A Scalable Algorithm for the Minimum Expected Cost Restorable Flow Problem

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

September 30, 2004

Abstract

We introduce an optimization model for multicommodity flow together with back-up flows for each commodity under a given set of failure states in a network. We call such a total routing strategy a *restorable multicommodity flow.* The main advantage of restorable flows over other network protection methods, such as 1+1 protection, is that the commodities may share capacity in the network for their back-up flows, thus reducing the overall network capacity required. 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 modeled 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. We develop a scalable approximation scheme for this problem, to solve realistic-sized restorable multicommodity flow problems. We focus on the special case where traffic flows for each commodity are restricted to a (pre-computed) collection of disjoint paths. In this setting, we devise a combinatorial algorithm for the minimum expected cost restorable flow problem. Since the problem is not of a simple packing or covering type, our approximation scheme must repeatedly solving an auxiliary problem that is slightly more complicated than previous algorithms in the literature, that solve shortest column or shortest path subproblems in each phase. We describe the rather technical proof of its convergence, and then give computational results showing that our approach scales for large networks in comparison with general-purpose LP solvers. An earlier version of this paper, with identical theoretical results, appeared in [9].

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

[*]L. Fleischer is with IBM Research, Yorktown Heights, NY, *lkf@watson.ibm.com.* A. Meyerson is with the Dept. of Computer Science, UCLA, *awm@cs.ucla.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 supported in part by the National Science Foundation under Grant No. 0208005. This work was done largely while all authors were at Bell Labs, Lucent Technologies.

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 back-up 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 [14] 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 back-up 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 the network *design problem*: compute a minimum cost subnetwork (with or without integrality constraints) which admits a restorable flow. The third is a *minimum cost restorable flow* problem: to give this last problem meaning, we must specify a measure of cost for a restorable flow.

The cost function we adopt for the minimum cost 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. We assume that we are also equipped with probability distributions for the failure of each routing path. This could be obtained for instance, from historical records on the failure of each link. Our objective is to find a restorable flow which minimizes the expected long-term operational cost of the network.

The resulting stochastic optimization problem can be formulated as a linear program (LP) which grows dramatically with the increase in network size and number of failure scenarios being considered. As with many optimization problems for protected routing, this LP is often too large to solve in practice for realistic-sized networks, cf. [25]. Furthermore, there is a need to carry out such optimizations repeatedly in operational networks. We thus need to devise fast, efficient and scalable approximations for the restorable flow problem with a *guaranteed* performance.

In order to capitalize on the potential for sharing back-up capacity, networks must have the flexibility to implement the restoration strategies imposed by these designs. Many existing technologies do not have the switching capability to do so. In our view, future optical and data switches will still have limited capabilities in this regard. This is one reason why we focus on the restriction where the flow for each commodity must be routed along a given pre-computed collection of (disjoint) paths for that commodity. Flows with this additional structure are operationally simpler for restoration after a failure, and recovery can be achieved by the endpoints for the commodity. The initial collections of disjoint paths may themselves be part of a preprocessing phase which, say, tries to solve the primary problem (in the network without failues) so as to minimize the maximum load on an edge, the overall load or some other operationally-cost-saving criterion.

We adapt the methodology of ϵ-approximation algorithms for multicommodity flows and more generally guaranteed approximations for the mixed packing and covering linear programs. (Details and references are presented in Section 3.) These methods yield algorithms for the feasibility and minimum cost restorable flow problems (the first and third among the three problems introduced above); however, they do not yield an approximation scheme for the network design version.

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 [13], 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 our basic computational findings, showing in particular that in contrast to general LP solvers, we can find solutions for the large and very large instances of the minimum cost restorable flow problem.

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 each path 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 back-up 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 $f(P)$ 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 $g^{P'}(P)$ denotes the amount of flow that is rerouted from path P' to path P when a link on path P' fails. The pair (f, g) 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. Basically, we are assuming that the probability of failure of a path due to 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 measuring 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 back-up 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:

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

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

$$\min \; EC(f,g)$$

subject to:

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

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).

