Approximate Multicommodity Flow for WDM Networks Design

Mohamed Bouklit, David Coudert, Jean-François Lalande, Christophe Paul, Hervé Rivano

Approximate Multicommodity Flow for WDM Networks Design*

M. Bouklit[‡] D. Coudert[§] J-F. Lalande[§] C. Paul[‡] H. Rivano[§¶]

Abstract

The design of WDM optical networks is an issue for telecom operators since the spreading of this technology will not occur unless enough performance guarantees are provided. Motivated by the quest for efficient algorithms for the Routing and Wavelength Assignment problem (RWA), we address approximations of the fractional multicommodity flow problem which is the central part of a complex randomized rounding algorithm for the integral problem. Through the use of dynamic shortest path computations and other combinatorial approaches, we improve on the best known algorithm. We also provide directions for further improvements.

Keywords: Multicommodity flow, WDM networks design, RWA, dynamic shortest paths.

1 Introduction

Optical networks with wavelength division multiplexing (WDM) technology seem to offer the most suitable solution to near future backbone networks for they can provide an unchallenged bandwidth. Nevertheless, the deployment of this type of networks needs dramatic investment and would not happen unless one can provide enough efficiency guarantees. One of the main factor to guarantee is to design the network in order to fit the traffic to be carried and to avoid waste of high-cost resources. This problem has been widely addressed in the literature from both theoretical and practical viewpoints and under different settings and models. [2, 3, 8–11].

In the following, we consider multifiber optical backbone networks deploying the WDM technology. In our model, the traffic is static and given as a set of *communication requests* for each of which a *lightpath* has to be assigned. The design of such networks to fit a set of communication requests is obtained by minimizing the resources (amount of fibers on each link and number of available wavelengths) required for a Routing and Wavelength Assignment (RWA) to exist. In a previous work [2], we have shown that this problem is efficiently modeled by an integer multicommodity flow on an auxiliary graph.

The objective of this article is to present several improvements on the best known combinatorial approximation algorithm for the fractional multicommodity flow problem proposed in [4]. This algorithm is to be used as the main function in an improvement of the randomized approximation algorithm for integer multicommodity flow given in [2].

[‡]Algorithmic and Combinatorics team, 161 rue Ada, F-34392 Montpellier Cedex, France.

[§]MASCOTTE project, CNRS-I3S-INRIA, 2004 route des Lucioles, B.P. 93, F-06902, Sophia Antipolis Cedex, France.

[¶]Hervé Rivano is also with France Telecom R&D.

*Work partially funded by the European projects RTN ARACNE and FET CRESCCO, and the action COLOR DYNAMIC.

Multicommodity flow The multicommodity flow problem is useful in numerous applications, especially when the issue is to compute paths for entities that are concurrent for some resources. In these cases, the resources are modeled by capacities on the edges of the graph bearing all possible routings.

More mathematically, a *flow network* is a graph $G = (V, E)$ with c a capacity function on the edges. The edges would support the flow, and the vertices would be intermediary points, flow sources, or destinations. We are given a multiset of *commodities* $C = \{(s_i, t_i),\ s_i, t_i \in V,\ i = 1, \ldots, k\}$. A *flow* f_i from s_i to t_i is a function on the edges fitting *flow conservation constraints* [5]. The flow *intensity* is the amount of flow leaving the source s_i, which equals to the amount arriving at t_i. A *multicommodity flow* of C is an union of flows for each commodity, such that the sum fits the capacity of $G : \forall e \in G,\ \sum_i f_i(e) \leq c(e)$.

If the flow functions are constrained to be integer, the problem is $\mathcal{NP}$-hard and non approximable in general. Nevertheless, Raghavan [12] proposed a randomized algorithm constructing a multicommodity flow based on the randomized rounding of the linear relaxation of the problem, namely the fractional multicommodity flow. In this setting where the flow functions are taking values in $\mathbb{R}^+$, the fractional multicommodity flow can be solved in polynomial time and space since it can be written as a polynomial size linear program. Unfortunately, the computing time and space grow fast with G and C: the average complexity of the simplex method to solve linear program of this size is $O\left(|C|^3|E|^2|V| + |C|^2|E|^3\right)$ [1]. Recently an improvement of Raghavan's algorithm has been proposed in [2]. This algorithm computes a better approximation of the integer multicommodity flow, but at the cost of solving a fractional multicommodity flow for each commodity in an iterative process. Solving the fractional multicommodity flow efficiently is therefore a crucial issue.

