and mincost restorable flow problems, however, they do not yield an approximation scheme for the network design version. Ref Young?

In summary, our main contributions are as follows. We introduce a model for computing shared back-up capacity for multicommodity flow that prescribes restoration routing strategies for each scenario in a given collection of failure events. The model takes as input, probability distributions for failures of the network elements and attempts to minimize the expected long-term cost of following the restoration strategies. We design a polynomial time approximation algorithm for this problem which for a given tolerance parameter $\epsilon > 0$, produces a solution whose cost is at most $(1 + \epsilon)$ times the optimum. Correctness of our algorithms follows similar lines to previous schemes such as [10], but the minimum cost version requires a different auxiliary LP (as opposed to a shortest path or column problem) to be solved at each iteration; this necessitates slightly more involved arguments in the proof. We have implemented the algorithms and present some of our basic computational findings. In particular, in contrast to general LP solvers, we find that the algorithms scale well as a function of problem size.

The organization of the paper is as follows. In Section 2, a model for restorable flows is described. In Section 3, we give some background in approximation schemes for linear programs. As a warmup to our approach, Section 4 considers a simplified version of this problem and develops a provably good ϵ-approximation algorithm for it. Section 5 presents the main algorithm. Section 6 shows how the currently-used dedicated protection mechanisms are a special case of our general model (and hence are solvable by our algorithms). We tabulate the results of implementing our scheme in Section 7. Conclusions

and directions for further research are summarized in Section 8.

2 The Model

Broadly, our setting of shared protection is as follows. We consider a network $G = (V, E)$ with a capacity $u(e)$ and cost $c(e)$ for each link or edge e; G is modeled as an undirected graph. We are also given a set of commodities (demand-pairs) K, each commodity $i \in K$ specified by a source-sink pair (s_i, t_i), demand d_i, and a collection Λ_i of pairwise link-disjoint paths, each of which connects s_i to t_i. (That is, no two paths in Λ_i have any common link.) Any single link of G may fail at any time, thus rendering all paths that pass through it temporarily useless, until the link is made live again.

Our objective is to design working and restoration flows so that under prescribed failure states, we have adequate capacity to meet our given demand. Moreover, we wish to minimize the long-term operational cost of these traffic flows. To do this, we assume statistical information on the relative likelihoods of failure of the various links, and consequently of paths in $\Lambda := \bigcup_{k \in K} \Lambda_k$ (a path fails if any of its links fails). This information may be gathered and updated as further failures occur (with old information discounted in order to have an accurate picture of the existing network). For each commodity (s_k, t_k), we must choose (a) appropriate primary flows on paths in Λ_k, and (b) how much flow is transferred from one path to another in the event of a failure. This important requirement that there be sufficient flow even under any link failure, is formalized below.

Under the normal state of no failure, as well as under the failure of any single link, no link e's capacity should be exceeded: the total demand using it should be at most $u(e)$. The objective function is to design the paths, flows, and restoration strategies, in order to minimize the long-term average cost of operating the network. This average is just the expected long-term cost of operating the network under a given collection of primary and backup paths, where the random variables are the failures of the various paths, each of whose probabilities are known as discussed above.

We now model the problem more formally. We consider a set of network states Q. Different states arise due to failures of certain network elements; we assume a state $q_0 \in Q$ which is the normal state when no failures have occurred. In our setting above, q_0 corresponds to the state of no link failure, and every other $q \in Q$ is in one-to-one correspondence with the failure of some subset of links in the network. If we are considering only single-link failure sets, then $|Q| = |E| + 1$.

We now formulate the minimum restorable flow problem. Variable $x(P)$ denotes the amount of flow on path P under state q_0. For each commodity k, and each pair of distinct paths $P', P \in \Gamma_k$, variable $y^{P'}(P)$ denotes the amount of flow that is rerouted from path P' to path P when a link on path P' fails. Such a pair (x, y) is called a *restorable flow*. When failure q occurs, the set of affected paths is denoted by $\Lambda(q)$ (so $\Lambda(q_0) = \emptyset$). Let $Q_k \subseteq Q$ denote the

set of all failures that affect some path in Λ_k. Our formulation assumes that $|\Lambda_k \cap \Lambda(q)| \leq 1$ for all k, q. That is, we assume that a failure does not affect more than one path in Λ_k. Note that this is the case if we consider only single-link failures. In fact, our assumption must be somewhat stronger than this. We are assuming that the probability of failure of a path due events not considered in Q is zero. It is sufficient that the probability of such events is relatively small. In essence, we are saying that we are measure the cost of our network only over periods of time for which it is in some state given in Q.

For each path $P \in \Lambda$, let $\kappa(P)$ denote the steady-state proportion of time for which P is in non-failed mode. (As mentioned above, we could continuously learn and update estimates on these probabilities.) To model the objective function, we assign, for each commodity k and paths $P_1, P_2 \in \Lambda_k$, a cost $c(P_1, P_2)$ as follows: $c(P, P) = \kappa(P) \sum_{e \in P} c(e)$, and if $P_1 \neq P_2$, $c(P_1, P_2) = (1 - \kappa(P_1)) \sum_{e \in P_2} c(e)$. Thus, $c(P, P)$ is the (long-term) expected cost of primary flow on path P, and $c(P_1, P_2)$, for $P_1 \neq P_2$, is the (long-term) expected cost of backup flow on P_2 from P_1. Henceforth we set $c(P) := c(P, P)$. So, by the linearity of expectation, our objective function of long-term average cost is:

$$EC(x, y) = \sum_k \sum_{P \in \Lambda_k} [c(P)x(P) + \sum_{P_1 \in \Lambda_k \setminus P} c(P_1, P) y^{P_1}(P)]. \quad (1)$$

We can now formulate the *minimum expected long-term cost restorable flow problem* (MELTCoRe) as the LP:

$$\min \ EC(x, y)$$

subject to:

- $\sum_{P \in \Lambda_k} x(P) \geq d_k, \forall\ k \in K$
- $\sum_{P \in \Lambda_k \setminus P'} [x(P) + y^{P'}(P)] \geq d_k, \ \forall\ P' \in \Lambda_k, \ \forall\ k \in K$
- $\sum_{k \in K} \sum_{\substack{P: e \in P \\ P \in \Lambda_k \setminus \Lambda(q)}} [x(P) + \sum_{P' \in \Lambda_k \cap \Lambda(q)} y^{P'}(P)] \leq u(e) \ \forall\ e \in E, \ \forall\ q \in Q$
- $x, y \geq 0$.

The first constraint above says that under no-failure conditions, the total demand d_k should be met for each commodity k. The second constraint says that this demand-satisfaction should hold even if any path $P' \in \Lambda_k$ fails. The third constraint is that under any network state q (including the no-failure state q_0), the total flow on any link e should be at most its capacity. We also have the final trivial constraints which force all variables to be non-negative.

We do not develop an approximation scheme for the network design version for restorable flows. We note that it is formulated as a linear program where for

each edge e we introduce an integer variable $z(e) \leq u(e)$. The objective function for the design problem is then $\sum_e c(e)z(e)$. Normally, network design problems would suggest that the variables $z(e)$ be integer. We use the term 'design' here nevertheless, since we are constructing a single (fractional) subnetwork which is meant to support multiple multicommodity flow vectors (one for each failure state).