2.1 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 $EC(f,g)$ (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* (BCR).

$$\begin{aligned}
&\textbf{maximize } \lambda \\
&\textbf{subject to} \\
&\sum_{P\in\Lambda_k} f(P) \geq \lambda d_k,\ \forall\, k\in K \\
&\sum_{P\in\Lambda_k\setminus P'} [f(P)+g^{P'}(P)] \geq \lambda d_k,\ \forall\, P'\in\Lambda_k,\ \forall\, k\in K \\
&\sum_{k\in K}\sum_{\substack{P:e\in P\\ P\in\Lambda_k\setminus\Lambda(q)}} [f(P)+\sum_{P'\in\Lambda_k\cap\Lambda(q)} g^{P'}(P)] \leq u(e),\ \forall\, e\in E,\ \forall\, q\in Q \qquad (2)\\
&\sum_{k\in K}\sum_{P\in\Lambda_k} [c(P)f(P)+\sum_{P'\in\Lambda_k\setminus P} c(P',P)g^{P'}(P)] \leq B \qquad (3)\\
&f,g,\lambda \geq \mathbf{0} \qquad (4)
\end{aligned}$$

2.2 Notation

For ease of notation in the remainder of the paper, if s is a linear function defined on ground set E and $P \subseteq E$, then we use $s(P)$ to denote $\sum_{e\in P} s(e)$. We use $\underline{s}(P)$ to denote $\min_{e\in P} s(e)$. We use $\mathbf{0}$ to denote the vector of all 0's.

3 Solving Difficult Linear Programs

The linear program (BCR) 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.

(BCR) and the LP's we solve in this paper are *k-block packing LP's*. k-block packing LP's are of the following form:

$$\begin{array}{rcl} \max & \lambda & \\ Ax & \leq & b \\ Cx & \geq & \lambda d, \\ x & \geq & \mathbf{0}, \end{array} \tag{5}$$

where A and C are nonnegative, real matrices, and b, d are strictly positive, real vectors. A is $m \times n$; C is a block diagonal matrix of size $(\sum_k m_k) \times n$, with each block C_k of size $m_k \times n_k$; b is $m \times 1$; and d is $(\sum_k m_k) \times 1$. The k-block packing problem is a mixed packing and covering problem. Let λ^* be the optimal solution to (5). We seek solutions (x, λ) such that (x, λ) is feasible for (5) and $\lambda \geq (1-\epsilon)\lambda^*$. Such a solution is called *ϵ-optimal.*

Approximation schemes based on Frank-Wolfe [11] methods have been developed for problems like (5) in a sequence of papers starting with a focus on multicommodity flow problems in [21, 22, 26], and generalized to problems of the above form in [23, 16, 17]. Since then work has focused on reducing the complexity of these algorithms or generalizing the range of applicability [7, 8, 12, 13, 18, 19, 20, 24, 27, 28]. Empirically, these methods have been shown to perform well in practice [1, 2, 4, 15]. For a more complete discussion of these techniques and some of their origins, the reader is referred to [4].

In this section, we describe a generalization of the k-commodity packing algorithm described in [10] to solve (5). The k-commodity packing algorithm is itself an extension of Garg and Könemann's algorithm for concurrent multicommodity flow. [1]

We begin by expressing (5) so that its block structure is apparent. Vector d decomposes into vectors d_k, each of dimension m_k. Variable vector x decomposes into K vectors, where x^k is of dimension n_k. Matrix A decomposes into

[1]Actually, by substitution of variables, it is possible to transform (5) directly into a k-commodity packing problem. Thus, we could simply employ the algorithm in [10] to the transformed problem. We have chosen to present the algorithm in general form here instead because the ideas involved in implementing the transformation are the same as using the general form, and we think that explaining the implementation is clearer if we can refer to the algorithm in its general form.

K matrices, the k^{th} of dimension $m \times n_k$. Thus we have:

$$\begin{aligned} \max \quad & \lambda && (6)\\ \sum_k A_k x^k \quad & \leq \quad b && (7)\\ C_k x^k \quad & \geq \quad \lambda d_k, \quad \text{for } 1 \leq k \leq |K| && (8)\\ x \quad & \geq \quad \mathbf{0}, \quad \text{for } 1 \leq k \leq |K| && (9)\end{aligned}$$

This algorithm requires an oracle to solve $|K|$ block problems, $1 \leq k \leq K$: let $y \in \mathbf{R}^m$ correspond to dual variables for the constraints (7) in the LP dual to (6)-(9) (given in (12)). Given values for y (generated by the algorithm), the k^{th} *block problem* is

$$\begin{array}{rcl} \min & yA_k x^k & \\ C_k x & \geq & d_k \\ x^k & \geq & \mathbf{0}. \end{array} \qquad (10)$$

Let $\mathsf{Oracle}(y, k)$ denote the subroutine that returns a solution to (10). For the multicommodity flow problems in [13], the oracle is either a shortest path algorithm or a minimum cost flow algorithm. For the generalized multicommodity flow problems in [10], the oracle is a generalized shortest path algorithm. For the problems we consider in this paper, the oracle requires a new algorithm, and hence in the subsequent sections we also describe efficient algorithms to solve the oracle problems.