Approximating the fractional multicommodity flow Given that the previous algorithm uses fractional multicommodity flows to compute an approximated integer solution, there is no real need for the use of optimal fractional flows. Indeed, the approximation yielded by the randomized rounding process is an additive gap of order the square root of the fractional capacities [2, 12]. If these fractional capacities are $(1+\epsilon)$-approximations of the optimal capacities, the magnitude of the final gap will not change dramatically. It is therefore natural to exploit this freedom in order to consequently speed up the integer multicommodity flow approximation algorithm. A first article [7] proposes a combinatorial algorithm computing a fractional multicommodity flow $(1 + \epsilon)$-approximation in time $O(|C||E|^2|V|^2\epsilon^{-2})$. It is then possible to avoid the linear program solvers, much more costly to run. This algorithm has been improved by Fleischer in [4], decreasing the complexity to $O(|E|^2 \log |V|\epsilon^{-2})$. Both algorithms are mainly based on sequences of shortest path computations.

In the following, we propose an improvement on Fleischer's algorithm by concentrating on this remark, both in general and in the specific setting of WDM networks design.

2 Fleischer's algorithm

The $(1 + \epsilon)$-approximation algorithm for fractional multicommodity flow proposed by Fleischer in [4] and reported as Algorithm 1 is based on a combinatorial understanding of the dual of the multicommodity flow *edge-path* linear program. This dual builds a length function l on the edges of the graph such that the length of an edge is related to the amount of flow it bears. Therefore, Algorithm 1 builds a valid maximum flow but does computations only on the length function.

The process starts by assigning to each edge the same initial length $\delta > 0$, a constant which depends on the parameters of the algorithm, including the approximation factor ϵ, and corresponds to a null amount of flow (line 1). Then, the algorithm pushes flow iteratively along single source shortest paths (SSSP) for each commodity (lines 3–11) in the following manner. At each step, for each commodity, a shortest path P from the source to the destination is computed, c_m is the minimum capacity over the edges of P. The

Algorithm 1 Fleischer's multicommodity flow $(1+\epsilon)$-approximation [4]

Input: $G=(V,E),\ |V|=n,\ |E|=m,\ C=\{(s_i,t_i),\ i=1\dots k\}\subseteq V^2, \epsilon>0$
Ouput: l length function over E s.t. every path $s_i \rightarrow t_i$ has length > 1.
Ouput: f $(1+\epsilon)$-approximation of max. multicommodity flow for C on G

1: {Initializing} $\forall e\in E, l(e)=\delta=(1+\epsilon)((1+\epsilon)n)^{-\frac{1}{\epsilon}}$, $f_i(e)=0$
2: {Lower bound on the length of a SSSP } $\lambda=\delta$
3: **while** $\lambda \leq 1+\epsilon$ **do**
4: **for all** $i=1\dots k$ **do**
5: $P \leftarrow$SSSP $(s_i \rightarrow t_i)$
6: **while** $l(P)\leq(1+\epsilon)\lambda$ **do**
7: $c_m \leftarrow \min_{e\in P} c(e)$
8: $\forall e\in P,\ f_i(e)\leftarrow f_i(e)+c_m/(\log_{1+\epsilon}\frac{1+\epsilon}{\delta})$
9: $\forall e\in P,\ l(e)\leftarrow l(e)(1+\epsilon\frac{c_m}{c(e)})$
10: $P \leftarrow$SSSP $(s_i \rightarrow t_i)$
11: $\lambda\leftarrow\lambda(1+\epsilon)$

amount of flow $c_m/(\log_{1+\epsilon}\frac{1+\epsilon}{\delta})$ is pushed along P while the length of each edge e of P is multiplied by $1+\epsilon\frac{c_m}{c(e)}$ (lines 5–10). The algorithm ends when every shortest paths from the source of a commodity to its destination are of length more than 1.