3 Solving Difficult Linear Programs

The linear program (MELTCoRe) is very large even for moderate-sized networks and a moderate number of failure scenarios. General-purpose LP codes may thus be too slow to solve it. (We shall see later some instances where our scheme works, while commercial LP solvers do not.) Moreover, if there are concurrent users running the optimizations, many copies of general-purpose LP software may need to be licensed. These are two motivations for adapting approximate LP techniques to obtain provably good solutions to difficult linear programs.

Our techniques use and build on previous work [8, 10] on finding such approximation algorithms for multicommodity flow problems. The common setting for these algorithms is that they repeatedly apply flow pushing operations to incrementally build a multicommodity flow which itself is not feasible. This "super-flow" becomes feasible, however, when suitably scaled down; moreover, this scaled-down flow is near-optimal. The number of "pushes" performed per demand depends on the predetermined error bound $\epsilon > 0$. Naturally, the smaller the ϵ, the more the number of pushes required. The flow-pushing steps themselves are essentially weighted shortest-path computations. This makes these algorithms attractive also from an implementation point of view.

We remark that the weights used for the shortest path subproblems change dynamically as the super-flow is being constructed. In particular, during a single push operation, the edge weights are increased exponentially in the flow being pushed on an edge. A similar exponential cost strategy was used by Shahrokhi and Matula [19] to devise ϵ-approximation algorithms for the maximum concurrent flow problem, which we discuss later. A central achievement in the area of approximation algorithms for difficult LPs was due to Plotkin, Shmoys and Tardos [16] as well as Grigoriadis and Khachiyan [13]. They devised such algorithms for the much more general class of *packing and covering linear programs*.

Computational experience for these methods can be found in [2, 12]. For a more complete discussion of these techniques and some of their origins, the reader is referred to [3].

4 Maximum Concurrent Restoration

4.1 An equivalent problem

We do not work directly with the formulation (MELTCoRe). Instead, we consider the maximum concurrent flow version with budget constraint: find the

largest value λ such that at least λd_k demand can be shipped from s_k to t_k for each k, subject to our given constraints, *as well as* a budget constraint that requires the value $EC(x, y)$ (see (1)) to be at most some parameter B. With an ϵ-approximate algorithm for this new problem, we can obtain an ϵ-approximate solution for our actual problem by using an efficient search method (such as binary search) to find the smallest value of B for which this new problem has a solution of value $\lambda \geq 1$. An important practical improvement can be obtained by a careful implementation of such a search method, as demonstrated by our third table in Section 7. Instead of doing a routine binary search, we implement a natural interpolated binary search. If a value B_0 for B returns a value $\lambda_0 > 1$ for this new problem, it can be shown that the optimal B is at most B_0/λ_0; conversely, if a value B_1 for B returns a value $\lambda_1 < 1$, the optimal B is at least B_1/λ_1. We use a slight refinement of this idea to carefully choose successive guesses for B, in order to quickly converge to a value of B that returns $\lambda \sim 1$.

The above problem of maximizing λ subject to a budget B can also be formulated as an LP as follows; we refer to it as the *Budget Constrained Concurrent Restorable Flow Problem* (BCRF).

$$\max \ \lambda$$

subject to:

- $\sum_{P \in \Lambda_k} x(P) \geq \lambda d_k, \forall \ k \in K$
- $\sum_{P \in \Lambda_k \setminus P'} [x(P) + y^{P'}(P)] \geq \lambda d_k, \ \forall \ P' \in \Lambda_k, \ \forall \ k \in K$
- $\sum_{k \in K} \sum_{\substack{P: e \in P \\ P \in \Lambda_k \setminus \Lambda(q)}} [x(P) + \sum_{P' \in \Lambda_k \cap \Lambda(q)} y^{P'}(P)] \leq u(e) \ \forall \ e \in E, \ \forall \ q \in Q$
- $\sum_{k \in K} \sum_{P \in \Lambda_k} [c(P)x(P) + \sum_{P' \in \Lambda_k \setminus P} c(P', P) y^{P'}(P)] \leq B$
- $x, y, \lambda \geq 0.$

4.2 The Concurrent Restorable Flow Problem

We start by considering the simpler case of just determining feasibility. This *Concurrent Restorable Flow Problem* (CRF) is obtained from (BCR) by dropping the budget constraint. This simpler "infinite budget" case provides much of the motivation for our main algorithm of Section 5. The dual to the linear program for (CRF) is

$$\min \sum_{e \in E} u(e) \sum_{q \in Q} h^q(e) \text{ subject to:}$$

- $\sum_{q: P' \in \Lambda(q)} \ \sum_{e \in P} h^q(e) \geq w^{P'}, \ \forall \ k \in K, \ \forall P' \in \Lambda_k, P \in \Lambda_k - \{P'\}$

- $\sum_{q:P\notin\Lambda(q)} \sum_{e\in P} h^q(e) \geq z_k + \sum_{P'\in\Lambda_k-\{P\}} w^{P'}, \forall\ P \in \Lambda_k,\ \forall\ k \in K$
- $\sum_{k\in K} d_k z_k + \sum_{k\in K} \sum_{P'\in\Lambda_k} d_k w^{P'} \geq 1$
- $h, w, z \geq 0$.

To get further intuition about this dual LP, we define new dual variables $Z_k = z_k + \sum_{P'\in\Lambda_k} w^{P'}$. Intuitively, the dual variable $h^q(e)$ represents the marginal cost of link e when the network is in state q. Following this intuition, $w^{P'}$ is the cost of the cheapest backup path if path P' fails. This means that Z_k is the cost of the cheapest backup path for commodity k, including in this the cost of the appropriate backup path in case of failure. We can thus rewrite the dual linear program as follows:

$$\min \sum_{e\in E} u(e) \sum_{q\in Q} h^q(e)$$

subject to:

- $\sum_{q:P'\in\Lambda(q)} \sum_{e\in P} h^q(e) \geq w^{P'}, \forall\ k \in K,\ \forall\ P \neq P', P' \in \Lambda_k, P \in \Lambda_k$
- $w^P + \sum_{q:P\notin\Lambda(q)} \sum_{e\in P} h^q(e) \geq Z_k, \forall\ P \in \Lambda_k,\ \forall\ k \in K$
- $\sum_{k\in K} d_k Z_k \geq 1$
- $h, w, Z \geq 0$.

4.3 Algorithm

We now present an ϵ-approximation for the concurrent flow problem without the budget constraint, that we are considering for now.

The algorithm cycles through the commodities. For each visit to commodity k, the algorithm pushes d_k flow along paths in Λ_k. This pushing is done on a "cheapest" pair of paths P_1, P_2: a pair for which the primary flow on P_1 plus restoration flow on P_2 is minimized. The amount u pushed on the pair is determined by the minimum of three quantities as follows. Let $u(P)$ denote the *bottleneck* capacity of P: $u(P) := \min_{e\in P} u(e)$. Then u is set equal to $\min\{u(P_1), u(P_2), d_k\}$. The algorithm then updates the primal and dual variables x, y, h as described in the algorithm below. The algorithm repeats this process until d_k units of flow have been pushed and then proceeds to the next commodity. (We comment that in practice, we start the algorithm by scaling all capacities so that $\max_k d_k \leq \min_e u(e)$. Thus, u is in effect always equal to d_k, and the algorithm sends flow exactly once per commodity per iteration.) The

algorithm stops when an ϵ-approximate solution is found (with ϵ-approximate dual solution as a witness to the fact).