3.1 Approximation Scheme

The approximation scheme for the k-block packing problem is fully described in Figure 1.

3.2 Analysis of the algorithm

An *iteration* is defined as the steps $\mathsf{blockpack}$ takes inside the outer while loop - i.e. the steps between two visits to line (2). Let x_r, y_r denote the vectors x and y at the beginning of iteration r, and suppose the algorithm stops in iteration t. Define $\hat{x} := x_t / \log_{e^\epsilon}(1/\delta)$ and $\hat{\lambda} := (t-1)/\log_{e^\epsilon}(1/\delta)$. We begin by showing that $(\hat{x}, \hat{\lambda})$ is feasible for (7)-(9). To do this we require the following simple technical lemma.

Lemma 1 *Given $f_0 = 0$ and $g_0 > 0$ and sequence $\{a_1, a_2, \ldots, a_t\}$ satisfying $0 \leq a_r \leq 1$ for all r, suppose $f_r \leq f_{r-1} + a_r$ and $g_r \geq g_{r-1} e^{\epsilon a_r}$. Then $g_r / g_0 \geq e^{\epsilon f_r}$ for all r.* ∎

Lemma 2 *$(\hat{x}, \hat{\lambda})$ is feasible for (7)-(9).*

```
blockpack($A, b,$ Oracle$(\cdot, k)\ \forall\ k$)

Input: $k$-block packing problem; Oracle$(y,k)$ given for block $k$ problem, $\forall\ k$
Output: $\epsilon$-optimal primal solution $x$ and dual solution $(y,z)$.

(1)   Initialize $\delta \leftarrow \frac{1+\epsilon}{m(1+\epsilon)^{1/\epsilon}}$, $x \equiv \mathbf{0}$, $\mathbf{y} \equiv \delta/\mathbf{b}$.
(2)   while $b^{\mathsf{T}} y < 1$,
(3)       for $k = 1$ to $|K|$,                                  – begin iteration –
(4)           $d' \leftarrow d_k$                                    – unsatisfied demand –
(5)           while $b^{\mathsf{T}} y < 1$ and $d' > \mathbf{0}$,
(5)               $x' \leftarrow$ Oracle$(y,k)$                      – $x'$ satisfies (10) –
(6)               $\sigma \leftarrow \min\{\frac{d'}{C_k x'}, \min_i \frac{b(i)}{A_k(i)x'}\}$        – step size –
(7)               $x \leftarrow x + \sigma x'$
(8)               $d' \leftarrow d' - \sigma C_k x'$
(9)               For $i = 1$ to $m$,
(10)                  $y(i) \leftarrow y(i) e^{\epsilon \frac{\sigma A_k(i) x'}{b(i)}}$           – dual update –
              end while
(11)          $z^k \leftarrow \arg\max\{d_k z^k \mid z^k C_k \le y^{\mathsf{T}} A,\ z^k \ge \mathbf{0}\}$.
          end for                                                – end iteration –
(12)      $\alpha \leftarrow \sum_k d_k z^k$.
(13)      If $b^{\mathsf{T}} y/\alpha < b^{\mathsf{T}} y^*/\alpha^*$,
              $(y^*, z^*) \leftarrow (y, z)$, $\alpha^* \leftarrow \alpha$.         – keep best solution –
      end while
(14)  return $x/\log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$, $(y^*/\alpha^*, z^*)$.
```

Figure 1: An FPTAS for K-block packing problem.

Proof: Since x starts at 0 and is never decreased, $\hat{x}$ satisfies the nonnegativity constraints (9).

The procedure stops in the first iteration t for which $b^{\mathsf{T}} y_t \ge 1$. In the first $t-1$ iterations, for every commodity k, blockpack increases x so that $C_k x \ge (t-1) d_k$. Thus, $\hat{x}, \hat{\lambda}$ satisfy (8).