The analysis of the algorithm mainly relies on the following idea, given in [7] and adapted to Algorithm 1 in [4]. Each edge starts with length δ and ends with length at most $1+\epsilon$. Each time some flow is pushed, the length of an edge is multiplied by $1+\epsilon$ (the one with the minimum capacity). In the worst case, each computed SSSP is made of only one edge, and therefore, there are at most $m\log_{1+\epsilon}\left(\frac{1+\epsilon}{\delta}\right)$ iterations. Considering this result, δ is carefully chosen and there are, at most, $O(m\epsilon^{-2}\log_{1+\epsilon}n)$ iterations, each of which corresponding to a SSSP computation and a push of flow. Using the classical Dijkstra algorithm for SSSP, the cost of each iteration is $O(m+n\log n)$. This yields the following worst case complexity for Algorithm 1.

$$O\left(\frac{m\log_{1+\epsilon}n}{\epsilon^2}(m+n\log n)\right)$$

3 Dynamic shortest paths

Fleischer's algorithm computes several consecutive SSSP for a given commodity $s\rightarrow t$ (lines 6–10 of Algorithm 1). Between each iteration the length l of the edges of the previous SSSP is increased. The knowledge of these modifications can be used to update the shortest path tree $T(s)$ at a lower cost than a new complete computation with Dijkstra algorithm. Indeed, the increase of the length may only impact a small part of the tree. This issue has been widely addressed in the literature and the best-known algorithm for the case where the length of a single edge is increased is given in [6]. In this section, we first sketch the algorithm presented in [6], which is divided into two main steps. We then show how this algorithm can be adapted with the same complexity (in a worst case analysis) to update the shortest path tree when the lengths of all the edges of a $s\rightarrow t$ shortest path are increased.

The first step of the algorithm in [6] colors the vertices in order to know whether they are subjected to modifications or should not be part of the computation as follows.

- $q\in V$ is **red** if its distance from s increases.
- $q\in V$ is **pink** if only its father in $T(s)$ changes.
- $q\in V$ is **white** if neither its distance nor its father changes.

The second step takes as input such a red-pink-white coloration with the set of red vertices stored in a queue, while the pink and white colors are implicit. It computes the shortest path tree updates by considering the red vertices in non-decreasing order with respect to the distance from s as in Dijkstra's algorithm.

Indeed, the second step considers only the red vertices. The following lemma proves the correctness of the combination of these two steps.

Lemma 1 ([6]) *When the length of one edge increases, if a good red-pink-white coloration of the vertices is given to the second step, a valid shortest path tree is produced.*

Lemma 1 implies that it is enough to modify the first step to adapt to the case of the increase of the lengths of all the edges of $s \rightarrow t$, the shortest path from s to t. Algorithm 2 produces such a good red-pink-white coloration of the vertices and Lemma 2 proves its correctness. In the following, for each $u \in V$, $D(u)$ represents the distance from s to u and $P(u)$ stands for the father of u in $T(s)$.

Algorithm 2 Coloration of the vertices for USSSP

Input: $T(s_i)$ a shortest path tree rooted in s_i, $s_i \rightarrow t_i$ a shortest path.
Ouput: a set of red colored vertices.

1: {Initializing} $\forall y \neq s_i$ a vertex of $s_i \rightarrow t_i$, Enqueue(M,$\langle y, D(y)\rangle$).
2: **while** Non-Empty(M) **do**
3: $\langle z, D(z)\rangle \leftarrow$ Extract-Min(M).
4: **if** there is a nonred neighbor $q \notin M$ of z such that $D(q) + c(q, z) = D(z)$ **then**
5: $P(z) \leftarrow q$ {z is pink}
6: **else**
7: $color(z) \leftarrow$ red
8: **for all** child v of z not in M **do**
9: Enqueue(M,$\langle v, D(v)\rangle$)

The idea of Algorithm 2 is similar to the coloring in [6] and only the initialization really differs. Algorithm 2 starts by inserting all the vertices of $s \rightarrow t$ in a priority queue M (line 1). This queue is to contain only vertices that are impacted by the increase of length (the red or pink vertices). Then, while M is not empty (line 2), the closest vertex from s in M (line 3) is processed. If its distance does not change (line 4), its father is updated, the vertex is implicitly colored pink (line 5) and the whole sub-tree rooted at this vertex is implicitly colored white since none of its vertices have to be inserted in M. If the distance from s of the vertex increases, it is colored red (line 7) and all its children are inserted in M if they are not yet enqueued since they have to be at least pink (lines 8, 9).

Lemma 2 *Let $G = (V, E)$ be a graph, $T(s)$ be the shortest path tree rooted at s and $s \rightarrow t$ a shortest path from s to t. If the length of each edge of the $s_i \rightarrow t_i$ path increases, then Algorithm 2 produces a valid red-pink-white coloration of the vertices of G.*

Preuve : The correctness of Algorithm 2 is a consequence of the following properties.