We now delve into some of the details, particularly with regard to how the algorithm updates the dual variables after each pushing of flow. (Please note that we use the symbol e below to denote a link in the network, as well as to denote the base of the natural logarithm. The latter usage is always of the form e^t for some expression t, so we hope that this abuse of notation is not confusing.) Let $\vec{0}$ denote a vector of all zeroes (the dimension of such a vector varies depending upon the context). We start with a primal solution $x = y = \vec{0}$ and dual solution $h^q(e) = \delta/c(e)$, for an appropriately small δ to be given later in (8) as a function of ϵ, m, and $|Q|$. Given any vector h, we obtain w by setting $w^{P'}$ equal to the minimum over all possible backup paths P for P' of $\sum_{q:P' \in \Lambda(q)} \sum_{e \in P} h^q(e)$. Define $h^q(P)$ as $h^q(P) := \sum_{e \in P} h^q(e)$. We then define Z_k as

$$\min_{P_1 \in \Lambda_k} [\sum_{q:P_1 \notin \Lambda(q)} h^q(P_1) + \min_{P_2 \in \Lambda_k \setminus P_1} \sum_{q:P_1 \in \Lambda(q)} h^q(P_2)]$$

i.e., Z_k equals

$$\min_{P_1,P_2 \in \Lambda_k : P_1 \neq P_2} [\sum_{q:P_1 \notin \Lambda(q)} h^q(P_1) + \sum_{q:P_1 \in \Lambda(q)} h^q(P_2)] \tag{2}$$

Note that we have now ensured that all the dual equations, but that for λ, are satisfied. To satisfy the λ equation, we simply divide all dual variables by $\alpha = \sum_{k \in K} Z_k d_k$ to obtain a feasible solution that satisfies $\sum_k d_k Z_k = 1$.

Given h^q for each q, define $D(h)$ as the value of the corresponding dual objective function. For an integer i, define $D(i)$ to be the dual objective function value at the beginning of the i^{th} iteration; whenever we say "iteration" below, we refer to the outer "while" loop in the algorithm. We show below that a sufficient stopping criterion is when the value of the dual objective $D(h)$ is at least 1.

```
Initialize h^q(e) = δ/u(e) ∀e ∈ E, ∀q ∈ Q
Initialize x ≡ 0⃗, y ≡ 0⃗
while D(h) < 1
    for k = 1 to |K| do
        d' ← d_k
        while D(h) < 1 and d' > 0
            P_1, P_2 ← shortest disjoint path-pair in
                Λ_k that minimize (2)
            u ← min{d', u(P_1), u(P_2)}
            x(P_1) ← x(P_1) + u
            y^{P_1}(P_2) ← y^{P_1}(P_2) + u
            d' ← d' − u
            for q ∈ Q
                if P_1 ∉ Λ(q) do
```

$\forall e \in P_1$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u/u(e)}$
else *do*
$\forall e \in P_2$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u/u(e)}$
end for
end while
end for
end while

4.4 Analysis of the algorithm

Note that the procedure stops at the first iteration t for which $D(t+1) \geq 1$. In the first $t-1$ iterations, for every commodity k we have routed $(t-1)d_k$ units of flow along primary paths and backup paths. This flow likely violates capacity constraints. By scaling appropriately, it can be made feasible.

Lemma 1 *We have $\lambda > (t-1)/\log_{e^\epsilon}(1/\delta)$; so, dividing the primal solution at the end of iteration $t-1$ by $\log_{e^\epsilon} 1/\delta$ gives a feasible solution.*

Proof: Consider link e. Consider the third primal inequality for some fixed (q, e). Whenever we increase $x(P)$ for some (k, P) pair such that $P \in \Lambda_k \backslash \Lambda(q)$ by some value u, the first "for" loop inside the inner while loop multiplies $h^q(e)$ by $e^{\epsilon u/u(e)}$. Whenever we increase $y^{P'}(P)$ for some (k, P, P') such that $P \in \Lambda_k \backslash \Lambda(q)$ and $P' \in \Lambda(q) \cap \Lambda_k$ by some value u, the second "for" loop inside the inner while loop multiplies $h^q(e)$ by $e^{\epsilon u/u(e)}$. Thus, $h^q(e)$ increases by a factor of e^ϵ with every $u(e)$ units using link e in state q. Since $h^q(e) = \delta/u(e)$ initially and is less than $1/u(e)$ at the end of iteration $t-1$, we must have that the flow through e is at most $u(e) \cdot \log_{e^\epsilon} 1/\delta$ at the end of iteration $t-1$. This completes the proof. ■

Theorem 2 *Suppose the optimal primal (and hence dual) solution value is at least 1. (If this is not true, it is easily achievable by scaling the numerical values in the primal and dual problems, as pointed out in [10].) The primal solution at the end of the $(t-1)$st iteration divided by $\log_{e^\epsilon} 1/\delta$ is an ϵ'-approximate solution, for $\epsilon' = c\epsilon$, for an appropriate constant $c > 0$, and appropriate choice of δ.*

Proof: Let h_i be the length functions at the beginning of the i^{th} iteration. For each i, k let $h_{i,k}$ be the length function before routing the k^{th} commodity in the i^{th} iteration. Also let $h_{i,k,s}$ be the length function before routing the s^{th} pair of paths for commodity k in the i^{th} iteration.

Let $Z_k(h)$ be the value of Z_k computed using the given values of h variables before the rescaling necessary to make the dual feasible. Equivalently, $Z_k(h)$ is the value of (2), where P_1, P_2 are the shortest disjoint-path pair which the algorithm would select when presented with the length functions given. Let $\alpha(h)$ equal $\sum_{k \in K} d_k Z_k(h)$. We have defined α so that h divided by $\alpha(h)$ is a feasible dual solution with value $D(h)/\alpha(h)$.