By the update of x and y, when $\frac{A(i)x}{b(i)}$ increases by an additive term of $a_r := A_k(i)(\sigma x')/b(i) \le 1$, the variable $y(i)$ is multiplied by $e^{\epsilon a_r}$. By the termination criterion of $b^{\mathsf{T}} y \ge 1$, we have that $y_t(i)/y_0(i) \le e^{\epsilon}/\delta$. Thus applying Lemma 1, with $f_r = A(i)x_r/b(i)$, $g_r = y_r$, we have that $A(i)x_r/b(i)$ is $\le \log_{e^\epsilon} 1/\delta$, for all $r \le t$, which implies that x_t satisfies (7). ■

The next theorem establishes that blockpack obtains ϵ-optimal solutions. There is a technical condition that is required: $\lambda^* \ge 1$. For most applications, this is already true; but it is also easy to fix: divide d_k, $1 \le k \le K$, by a rough upper bound μ on λ^*. The resulting problem is equivalent to the original, except that the final $\hat{\lambda}$ must be divided by μ to yield an ϵ-optimal solution to the original.

Theorem 3 *Assuming $\lambda^* \geq 1$, there exists a constant c such that for $\delta = O(m^{-c/\epsilon})$, $(\hat{x}, \hat{\lambda})$ is ϵ-optimal.*

Proof: As before, x_r and y_r are the vectors x and y at the beginning of the r^{th} iteration. For each r, k let $x_{r,k}$ and $y_{r,k}$ be the corresponding vectors before starting the k^{th} commodity in the r^{th} iteration. Finally, let $x_{r,k,s}$ and $y_{r,k,s}$ be the corresponding vectors before visiting line (6) for commodity k the s^{th} time in iteration r. Let $\sigma_{r,k,s}$ be the step size at this point.

Let $\alpha(y)$ be the optimal value to the following LP with variable vectors z^k:

$$\begin{array}{rcl} \max & \sum_k d_k z^k & \\ \sum_k z^k C_k & \leq & y^\mathsf{T} A_k, \ \ \forall\, k \\ z^k & \geq & \mathbf{0}, \ \ \forall\, \mathbf{k}. \end{array} \tag{11}$$

Note that (11) is the LP dual of (10).

We obtain

$$b^\mathsf{T} y_{r,k,s} = b^\mathsf{T} y_{r,k,s-1} + \sum_i b(i) y_{r,k,s-1}(i)[e^{\epsilon \sigma_{r,k,s} A_k(i) x'_{r,k,s}/b(i)} - 1].$$

Since $e^a \leq 1 + a + a^2$ for $0 \leq a \leq 1$, we get that $b^\mathsf{T} y_{r,k,s}$ is at most $b^\mathsf{T} y_{r,k,s-1}$ plus

$$\sum_i y_{r,k,s-1}(i)[\epsilon \sigma_{r,k,s} A_k(i) x'_{r,k,s} + \epsilon^2 \sigma^2_{r,k,s} (A_k(i) x'_{r,k,s})^2 / b(i)].$$

So,

$$b^\mathsf{T} y_{r,k,s} \leq b^\mathsf{T} y_{r,k,s-1} + \epsilon(1+\epsilon) \sum_i y_{r,k,s-1}(i) A_k(i)(\sigma_{r,k,s} x'_{r,k,s}).$$

Summing over all steps for commodity k, we get

$$\begin{array}{rcl} b^\mathsf{T} y_{r,k+1} & \leq & b^\mathsf{T} y_{r,k} + \epsilon(1+\epsilon) \sum_s y_{r,k,s-1} A_k (\sigma_{r,k,s} x'_{r,k,s}) \\ & \leq & b^\mathsf{T} y_{r,k} + \epsilon(1+\epsilon) d_k z^k \end{array}$$

where the last inequality follows using strong LP duality for (10) with $y_{r,k,s-1} A_k x'_{r,k,s}$, the fact that y is nondecreasing, and $\sum_s \sigma_{r,k,s} = 1$. Summing over all commodities k, and using the fact that y and z are nondecreasing throughout, yields

$$b^\mathsf{T} y_{r+1} \ \ \leq \ \ b^\mathsf{T} y_r + \epsilon(1+\epsilon)\alpha(y_{r+1}).$$

Thus we have established a crucial inequality in the analysis presented in [13] Section 5, and in [10]. Note that the LP dual to (6)-(9) is

$$\begin{array}{rcl} \min & b^\mathsf{T} y & \\ y A_k & \geq & C_k z^k \\ \sum_k z^k d_k & \geq & 1. \end{array} \tag{12}$$

Let β be the optimal value to this LP. Thus $\beta \leq b^\mathsf{T} y_r / \alpha(y_r)$. Now we can follow the analysis in [10] exactly to obtain that for

$$\delta = (m/(1-\epsilon))^{-1/\epsilon} \tag{13}$$

the algorithm blockpack returns ϵ-optimal solutions. ■

Using arguments identical to those in [10, 13], we get the following bounds on the number of iterations and oracle calls.