($\mathcal{P}_1$) Any vertex of the $s \rightarrow t$ path is inserted in M

($\mathcal{P}_2$) Any child of a red vertex is enqueued in M and no child of a pink vertex is enqueued in M unless it is in the $s \rightarrow t$ path.

($\mathcal{P}_3$) A vertex is white if and only if it is never enqueued in M.

($\mathcal{P}_4$) The vertices are extracted of M ordered by increasing distance : if u and v are enqueued in M, then $D(u) < D(v)$ implies that u is extracted before v.

($\mathcal{P}_1$) is a direct consequence of the initialization of Algorithm 2 (line 1). If a vertex is colored red, either the distance or the father of any of its child have to be modified. Therefore they have to be enqueued in M. If the vertex is pink, its distance does not change and all its sons that are not in the original $s \rightarrow t$ path are white. These facts correspond to properties ($\mathcal{P}_2$) and ($\mathcal{P}_3$). We have also to check that any vertex is enqueued only once in M. This is a direct consequence of property ($\mathcal{P}_4$) with $u = v$.

For the sake of proving ($\mathcal{P}_4$), let $t(x)$ be the date when x is dequeued of M. ($\mathcal{P}_4$) is trivially true when the first vertex is extracted. Let v be a given vertex and $\Delta(v)$ the set of vertices dequeued before v. Suppose that ($\mathcal{P}_4$) is true for any $u, u' \in \Delta(v)$. At time $t(v)$, $v = \min\{x \in M\}$. Let $P(v) \in \Delta(v)$ be the father of v.

- $\forall u \in \Delta(v)$ such that $t(u) < t(P(v)), \mathcal{P}_4$ and the fact that the father of v is closer from s than v imply that $D(u) < D(P(v)) < D(v)$.
- $\forall u \in \Delta(v)$ such that $t(P(v)) < t(u)$, note that the definition of $\Delta(v)$ forces that $t(u) < t(v)$. Moreover, since v is enqueued in M at last when $P(v)$ is extracted (lines $8, 9$), it follows that v is in M at $t(u)$. As far as M is a a priority queue, at $t(u)$ $u = \min\{x \in M\}$ which implies that $D(u) < D(v)$.

Therefore, ($\mathcal{P}_4$) is true for $\Delta(v) \cup \{v\}$. □

Complexity Note that it is difficult to give an exact estimation for the complexity of the USSSP algorithm. Indeed, in [6] the authors express it as a function of the number α of *modification updates*, i.e. the number of vertices impacted by the modification. These vertices correspond to those colored in red or pink by Algorithm 2. The authors also use the notion of *accounting function* and define a characteristic β of the graph structure which is a somehow smart maximum degree: $\beta \leq 3$ for planar graphs, $\beta \leq d$ for graphs with maximum degree d, ..., $\beta = O(\sqrt{m})$ for generic graphs. The complexity of USSSP is then $O(\alpha\beta \log n)$ but is always less than the complexity of Dijkstra's shortest path algorithm, $O(m + n \log n)$.

We now evaluate the complexity of Fleischer's algorithm with shortest path tree update algorithm.

Our modified version of the algorithm in [6] have the same complexity as the original one. Indeed, when only the first edge of the $s_i \rightarrow t_i$ path is modified, the set of red vertices computed by the red-pink-white coloration in [6] contains, in the worst case, the set of red vertices computed by our modification.

Moreover, the analysis of Fleischer's algorithm shows that between lines 4 and 10, Algorithm 1 processes m SSSP where k of them cannot be replaced by USSSP (line 5). Therefore, Algorithm 1 with shortest path tree update computes $(m - k)$ USSSP and m SSSP and the complexity, in terms of SSSP and USSSP complexities, is

$$O\left(\frac{\log_{1+\epsilon} n}{\epsilon^2}\left(k.\text{SSSP} + (m-k).\text{USSSP}\right)\right).$$

In the following, we reduce the complexity of the $(1 + \epsilon)$-approximation multicommodity flow in the context of the Routing and Wavelength Assignment problem in WDM optical networks. To simplify, we will bound the complexity of USSSP by $O(m + n \log n)$ from above, except specified otherwise.