After flow u has been routed on P_1 and P_2 as the s^{th} pair of primary and backup paths for commodity k, we have the following bound on $D(h_{i,k,s+1})$; here, for notational convenience, we use

$$S \doteq \{(e,q):\ e \in P_1, P_1 \notin \Lambda(q)\} \tag{3}$$
$$S' \doteq \{(e,q):\ e \in P_2, P_1 \in \Lambda(q)\}. \tag{4}$$

We obtain

$$\begin{aligned} D(h_{i,k,s+1}) &= \sum_{e\in E} u(e) \sum_{q\in Q} h^q_{i,k,s+1}(e) \\ &= D(h_{i,k,s}) + \\ &\quad \sum_{(e,q)\in S} u(e) h^q_{i,k,s}(e)(e^{\epsilon u/u(e)} - 1) + \\ &\quad \sum_{(e,q)\in S'} u(e) h^q_{i,k,s}(e)(e^{\epsilon u/u(e)} - 1) \\ &\leq D(h_{i,k,s}) + \\ &\quad \sum_{(e,q)\in S} h^q_{i,k,s}(e)(u\epsilon + \epsilon^2 u^2/u(e)) + \\ &\quad \sum_{(e,q)\in S'} h^q_{i,k,s}(e)(u\epsilon + \epsilon^2 u^2/u(e)), \end{aligned} \tag{5}$$

where inequality (5) uses the fact that $e^a \leq 1 + a + a^2$ for $0 \leq a \leq 1$. For notational convenience, let

$$\begin{aligned} \gamma &\doteq \sum_{e\in P_1} \sum_{q:P_1\notin\Lambda(q)} h^q_{i,k,s}(e) + \sum_{e\in P_2} \sum_{q:P_1\in\Lambda(q)} h^q_{i,k,s}(e) \\ &= \sum_{q:P_1\notin\Lambda(q)} h^q_{i,k,s}(P_1) + \sum_{q:P_1\in\Lambda(q)} h^q_{i,k,s}(P_2). \end{aligned}$$

Thus we have from (5) that

$$D(h_{i,k,s+1}) \leq D(h_{i,k,s}) + u\epsilon(1+\epsilon)\cdot\gamma \tag{6}$$
$$\leq D(h_{i,k,s}) + u\epsilon(1+\epsilon)Z(h_{i,k,s}), \tag{7}$$

where inequality (6) uses the fact that u is upper-bounded by $\min\{u(P_1), u(P_2)\}$; and inequality (7) follows from the definition of Z_k in (2).

Note that the length functions are non-decreasing. Then over all the path pairs chosen in the process of pushing d_k flow for commodity k, we can conclude that:

$$D(h_{i,k+1}) \leq D(h_{i,k}) + \epsilon(1+\epsilon)d_k Z_k(h_{i,k+1})$$

Each iteration pushes d_k for every commodity. Extending this inequality over all values of k and again noticing that the length functions (and thus Z_k) are nondecreasing, we can conclude that

$$\begin{aligned} D(i+1) &\leq D(i) + \epsilon(1+\epsilon) \sum_{k \in K} d_k Z_k(h_{i+1}) \\ &\leq D(i) + \epsilon(1+\epsilon)\alpha(h_{i+1}). \end{aligned}$$

Thus we have established a crucial inequality in the analysis presented in [10], Section 5. Let β be the optimal dual (and thus primal) objective function value. Let m denote the number of links in the network G. Since the initial objective function value is at most $m|Q|\delta$, through a very similar analysis to that in [10], we can conclude that if $\beta \geq 1$ and if the algorithm terminates in iteration t, then

$$\beta/(t-1) \leq \epsilon(1+\epsilon) \cdot [(1-\epsilon)\ln((1-\epsilon)/(m|Q|\delta))]^{-1}.$$

Thus, setting

$$\delta = (m|Q|/(1-\epsilon))^{-1/\epsilon} \tag{8}$$

implies (using the same arguments as in [10]) that this gives a provably good approximate solution.

■

Runtime analysis:
Let n denote the number of nodes in G (as mentioned above, m denotes the number of links in G); also recall that $|Q| = m+1$ in our application. Each augmentation requires $O(m|Q||\Lambda|^2(\log n)^{O(1)})$ time to find a shortest path. Similar to the analysis in [10], Section 7, the number of iterations before the stopping criterion is met is $O(\epsilon^{-2}(|\Lambda| + m|Q|)(\log n)^{O(1)})$. However, in our application, typically $d_k \leq u(P)$, so that the shortest path calculation at each step is also a minimum cost flow computation. Since minimum cost flow computations require fewer iterations (see, e.g., [14, 10]), this gives an improved bound on the run time: $O(\epsilon^{-2}m|Q||\Lambda|^3(\log n)^{O(1)})$.

5 The main algorithm

We now present our ϵ-approximation algorithm for (BCR). For ease of notation in the following, if f is a linear function defined on ground set E and $P \subseteq E$, then we use $f(P)$ to denote $\sum_{e \in P} f(e)$.

5.1 The Algorithm

Our main LP (BCR) of Section 2 is a *mixed packing and covering linear program*: it has the form $\max \lambda,\ Px \leq p,\ Cx \geq \lambda c,\ x \geq 0$ for nonnegative matrices P and C and positive vectors p and c.

The dual of (BCR) is

$$\min\ \phi B + \sum_{e \in E} u(e) \sum_{q \in Q} h^q(e) \text{ subject to:}$$

- $c(P', P)\phi + \sum_{q:P' \in \Lambda(q)} \sum_{e \in P} h^q(e) \geq w^{P'}$, $\forall\ k \in K,\ \forall P', P \in \Lambda_k,\ P \neq P'$;
- $c(P)\phi + \sum_{q:P \notin \Lambda(q)} \sum_{e \in P} h^q(e) \geq z_k + \sum_{P' \in \Lambda_k - \{P\}} w^{P'}$, $\forall\ P \in \Lambda_k,\ \forall\ k \in K$;
- $\sum_{k \in K} d_k z_k + \sum_{k \in K} \sum_{P' \in \Lambda_k} d_k w^{P'} \geq 1$;
- $\phi, h, w, z \geq 0$.

We start with $x, y \equiv 0$, $h^q(e) = \delta / u(e)$, and $\phi = \delta / B$. Given a singleton-vector pair (ϕ, h), let $\mu_P(\phi, h)$ and $\nu_{P',P}(\phi, h)$ be defined as follows:

$$\mu_P(\phi, h) := c(P)\phi + \sum_{q:P \notin \Lambda(q)} h^q(P),$$

$$\nu_{P',P}(\phi, h) := c(P', P)\phi + \sum_{q:P' \in \Lambda(q)} h^q(P).$$

When ϕ and h are clear from context, we write simply μ_P and $\nu_{P',P}$. We obtain z_k and $w^{P'}$ for $P' \in \Lambda_k$ by analytically finding an explicit optimal solution to the following new linear program, where the constraint (10) is supposed to hold for all $P, P' \in \Lambda_k,\ P \neq P'$:

$$\max\ d_k\ [\ z_k + \sum_{P' \in \Lambda_k} w^{P'}\ ] \tag{9}$$

$$w^{P'} \leq \nu_{P',P} \tag{10}$$

$$z_k + \sum_{P' \in \Lambda_k - \{P\}} w^{P'} \leq \mu_P \quad \forall\ P \in \Lambda_k \tag{11}$$

$$z, w \geq 0 \tag{12}$$

Let Z_k be the optimal value of this problem; that is, $Z_k = z_k + \sum_{P \in \Lambda_k} w^P$. By dividing our solution (ϕ, h, w, z) by $\sum_k d_k Z_k$, we obtain a feasible solution to the above-seen LP that is the dual of (BCR). Thus, for now, we concern ourselves with only the dual variables h and ϕ.