Theorem 4 *For $\delta = O((m/(1-\epsilon))^{-1/\epsilon})$, algorithm* blockpack *terminates after $O(\epsilon^{-2} \log m)$ iterations, requiring at most $O(\epsilon^{-2}(m+K)\log m)$ oracle calls.* ■

Remark. To obtain an ϵ-optimal solution to (6)-(9), we actually do not require lines (11)-(13). These are used only to establish ϵ-optimality of the primal solution. In practice, it might be useful to keep these values around so that they can be used to give an alternative termination criterion. On the other hand, it is not necessary to maintain z and α. For cleaner presentation in the specialized versions of this algorithm presented in later sections, we do not include the equivalent variables and versions of these lines.

4 The Concurrent Restorable Flow Problem

We start by considering the problem (BCR) without the budget constraint. We call this the *Concurrent Restorable Flow Problem (CRF)*. This simpler "infinite budget" case provides much of the motivation for our algorithm to solve (BCR) in Section 5.

In this case, the dual variables corresponding to y in the k-block problem are $h^q(e)$, for each pair $(e,q) \in E \times Q$. Given values for h and a particular k, the oracle (10) we need for (CRF) is

$$\text{minimize} \sum_P \left[\sum_{q:P\notin\Lambda(q)} h^q(P) f(P) \;+\; \sum_{P'\in\Lambda_k\setminus P} \; \sum_{q:P\in\Lambda(q)} h^q(P') g^P(P') \right] \tag{14}$$

subject to

$$\begin{aligned} \forall k, \quad \sum_{P\in\Lambda_k} f(P) \quad &\geq \quad \lambda d_k, \\ \forall k \; \forall \; P' \in \Lambda_k, \quad \sum_{P\in\Lambda_k\setminus P'} [\, f(P) \;+\; g^{P'}(P) \,] \quad &\geq \quad \lambda d_k. \end{aligned}$$

The extreme points of the polytope described by these constraints have only two positive variables: either two values of f are nonzero, or one value of f and one value of g is nonzero. The oracle will always generate solutions that have one value f and one value g nonzero, since the objective function counts $f(P)$

for every scenario that P does not fail, but counts $g^P(P')$ only for the scenarios that P fails. Thus the oracle is an algorithm that finds two edge disjoint paths $P, P' \in \Lambda_k$ that minimize (14). Since we assume that Λ_k is a set of disjoint paths, this is equivalent to finding the shortest two paths in Λ_k.

The step size is determined by $\min\{d', \underline{u}(P), \underline{u}(P')\}$. The dual update to h is the following subroutine. For all $q \in Q$: if $P \notin \Lambda(q)$ then for all $e \in P$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u/u(e)}$; otherwise, for all $e \in P'$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u/u(e)}$. Here, the base of the exponent is the base of the natural logarithm, not to be confused with the argument for h and u, which is an edge. For ease of reference, the full approximation scheme for (CRF) is presented in Figure 2. Here, $D(h)$ is the objective function of the dual LP to (CRF). So, $D(h) := \sum_{e \in E} u(e) \sum_{q \in Q} h^q(e)$.

```
ConcurrentRecovery($u, d, \Gamma(k)$)

Input: (CRF) instance
Output: $\epsilon$-optimal solution $(f, g)$.

    Initialize $\delta \leftarrow \frac{1+\epsilon}{m|Q|(1+\epsilon)^{1/\epsilon}}$, $h^q(e) = \delta/u(e)\ \forall e \in E,\ \forall q \in Q$
    Initialize $f \equiv \mathbf{0}$, $g \equiv \mathbf{0}$
    while $D(h) < 1$
        for $k = 1$ to $|K|$ do
            $d' \leftarrow d_k$
            while $D(h) < 1$ and $d' > 0$
                $P_1, P_2 \leftarrow$ disjoint path-pair in $\Lambda_k$ that minimize (14)
                $u \leftarrow \min\{d', u(P_1), u(P_2)\}$
                $f(P_1) \leftarrow f(P_1) + u$
                $g^{P_1}(P_2) \leftarrow g^{P_1}(P_2) + u$
                $d' \leftarrow d' - u$
                for $q \in Q$
                    if $P_1 \notin \Lambda(q)$,
                        $\forall e \in P_1$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u/u(e)}$
                    else,
                        $\forall e \in P_2$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u/u(e)}$
                end for
            end while
        end for
    end while
    return $(f, g)/\log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$.
```

Figure 2: An FPTAS for the concurrent recovery flow problem.