4 Flow and RWA

Our objective is to use the fractional multicommodity flow computation for the design of optical backbone networks deploying the WDM technology. In the model where the traffic is given as a set of static communication requests requiring lightpaths, the design of WDM networks is called the Routing and Wavelength Assignment problem (RWA), that we modeled in a previous paper as an integer multicommodity flow in an auxiliary graph [2]. We also proposed an approximation algorithm for integer multicommodity flow based

on an iterative sequence of randomized rounding of fractional multicommodity flows. The main problem we had to face with this approach is that the fractional multicommodity flows are computed with linear program solvers that need fast growing time and space: the time complexity is $O\left(n^8w^3\right)$, the space complexity is at least $O\left(n^3w^2m\right)$, where n is the number of nodes of the network, m the number of links and w the number of available wavelengths.

In the following, we describe shortly the multicommodity flow model of the RWA and express the complexity of Fleisher's Algorithm 1 in this setting. Besides, we improve this algorithm by specializing the SSSP computation to the specific topology of the WDM network model.

4.1 WDM model

Our model is based on the idea that there is a direct link between routing different entities that are concurrent for some resources and computing flows: each lightpath using one wavelength is modeled by a unitary flow. Moreover, "wavelength division multiplexing" means that whatever happens to a given wavelength, it will not influence what can happen to another wavelength, in particular the routings on each wavelength are mutually independent. Therefore, the routing will be done by computing flows in an auxiliary graph with a layered structure, each layer being a copy of the network topology, one per available wavelength. In these settings, a unit of flow in the i^{th} layer models a part of a lightpath with the i^{th} wavelength in the network. As far as a link made of k optical fibers allows k lightpaths with the same wavelength to cross, the capacity of each copy of a link is k.

Let us give a detail of the auxiliary graph construction. For each source of traffic s, there is a vertex S in the auxiliary graph, uncapacited edges from S to each copy s_i of s, and a vertex S'. For each destination t of traffic from s, there is a vertex D_s_t, uncapacited edges from each copy t_i of t to D_s_t, and an edge with capacity $d(s,t)$ from D_s_t to S', where $d(s,t)$ is the number of lightpaths to be computed from s to t. An example of the auxiliary graph is illustrated in Figure 1.

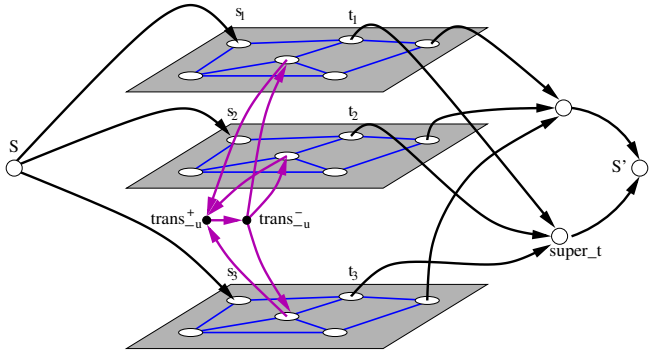

Figure 1: Auxiliary graph for 4 requests from 2 sources, $w = 3$

Therefore, a lightpath with the i^{th} wavelength from a source s to a destination t is modeled by a unitary flow in the auxiliary graph that goes from S to s_i, then crossing t_i, D_s_t and reaching S'.

Note that the edges capacities are such that a maximum multicommodity flow with commodities the set $\{(S,S'),\ \forall\ s \text{ source of traffic}\}$ saturates all the (D_s_t, S') edges if and only if there is a RWA of the requested traffic with the given resources.

Note also that the model presented in [2] also includes the presence of wavelength translation equipment on each node of the network. For that, a *translator widget* is added for each node u of the network. This widget links all the copies of node u, allowing a unit of flow to change from a layer to another. This case is

not studied in the following.

4.2 Specializing the shortest path computations

Considering the original graph G with n nodes and m links, the size of the auxiliary graph depends on the number of wavelengths w and the set of communication requests. The number of edges (t, D_s_t) equals to the number of destination the source node s has to deserve. In the case of a set of communication requests including at least once each pair of node, the auxiliary graph has $O(n^2)$ nodes and $O(wn^2)$ edges. Plugging this size of graph into Fleischer's algorithm complexity leads to a running time of $O\left(\frac{\log_{1+\epsilon} n}{\epsilon^2} wn^4(w + 2\log n)\right)$.