As with the previous algorithm, our algorithm for the budgeted concurrent restoration problem cycles through the commodities. For commodity k, using our current vectors $h_{l,k-1}$ and $\phi_{l,d-1}$, we find an optimal solution to the following problem, and use this solution to determine the update step in the current iteration (the second of the following three constraints is supposed to hold for all $P' \in \Lambda_k$):

$$\min \sum_{P \in \Lambda_k} \mu_P \cdot x(P) + \sum_{(P,P') \in \Lambda_k^2} \nu_{P',P} \cdot y^{P'}(P)$$

subject to:

$$\begin{aligned} \sum_{P\in\Lambda_k} x(P) & & & \geq & d_k \\ \sum_{P\in\Lambda_k;\ P\neq P'} x(P) & + & \sum_{P\in\Lambda_k,\ P\neq P'} y^{P'}(P) & \geq & d_k \\ & & x, y & \geq & 0 \end{aligned} \tag{13}$$

Note that the linear program described by (9)-(12) is the dual of (13). The optimal solutions to both problems take one of two forms. We describe the structure of the optimal solution to both (13) and (9)-(12) in Lemma 3.

To begin with, we introduce two functions that are useful in demonstrating this. The first is an extension of the Z_k introduced in the previous section:

$$\overline{Z}_k = \min_{P_1,P_2\in\Lambda_k:\ P_1\neq P_2} \mu_{P_1} + \nu_{P_1,P_2} \tag{14}$$

To obtain the second, assume the $P_i \in \Lambda_k$ are indexed by increasing μ_{P_i} value, so that $\mu_{P_1} \leq \mu_{P_2} \leq \cdots$.

$$\hat{Z}_k = \min_{2\leq r\leq|\Lambda_k|} (\sum_{i=1}^{r} \mu_{P_i})/(r-1). \tag{15}$$

Let r_k be the value of r that determines $\hat{Z}_k$ in this expression.

Lemma 3 *The optimal solution to (13) has value equal to* $d_k\ \min\{\overline{Z}_k, \hat{Z}_k\}$. *If the optimal solution has value* $d_k\overline{Z}_k$ *determined by* P_1 *and* P_2, *then it satisfies* $x(P_1) = y^{P_1}(P_2) = d_k$, *all other variables zero. Otherwise, it has value* $d_k\tilde{Z}_k$ *and there are* r *distinct paths in* Λ_k *such that* $x(P_i) = d_k/(r-1)$ *for* $i = 1, 2, \ldots, r$, *all other variables are zero.*

Proof: $\hat{Z}_k \leq \overline{Z}_k$: In this case, we claim that $x(P_i) = d_k/(r_k - 1)$ for $i \leq r_k$ for r_k the value that determines $\hat{Z}_k$, and $x(P_i) = 0$ for $i > r_k$ is an optimal solution: It is clearly feasible and has value $d_k\hat{Z}_k$. Now consider the dual solution $w(P_i) = \hat{Z}_k - \mu_i$ if $i \leq r_k$ and $z_k = 0$. This clearly satisfies (11). By Lemma 4, $w(P_i) \geq 0$ for all i, satisfying (12). Since $\hat{Z}_k \leq \overline{Z}_k \leq \mu_{P_i} + \nu_{P_i,P_j}$, we have that (10) are satisfied for all P, P'. Thus the solution is feasible. Since its value is $d_k\hat{Z}_k$, matching the value of the above solution to (13), both primal and dual solutions are optimal.

$\overline{Z}_k \leq \hat{Z}_k$: If $\overline{Z}_k = \mu_{P_1} + \nu_{P_1,P_2}$, we set $x(P_1) = y^{P_1}(P_2) = d_k$ and all other primal variables equal to 0. This solution is clearly feasible and has value $d_k\overline{Z}_k$. Consider the following dual solution.

$$w(P_i) = \max\{Z_k - \mu_{P_i}, 0\} \quad \forall\ i \leq |\Lambda_k| \tag{16}$$

and $z_k = \overline{Z}_k - \sum_i w(P_i)$. Since $\overline{Z}_k \leq \hat{Z}_k$, Lemma 4 implies that $Z_k \leq \mu_P$ for all $w(P) = 0$, and hence this solution satisfies (11). By definition, w is nonnegative. Summing (16) over all i with $w(P_i) > 0$ yields $\sum w(P_i) = r\overline{Z}_k - \sum \mu_{P_i}$ which

implies that $z_k = \overline{Z}_k - \sum w(P_i) = \sum \mu_{P_i} - (r-1)\overline{Z}_k \geq \sum \mu_{P_i} - (r-1)\hat{Z}_k \geq 0$, where the second inequality follows from the definition of $\hat{Z}_k$. Finally, $\overline{Z}_k \leq \mu_{P_i} + \nu_{P_i,P_j}$ for all i and j. Hence (10) are satisfied, and the solution is feasible. It has value $d_k\overline{Z}_k$, matching the given solution to its dual (13), hence both are optimal. ∎

Lemma 4 *$\hat{Z}_k \geq \mu_{P_i}$ for $i \leq r_k$ and $\hat{Z}_k \leq \mu_{P_i}$ for $i > r_k$.*

Proof: Let $\hat{Z}_k(r) := (\sum_{i=1}^{r} \mu_{P_i})/(r-1)$. For ease of notation, we write μ_i for μ_{P_i} for the remainder of the proof. The lemma follows by noting that $\hat{Z}_k(r+1)$ is a convex combination of $\hat{Z}_k(r)$ and μ_{r+1}. Thus μ_i for $i > 2$ is included in definition of $\hat{Z}_k$ if and only if $\mu_i \leq \hat{Z}_k(i-1)$. Since $\mu_i \leq \mu_j$ for $i < j$, if $\mu_i \leq \hat{Z}_k(i-1)$, then $\mu_i \leq \hat{Z}_k(j)$ for all $j > i$. ∎

Lemma 3 implies that the optimal solution to (13) may be described as a set of paths, each carrying an equal amount of flow. Without loss of generality, assume this set is $\{P_1, \ldots, P_{r_k}\}$, where $r_k \geq 2$. (If $\overline{Z}_k < \hat{Z}_k$, this set is $\{P_1, P_2\}$ and $r_k = 2$.) The amount of flow sent along each path is u where $u := \min\{d_k/(r_k-1), \min_{i \leq r_k} u(P_i), B/(\sum_{i=1}^{r_k} c(P_i))\}$. This quantity u is set so that the left side of an inequality in our main LP (BCR) does not increase by more than the fixed value of the right side in any one step. In practice, $\sum_{P \in \Lambda_k} c(P)$ is significantly smaller than B, so that, when combined with scaling of capacities, u is determined by d_k. The updates to the dual variables for each case are described in the algorithm in Figure 1. The stopping criterion is the same as in [10, 9].

5.2 Analysis

The extended arguments for $\lambda > (t-1)/\log_{e^\epsilon}(1/\delta)$ are similar to the previous arguments. We now show that the primal solution divided by this choice of λ after t iterations is indeed an ϵ-approximate solution.