Theorem 5 *An ϵ-optimal solution to the concurrent recovery flow problem can be computed with at most $O(\epsilon^{-2}(m|Q|+|K|)\log(m|Q|))$ oracle calls. Each oracle call uses at most $O(m|Q||\Lambda_k|^2)$ computational steps.*

Proof: Theorem 4 and Theorem 3 together imply that an ϵ optimal solution is found in $O(\epsilon^{-2}(m|Q|+|K|)\log(m|Q|))$ oracle calls. Each oracle call for commodity k requires finding 2 paths in Λ_k that minimize an objective function with $m|Q|$ terms per path. ■

5 Budget-Constrained, Concurrent Restorable Flow

For the budget constrained, concurrent restorable flow problem, the packing constraint matrix grows by one additional constraint — the budget constraint — over the linear program of the previous section. This has the effect of changing both the dual variables used by the algorithm and the coefficients in the objective function for the oracle problem. We now present our ϵ-approximation algorithm for (BCR).

The dual variables corresponding to y in the k-block problem are of two types: $h^q(e)$ for each pair (e,q) that correspond to the primal constraints (2) and a variable ϕ that corresponds to the budget constraint (4).

Given a singleton-vector pair (ϕ,h), let 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 $\nu_{P',P}$. For commodity k, the oracle (10) we need for (BCR) is

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

subject to:

$$\begin{array}{rcrcl}
\sum_{P\in\Lambda_k} f(P) & & & \geq & d_k \\
\sum_{P\in\Lambda_k;\ P\neq P'} f(P) & + & \sum_{P\in\Lambda_k,\ P\neq P'} g^{P'}(P) & \geq & d_k \quad \forall\ P'\in\Lambda_k \\
 & & f,g & \geq & \mathbf{0}
\end{array}$$

The optimal solutions to (5) takes one of two forms, as described in Lemma 6. We begin by introducing two functions used in the lemma.

$$\overline{Z}_k = \min_{P_1,P_2\in\Lambda_k:\ P_1\neq P_2} \mu_{P_1} + \nu_{P_1,P_2} \tag{15}$$

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

$$\hat{Z}_k = \min_{2 \leq r \leq |\Lambda_k|} (\sum_{i=1}^{r} \mu_{P_i})/(r-1). \tag{16}$$

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

Lemma 6 *The optimal solution to (5) 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 $f(P_1) = g^{P_1}(P_2) = d_k$, all other variables zero. Otherwise, it has value $d_k\hat{Z}_k$ and there are r distinct paths in Λ_k such that $f(P_i) = d_k/(r-1)$ 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 $d_k\hat{Z}_k$. The LP dual to (5) is

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

$$w^{P'} \quad \leq \nu_{P',P} \tag{17}$$

$$z_k + \sum_{P' \in \Lambda_k - \{P\}} w^{P'} \quad \leq \mu_P \qquad \forall \ P \in \Lambda_k \tag{18}$$

$$z, w \quad \geq \mathbf{0}. \tag{19}$$

Consider the dual solution $w(P_i) = \hat{Z}_k - \mu_i$ if $i \leq r_k$ and $z_k = 0$. This clearly satisfies (18). By Lemma 7, $w(P_i) \geq 0$ for all i, satisfying (19). Since $\hat{Z}_k \leq \overline{Z}_k \leq \mu_{P_i} + \nu_{P_i,P_j}$, we have that (17) 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 (5), 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 $f(P_1) = g^{P_1}(P_2) = d_k$ 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.

$$w(P_i) = \max\{Z_k - \mu_{P_i}, 0\} \ \ \forall \ i \leq |\Lambda_k| \tag{20}$$

and $z_k = \overline{Z}_k - \sum_i w(P_i)$. Since $\overline{Z}_k \leq \hat{Z}_k$, Lemma 7 implies that $Z_k \leq \mu_P$ for all $w(P) = 0$, and hence this solution satisfies (18). By definition, w is nonnegative. Summing (20) 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 (17) are satisfied, and the solution is feasible. It has value $d_k\overline{Z}_k$, matching the given solution to its dual (5), hence both are optimal. ■

Lemma 7 *$\hat{Z}_k \geq \mu_{P_i}$ 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 6 implies that the optimal solution to (5) may be described as a set of paths, each carrying an equal amount of flow. Without loss of generality, assume this set is $\{P_1, \dots, P_{r_k}\}$, where $r_k \geq 2$. (If $\overline{Z}_k < \hat{Z}_k$, this set is $\{P_1, P_2\}$ and $r_k = 2$.)

The algorithm for (BCR) is given in Figure 3. The step size for (BCR) determines the amount of flow sent along each path. This is $u := \min\{d_k/(r_k - 1), \min_{i \leq r_k} u(P_i), B/(\sum_{i=1}^{r_k} c(P_i))\}$. 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 3. Here, $D(h, \phi)$ is the value of the objective function of the LP dual to (BCR), which is

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

Theorem 8 *An ϵ-optimal solution to the budget-constrained, concurrent recovery flow problem can be computed with at most $O(\epsilon^{-2}(m|Q| + |K|)\log(m|Q|))$ oracle calls. Each oracle call uses at most $O(|\Lambda_k|^2 n|Q|)$ computational steps.*

Proof: Theorem 3 and Theorem 4 together imply that an ϵ optimal solution is found in $O(\epsilon^{-2}(m|Q| + |K|)\log(m|Q|))$ oracle calls. Each oracle call for commodity k requires computing values μ and ν for all paths in Λ_k, (each with $n|Q|$ terms per path), and finding the minimum of $\overline{Z}_k$ and $\hat{Z}_k$. ■

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 (or 1+1/1:1 protection) for each commodity. In our main problem, capacity used by a back-up 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

```
BudgetRecovery $(G = (\Lambda, E, c, u, d)\ )$

Initialize $\delta \leftarrow \frac{1+\epsilon}{m|Q|(1+\epsilon)^{1/\epsilon}}$, $h^q(e) = \delta/u(e)\ \forall e \in E,\ \forall q \in Q$, $\phi = \delta/B$,
Initialize $x \equiv \mathbf{0}$, $y \equiv \mathbf{0}$
while $D(h) < 1$
  for $k = 1$ to $|K|$
    $d' \leftarrow d_k$
    while $D(h) < 1$ and $d' > 0$
      $Z_k = \min\{\overline{Z}_k, \hat{Z}_k\}$.
      Let $\{P_1, P_2, \dots, P_{r_k}\}$ 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 $P_1 \notin \Lambda(q)$
            $\forall e \in P_1$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u_0/u(e)}$.
          else
            $\forall e \in P_2$, $h^q(e) \leftarrow h^q(e)e^{\epsilon u_0/u(e)}$.
        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 3: Algorithm for (BCR)

the fly, has been studied by Bienstock and Muratore [3] in the context of capacity expansion with integrality constraints and in the single source-destination pair by Brightwell et al. [5].

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:

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

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):

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

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:

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

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

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

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 $P \in \Lambda_k$:

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

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

7 Computational Results

We now describe our preliminary computational results for solving the problem (MELTCoRe) on several real world networks. We compare the performance of an initial implementation of our ϵ-approximation algorithm with solutions obtained from an exact LP commercial solver CPLEX [6]. 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 used. Further topological details of these instances are not possible for reasons of confidentiality. The second table lists the memory usage and running times of our ϵ-approximation and of the general LP solver on the corresponding instances. The third table presents objective function values obtained by the algorithm for each instance. It also demonstrates that even “large” values of ϵ such as 0.4 give near-optimal solutions! Since our running time upper bounds are proportional to $1/\epsilon^2$, one might expect a possible advantage from using larger values of ϵ. Indeed this is borne out, and some reasons for this “best of both worlds” (near-optimal solution and improved running time) phenomenon are presented after discussing the first two tables.

The Instances: We considered the following real-world instances in our experiments. Instances 6-9 have the same underlying network topology.

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 Machine: The runs were conducted on an 8-processor (195MHz each) SGI Origin 2000 machine, where each processor had 2 Gb of main memory. The running times reported are (rough) user-time measures obtained from the UNIX "time" function. The resident memory measurements are obtained through the "top" routine.

Memory and Run Time: The following table shows several things. First, the general solver is typically faster, especially for the smaller instances. We use the CPLEX barrier method and solve each instance to optimality. About half the time for the LP solver is spent reading in the LP file itself. In a comprehensive run-time comparison one would obviously use the callable library of CPLEX, but this fact highlights one main advantage of the combinatorial algorithm. Namely, for larger instances, the general solver requires much more resident memory, even after its row-elimination heuristics are applied. To get a handle on how the memory curves behave for the two algorithms, one can compute the number of nonzeros in the constraint matrix for MELTCoRe; this gives a rough lower bound on the CPLEX memory requirements. For the largest instances the process is actually killed due to memory needs beyond the available memory. Conversely, the approximation algorithm requires much less memory – about 10 percent on the problems considered. On instances 6-9 – which have the same underlying network – the combinatorial 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