Theorem 5 *Suppose the optimal primal (and hence dual) solution value is at least 1. The primal solution at the end of the $(t-1)$st iteration divided by $\log_{e^\epsilon} 1/\delta$ is an ϵ'-approximate solution, for $\epsilon' = c\epsilon$, for an appropriate constant $c > 0$, and appropriate choice of δ.*

Proof: Let h_i and ϕ_i be the dual variables at the beginning of the i^{th} iteration. Let $h_{i,k}$ and $\phi_{i,k}$ be the dual variables before routing the k^{th} commodity in the i^{th} iteration. Let $h_{i,k,s}$ and $\phi_{i,k,s}$ be the dual variables before routing the s^{th} pair of paths for commodity k in the i^{th} iteration. Let $D(i,k,s)$ be the value of the dual objective function using dual variables $h_{i,k,s}$ and $\phi_{i,k,s}$.

Let $Z_k(h,\phi)$ be the value of Z_k computed using the given values of h and ϕ before the rescaling necessary to make the dual feasible. Equivalently, $Z_k(h,\phi) = \min\{\overline{Z}_k(h,\phi), \hat{Z}_k(h,\phi)\}$. Let $\alpha(h,\phi)$ equal $\sum_{k \in K} d_k Z_k(h,\phi)$. We have defined α so that (h,ϕ) divided by $\alpha(h,\phi)$ is a feasible dual solution with value

Budget-LP ($G = (\Lambda, E, c, u, d)$)

Initialize $h^q(e) = \delta/u(e)\ \forall e \in E,\ \forall q \in Q$
Initialize $\phi = \delta/B$
Initialize $x \equiv \vec{0}$, $y \equiv \vec{0}$
while $D(h) < 1$
  for $k = 1$ to $|K|$ *do*
    $d' \leftarrow d_k$
    while $D(h) < 1$ and $d' > 0$
      $Z_k = \min\{\overline{Z}_k, \hat{Z}_k\}$.
      Let $\{P_1, P_2, \ldots, P_{r_k}\}$ be paths achieving
        the optimum Z_k.
      $C \leftarrow B/(\sum_{i=1}^{r_k} c(P_i))$
      $u_0 \leftarrow$ minimum of: (i) $d'/(r_k - 1)$,
        (ii) $\min_{1 \le i \le r_k, e \in P_i} u(e)$, and (iii) C.
      $\phi \leftarrow \phi e^{\epsilon u_0/C}$
      $d' \leftarrow d' - (r_k - 1)u_0$
      if $Z_k = \overline{Z}_k$,
        $x(P_1) \leftarrow x(P_1) + u_0$
        $y^{P_1}(P_2) \leftarrow y^{P_1}(P_2) + u_0$
        for $q \in Q$
          if $P_1 \notin \Lambda(q)$ *do*
            $\forall e \in P_1$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u_0/u(e)}$.
          else *do*
            $\forall e \in P_2$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u_0/u(e)}$.
        end for
      else ($Z_k = \hat{Z}_k$),
        for $i \le r_k$,
          $x(P_i) \leftarrow x(P_i) + u_0$
          for $e \in P_i$, q such that $P_i \notin \Lambda(q)$,
            $h^q(e) \leftarrow h^q(e)e^{\epsilon u_0/u(e)}$.
          end for
        end for
    end while
  end for
end while

Figure 1: Algorithm for (BCR)

$D(h,\phi)/\alpha(h,\phi)$. Let $D(i,k,s)$ be the value of the dual solution with variables $h_{i,k,s}$ and $\phi_{i,k,s}$. Below, we establish that

$$D(i,k,s+1) \leq D(i,k,s) + u\epsilon(1+\epsilon)Z(h_{i,k,s},\phi_{i,k,s}) \tag{17}$$

holds no matter how Z_k is determined. We do this by analyzing each case separately. Once we establish (17), the remaining argument mirrors the argument in the proof of Theorem 2, so we omit it here.

Suppose $Z_k(h_{i,k,s},\phi_{i,k,s}) = \overline{Z}_k(h_{i,k,s},\phi_{i,k,s})$. Let S and S' be as in (3), (4). After u is routed on P_1 and P_2 as the the s^{th} pair of primary and backup paths for commodity k, we have that

$$\begin{aligned}
D(i,k,s+1) &= B\phi_{i,k,s+1} + \sum_{e\in E} u(e) \sum_{q\in Q} h^q_{i,k,s+1}(e) \\
&= D(i,k,s) + \\
&\quad B\phi_{i,k,s}(e^{\epsilon\cdot u\cdot(c(P_1)+c(P_2))/B} - 1) + \\
&\quad \sum_{(e,q)\in S} u(e) h^q_{i,k,s}(e)(e^{\epsilon u/u(e)} - 1) + \\
&\quad \sum_{(e,q)\in S'} u(e) h^q_{i,k,s}(e)(e^{\epsilon u/u(e)} - 1).
\end{aligned}$$

Thus, letting $\gamma \doteq c(P_1) + c(P_2)$, we get

$$\begin{aligned}
D(i,k,s+1) &\leq D(i,k,s) + \\
&\quad \epsilon u\gamma\phi_{i,k,s} + \epsilon^2 u^2\gamma^2/B) + \\
&\quad \sum_{(e,q)\in S} h^q_{i,k,s}(e)(u\epsilon + \epsilon^2 u^2/u(e)) + \\
&\quad \sum_{(e,q)\in S'} h^q_{i,k,s}(e)(u\epsilon + \epsilon^2 u^2/u(e)) \qquad (18) \\
&\leq D(i,k,s) + u\epsilon(1+\epsilon)\ \phi_{i,k,s}\ \gamma + \\
&\quad u\epsilon(1+\epsilon) \sum_{(e,q)\in(S\cup S')} h^q_{i,k,s}(e) \qquad (19) \\
&= D(i,k,s) + \\
&\quad u\epsilon(1+\epsilon)[\ \mu_{P_1} + \nu_{P_1,P_2}\ ] \\
&= D(i,k,s) + \\
&\quad u\epsilon(1+\epsilon) Z(h_{i,k,s},\phi_{i,k,s}), \qquad (20)
\end{aligned}$$

where inequality (18) uses the fact that $e^a \leq 1+a+a^2$ for $0 \leq a \leq 1$; inequality (19) uses the fact that $u \leq \min\{u(P_1), u(P_2), B/(c(P_1)+c(P_2))\}$; and inequality (20) follows from the determination of Z_k as in (14).

Next suppose $Z_k(h_{i,k,s},\phi_{i,k,s}) = \hat{Z}_k(h_{i,k,s},\phi_{i,k,s})$. For ease of notation, we define:

$$S'' \doteq \bigcup_{j=1}^{r} \{(e,q) :\ e \in P_j, P_j \notin \Lambda(q)\}; \tag{21}$$

$$\begin{aligned}\psi &\doteq \epsilon u \cdot (\sum_{j=1}^{r} c(P_j))/(r-1), \text{ and}\\ \eta &\doteq u\epsilon/(r-1).\end{aligned}$$