Using Higher ϵ Values: 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 $\epsilon = 0.4$. Let us examine the results produced by our algorithms, and the reasons for this desirable phenomenon (of even large values of ϵ producing near-optimal values). Recall the interpolation search mechanism that we discuss early in Section 2.1. 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. The advantage of using large values of ϵ is that we obtain reduced running times. 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 an approximation algorithm for routing with shared protection in (optical core) networks that scales well as a function of network size. 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.

Related previous approximation schemes relied on repeatedly solving shortest path or shortest column problems. In the present setting, a more complex auxiliary LP must be solved since the main problem is neither of simple packing nor covering type. We have implemented our algorithm, and given empirical results on a family of real-world instances. These computations show that the proposed scheme works well even for large real-world instances of this problem, whereas general-purpose LP solvers do not scale to the largest instances. Future work is required to develop an optimized version of the code to make it competitive with general purpose solvers for smaller instances.

An 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 mathematical questions have remained unresolved. The first is to produce a provable approximation scheme for the minimum cost network design problem for restorable flows (as opposed to the expected cost model). Ths second 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 $\gamma_k \in [0,1]$.

Finally, an analytical or empirically-supported theory of how well such algorithms perform as a function of ϵ, would be of interest. Such a theory could explain how or when even "large" values of ϵ suffice (as discussed in Section 7), and hence could drive practical algorithms in this domain.

Acknowledgements: The authors are grateful to Bill Cook who gave a number of very helpful comments on the content of the paper.

References

[1] D. Bienstock. Experiments with a network design algorithm using ϵ-approximate linear programs. Submitted for publication, 1996.

[2] D. Bienstock. Approximately solving large-scale linear programs. i. strengthening lower bounds and accelerating convergence. Technical report, CORC Report 1999-1, Columbia University, 1999. Extended abstract: Proc. 11th Ann. ACM-SIAM Symposium on Discrete Algorithms, January 2000.

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

[4] Daniel Bienstock. *Potential function algorithms for approximately solving linear programs: theory and practice.* International Series in Operations Research and Management Science. Kluwer Academic Publisher, 2002.

[5] G. Brightwell, G. Oriolo, and B. Shepherd. Some strategies for reserving resilient capacity in a network. *SIAM Journal of Discrete Mathematics*, 14(4):524–539, 2001.

[6] Cplex 6.6.1. ILOG, Inc. http://www.ilog.com/products/cplex/.

[7] L. Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004. To appear.

[8] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[9] L. K. Fleischer, A. W. Meyerson, I. Saniee, F. B. Shepherd, and A. Srinivasan. Near-optimal design of MPλS tunnels with shared recovery. Working paper, 2001.

[10] L. K. Fleischer and K. D. Wayne. Fast and simple approximation schemes for generalized flow. *Math. Programming*, 91:215–238, 2002.

[11] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Res. Logistics Quarterly*, 3:149–154, 1956.

[12] N. Garg and R. Khandekar. Fast approximation algorithms for fractional Steiner forest and related problems. In *43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 500–, 2002.

[13] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[14] R. Giles, K. Kumaran, D. Mitra, C. Nuzman, and I. Saniee. Selective transparency in optical networks. In *Proc. of ECOC2002*, Copenhagen, October 2002.

[15] A. V. Goldberg, J. D. Oldham, S. Plotkin, and C. Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 338–352, Berlin, 1998. Springer.

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

[17] M. D. Grigoriadis and L. G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21:321–340, 1996.

[18] M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. Approximate max-min resource sharing for structured concave optimization. *SJO*, 11(4):1081–1091, 2001.

[19] G. Karakostas. Faster approximation schemes for fractional multicommodity flow. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[20] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 18–25, 1995.

[21] P. Klein, S. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23:466–487, 1994.

[22] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50:228–243, 1995.

[23] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[24] T. Radzik. Fast deterministic approximation for the multicommodity flow problem. *Mathematical Programming*, 78:43–58, 1997.

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

[26] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318–334, 1990.

[27] N. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.

[28] N. Young. Sequential and parallel algorithms for mixed packing and covering. In *42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001. 538-546.