(Note that the union in (21) is a union of r pairwise disjoint sets, since the paths P_j are pairwise link-disjoint.) After $u/(r-1)$ is routed in the i^{th} iteration in the s^{th} round for commodity k on the cheapest r_k paths determined by $\hat{Z}_k$, we have that

$$\begin{aligned}D(i,k,s+1) &= B\phi_{i,k,s+1} + \sum_{e\in E} u(e) \sum_{q\in Q} h^q_{i,k,s+1}(e)\\ &= D(i,k,s) + B\phi_{i,k,s}(e^{\epsilon u/C} - 1) +\\ &\quad \sum_{(e,q)\in S''} u(e) h^q_{i,k,s}(e)(e^{\eta/u(e)} - 1)\\ &\leq D(i,k,s) + \phi_{i,k,s}(\psi + \psi^2/B)) +\\ &\quad \sum_{(e,q)\in S''} h^q_{i,k,s}(e)(\eta + \eta^2/u(e)) \qquad (22)\\ &\leq D(i,k,s) +\\ &\quad \eta(1+\epsilon)\sum_{j=1}^{r} \mu_{P_j}(\phi_{i,k,s}, h_{i,k,s}) \qquad (23)\\ &= D(i,k,s) +\\ &\quad u\epsilon(1+\epsilon) Z_k(h_{i,k,s}, \phi_{i,k,s}), \qquad (24)\end{aligned}$$

where inequality (22) uses the fact that $e^a \leq 1 + a + a^2$ for $0 \leq a \leq 1$; inequality (23) uses the fact that $u \leq \min\{(r-1)u(P_j), C\}$; and inequality (24) follows from the determination of $\hat{Z}_k$ as in (15), the fact that $Z_k = \hat{Z}_k$, and the choice of r and P_j. ∎

6 Extension to the commodity-dedicated capacity reservation

In this section, we consider a simplified relative of our main problem, where we insist on dedicated reserve capacity for each commodity. In our main problem, capacity used by a backup path for a failure affecting commodity k could be used by a different commodity when a different failure occurs. This more sophisticated model allows for better utilization of network capacity. In current network models, however, this is not done. Instead each commodity has dedicated reserve capacity. That is, if any path fails, then there is enough spare capacity reserved for that commodity to route all the demand for that commodity. Since there is no transfer of flow from one path to another in case of failure, we consider another commonly used objective function: letting $f(e)$ be the total flow on link e, we wish to minimize the total cost of the flow, i.e.,

$\sum_{e \in E} c(e) f(e)$. This problem and its variant where the paths are chosen on the fly, has been studied by Bienstock and Muratore [1] in the context of capacity expansion with integrality constraints and in the single source-destination pair by Brightwell et al. [4].

We describe here an ϵ-approximation scheme to solve the corresponding linear program for this problem by observing that this problem corresponds to the special case of the previous problem with all y variables forced to zero. In this case, Z_k is always determined by $\hat{Z}_k$. Likewise, the $h^q(e)$ variables are replaced by a single variable $h(e)$, and $z_k = 0$ so may also be omitted. This modification in the linear program formulation is highlighted below. The algorithm and analysis then follow from the previous section, and so are omitted here. The dedicated capacity concurrent restoration problem is as follows:

$$\begin{aligned} \min \sum_{e \in E} c(e) f(e) & \\ \sum_{P \in \Lambda_k \setminus \{P'\}} x(P) & \geq d_k \quad \forall\, k \in K, P' \in \Lambda_k \\ \sum_{P : e \in P} x(P) & \leq f(e) \quad \forall\, e \in E \\ f(e) & \leq u(e) \quad \forall\, e \in E \\ x(P) & \geq 0 \end{aligned} \tag{25}$$

It is possible to remove the variables $f(e)$ from this formulation to obtain an equivalent formulation (here, $c(P)$ simply stands for $\sum_{e \in P} c(e)$, and is *not* what is defined in Section 2 in the context of our original problem):

$$\begin{aligned} \min \sum_{P \in \Lambda} c(P) x(P) & \\ \sum_{P \in \Lambda_k \setminus \{P'\}} x(P) & \geq d_k \quad \forall\, k \in K, P' \in \Lambda_k \\ \sum_{P : e \in P} x(P) & \leq u(e) \quad \forall\, e \in E \\ x(P) & \geq 0. \end{aligned} \tag{26}$$

We solve this by solving the concurrent flow version with a budget constraint and then searching for the optimal budget. The corresponding primal LP is:

$$\begin{aligned} \max \lambda d_k & \\ \sum_{P \in \Lambda_k \setminus \{P'\}} x(P) & \geq \lambda \quad \forall\, k \in K, P' \in \Lambda_k \\ \sum_{P : e \in P} x(P) & \leq u(e) \quad \forall\, e \in E \\ \sum_{P \in \Lambda} c(P) x(P) & \leq B \\ x(P) & \geq 0 \end{aligned}$$

The dual LP is as follows, where the first constraint is supposed to hold for all $k \in K$ and for all $P \in \Lambda_k$:

$$\begin{aligned}
\min B\phi \; + \sum_{e\in E} u(e)h(e) & \\
c(P)\phi \; + \sum_{e\in P} h(e) \quad &\geq \sum_{P'\in\Lambda_k\setminus\{P\}} w^{P'} \\
\sum_{k\in K} d_k \sum_{P\in\Lambda_k} w^P \quad &\geq 1 \\
l, w \quad &\geq 0
\end{aligned}$$

Substituting $Z_k = \sum_{P\in\Lambda_k} w^P$, the dual can be rewritten as follows, once again with the first constraint required to hold for all $k \in K$ and for all $P \in \Lambda_k$:

$$\begin{aligned}
\min B\phi \; + \sum_{e\in E} u(e)h(e) & \\
w^P + c(P)\phi \; + h(P) \quad &\geq Z_k \\
\sum_{k\in K} d_k Z_k \quad &\geq 1 \\
l, w \quad &\geq 0.
\end{aligned}$$

As mentioned above, the algorithm and analysis now follow from the previous section.

7 Computational Results

We now describe our preliminary computational results which compare an initial implementation of our ϵ-approximation algorithm with solutions obtained from an exact LP commercial solver CPLEX [5]. We point out that our current implementation is not optimized for either performance or memory. We present our results in three tables. The first lists the sizes of the instances we present, and the second lists the memory usage and running times of our ϵ-approximation and of the general LP solver on these instances. The third table presents objective function values. Importantly, it also demonstrates that even "large" values of ϵ such as 0.4 give near-optimal solutions! The advantage is that since ϵ is "large" here, the running time is better than when ϵ is smaller (recall that our upper bound on the running time is proportional to $1/\epsilon^2$). Some reasons for this "best of both worlds" (near-optimal solution and improved running time) phenomenon are presented after discussing the first two tables.

We considered the following real-world instances in our experiments.

Instance	Nodes	Links	Demands	Paths
1	30	38	435	900
2	29	55	406	1221
3	39	49	630	1288
4	226	302	241	578
5	121	208	3983	11581
6	224	304	475	997
7	224	304	7288	15194
8	224	304	9844	20531
9	224	304	12355	25750

The runs were conducted on a 8-processor SGI Origin 2000 machine with 8 Gbps of resident memory. We remark that running times reported are only (rough) user-time measures obtained from the UNIX "time" function. The resident memory measurements are obtained through the "top" routine. The following table shows that general solver is typically faster for the smaller instances. (We remark that over half the time for the LP solver is spent creating and reading in the LP file itself.) For the larger instances, however, the general solver requires much more resident memory, even after its row-elimination heuristics are applied. For the largest instances the process is actually killed due to memory needs beyond the available memory. The approximation algorithm typically requires about 90 percent less memory, and instances 6-9 – which have the same underlying network – imply that the algorithm scales (in run time and memory) sublinearly with the number of demands.

	Run Time		Memory	
Instance	ϵ-apx: $\epsilon = 0.1$	CPLEX	ϵ-apx: $\epsilon = 0.1$	CPLEX
1	13.8 s	1.5 s	0.11MB	0.5MB
2	24.5 s	3.2 s	0.13MB	1.2 MB
3	44.4	4.6	0.14MB	1.45
4	96.7	31.1	0.28	2.85
5	2367	516	1.63	2.55
6	428	82.3	0.36	29
7	12119	1728	3.0	62
8	13590	Killed	4.0	Killed
9	16196	Killed	5.0	Killed

We now present our final table which shows that our approximation algorithm performs better than its prescribed error bound. For instance, we get solutions very close to optimal even when ϵ is as large as 0.4. Moreover, there is little difference in objective between setting $\epsilon = 0.1$ and $\epsilon = 0.4$. We studied the details of the intermediate results produced by our algorithms, and the primary reason for this desirable phenomenon (of even large values of ϵ producing near-optimal values) seems to be the following. Recall the interpolation search mechanism that we discuss early in Section 4.1. In practice, this seems to help

select a near-optimal value for the optimal budget B very quickly, whether ϵ is "small" or "large"; a lot of the effort from then on is in proving that this B is near-optimal–i.e., that if B is somewhat smaller, then λ must be smaller than 1. This quick convergence enables us to get near-optimal solutions even if ϵ is comparatively large. A significant advantage of using large values of ϵ is that we obtain reduced running times: this is especially valuable for very large problems. We indicate these running time reductions in the far-right column of the next table, as percentage improvements for $\epsilon = 0.4$ as compared with the case of $\epsilon = 0.1$. The solution values we get for $\epsilon = 0.4$ are the same as for $\epsilon = 0.1$ (except for instance 5, where we get a solution value of $31.562\cdots$ with $\epsilon = 0.4$, which is smaller than the $31.563\cdots$ that we get with $\epsilon = 0.1$!), and so we do not display the solution values for $\epsilon = 0.4$; also, resident memory usage is the same for $\epsilon = 0.1$ and $\epsilon = 0.4$.

Instance	CPLEX	$\epsilon = 0.1$	$\epsilon = 0.4$: Runtime Improvement
1	16.999	17.0	13%
2	1.579	1.586	15%
3	1.713	1.713	32%
4	8.204	8.279	24%
5	29.5	31.563	24%
6	161.5	161.98	23%
7	2,525	2,531	13%
8	Killed	3,427	16%
9	Killed	4,310	16%

8 Conclusions

We have presented a scalable algorithm for routing with shared protection in (optical core) networks. The main analytical novelties are the modeling of the fact that different links can have widely differing probabilities of failure, and the development of an ϵ-approximation algorithm for this model. We have also shown how this scheme specializes for the dedicated capacity routing problem. We have implemented this algorithm, and given empirical results on a family of real-world instances. These show that there are instances where general-purpose LP solvers fail but our scheme works. Future work is required to develop a version of the code which is optimized for performance in order to make a more thorough comparison of running times.

Another interesting avenue for study is the modification of the algorithm to work in a distributed environment where the nodes of the network use some minimal link-state information. Two other technical questions have remained unresolved. The first is to produce a provable approximation scheme for the network design problem for restorable flows. Another is to extend the problem MELTCoRe as follows. In general, for each commodity k, we may only require a fraction of its flow to be restored, depending say on a priority basis. Thus it

would be of interest to replace the right hand sides of the second collection of constraints in the definition of (MELTCoRe) by a value $\gamma_k d_k$, with $\gamma_k \in [0, 1]$.

References

[1] D. Bienstock, G. Muratore, *Strong inequalities for capacitated survivable network design problems*, Math. Programming 89: 127-147, 2001.

[2] D. Bienstock, *Approximately solving large-scale linear programs. I. Strengthening lower bounds and accelerating convergence*, CORC Report 1999-1, Columbia University.

[3] D. Bienstock, *Approximation Algorithms for Linear Programming: Theory and Practice*, CORE Lecture Series monograph, To appear, 2001.

[4] G. Brightwell, G. Oriolo, B. Shepherd, *Some strategies for reserving resilient capacity in a network*, SIAM Journal of Discrete Mathematics, Vol. 14, No. 4, 524–539, 2001.

[5] CPLEX 6.6.1, ILOG, Inc., http://www.ilog.com/products/cplex/.

[6] R. Davis, K. Kumaran, G. Liu, and I. Saniee, *SPIDER: A Simple and Flexible Tool for Design and Provisioning of Protected Lightpaths in Optical Networks*, Bell Labs Technical Journal, Vol. 6, No. 1, January 2001.

[7] L. Fleischer, A. Meyerson, I. Saniee, F. B. Shepherd and A. Srinivasan. *Near-optimal design of MPλS tunnels with shared recovery.* DIMACS Mini-Workshop on Quality of Service Issues in the Internet, 2001.

[8] L. Fleischer, *Approximating Fractional Multicommodity Flow Independent of the Number of Commodities*, SIAM Journal on Discrete Mathematics, 13:505–520, 2000.

[9] L. K. Fleischer and K. D. Wayne, *Fast and Simple Approximation Schemes for Generalized Flow*, Math. Programming, To appear.

[10] N. Garg and J. Könemann, *Faster and simpler algorithms for multicommodity flow and other fractional packing problems*, IEEE Symposium on Foundations of Computer Science, pages 300–309, 1998.

[11] R. Giles, K. Kumaran, D. Mitra, C. Nuzman and S. Saniee, "Selective transparency in optical networks", *Proc. of ECOC2002*, Copenhagen, Oct 2002.

[12] A. V. Goldberg, J. D. Oldham, S. Plotkin, C. Stein, *An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow*, in Integer Programming and Combinatorial Optimization, (Bixby, Boyd, Rios-Mercado, eds.) 1412: 338-352, 1998.

[13] M. D. Grigoriadis and L. G. Khachiyan, *Fast approximation schemes for convex programs with many blocks and coupling constraints*, SIAM J. on Optimization, 4: 86-107, 1994.

[14] M. D. Grigoriadis and L. G. Khachiyan, *Approximate minimum-cost multicommodity flows*, Mathematical Programming, 75:477–482, 1996.

[15] IEEE Communications Magazine, December 1999.

[16] S. Plotkin, D. B. Shmoys, E. Tardos, *Fast approximation algorithms for fractional packing and covering problems*, Math. of Oper. Res. 20: 257-301, 1995. Extended abstract: *Proc. 32nd Annual IEEE Symp. On Foundations of Computer Science*, 495-504, 1991.

[17] E. Rosen et al., *Multiprotocol Label Switching Architecture*, IETF RFC3031, January 2001.

[18] I. Saniee, *Optimal routing in self-healing communications networks*, Int. Trans. in Operations Research, 3:187–195, 1996.

[19] F. Shahrokhi, D. W. Matula, *The maximum concurrent flow problem*, Journal of the ACM, 37:318-334, 1991.