This complexity reflects Fleischer's algorithm efficiency on the auxiliary graph but supposes that the graph has a totally unknown topology. In our case, we can specialize the algorithm considering the following topological and logical issues.

- When computing a shortest path between S and S' only one S and one set of D_s_t is used and we can forget all other source u and destination D_u_v vertices. In this case, the considered graph has only $O(nw)$ nodes and $O(mw)$ arcs, which improves the computations of lines 5 and 10 in Algorithm 1.
- When computing the commodities and performing the increase of a path respectively in line 4 and 6, the number of iterations depends on the whole graph which size is still in $O(wn^2)$.

Consequently, the multicommodity flow complexity is improved to

$$O\left(\frac{\log_{1+\epsilon} n}{\epsilon^2} n^2 w^2 (m + n\log nw)\right).$$

4.3 Specializing the shortest path tree updates

The layered structure of the auxiliary graph illustrated in Figure 1 can be exploited to obtain significant improvements through the specialization of USSSP. Indeed one can easily be convinced that only one layer is modified when the length of each arc of a shortest path increases (lines 8 and 9 of Algorithm 1). Therefore, we can update the shortest path tree structure on this layer only and manage the update of arcs (t_i, D_s_t) and (D_s_t, S') afterward.

Using our algorithm described in Section 3, the USSSP in the concerned layer takes time $O(\alpha\beta \log n) < O(m + n\log n)$. We manage each D_s_t and S' using queues which store their best parent, this finalizes the update of the shortest path tree structure in $O(n \log w)$. Summing all, the USSSP complexity is at most $O(m + n\log nw)$.

Consequently, Fleischer's Algorithm with our improvements for WDM networks is computed in the worst case in time

$$O\left(\frac{\log_{1+\epsilon} n}{\epsilon^2} n^2 w (m + n\log nw)\right).$$

In this setting, the use of USSSP improves on the previous version by a factor of $O(w)$. Compared to the solution provided by a linear program computed in $O(n^8 w^3)$ in [2], the gain becomes $O\left(\frac{n^4 w^2 \epsilon^2}{\log_{1+\epsilon} n}\right)$.

4.4 Experiments

We did experiments on a PIV computer, 2.2GHz and 512MB of memory. We use the Belman-Ford algorithm which is more efficient than Dijkstra for graphs with less than one thousand of nodes. USSSP algorithm is also based on Belman-Ford.

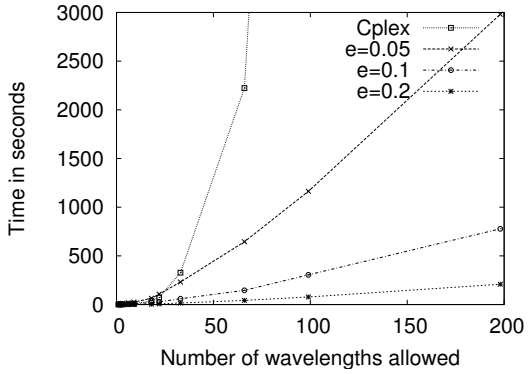

Figure 2: Running times of Algorithm 1 and CPLEX

A first set of experiments were ran on a backbone network interconnecting 65 main US cities with 78 links. This network carries 1305 communication requests between 192 pairs of cities, requiring a capacity per link ($w \times k$) equal to 198. The running times, plotted in Figure 2, show that our version of Algorithm 1 allows to address bigger networks than the linear program solver. In particular it is possible to consider *dense* WDM networks with several hundreds of wavelengths while the linear program solver CPLEX fails on the pan-american network before this critical threshold: 13180 seconds are needed for the last computable instance with 99 wavelengths. Beyond this threshold, the space requirement of the linear program overcomes the capacity of our computer.

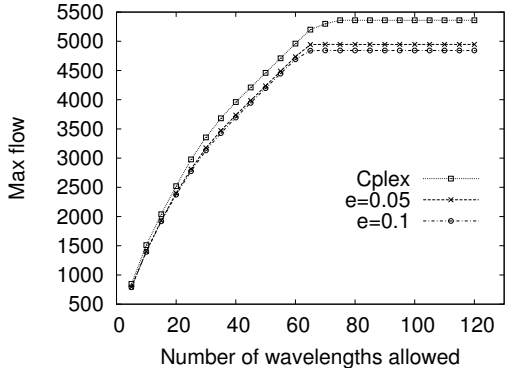

Figure 3: Approximate and optimal maximal flow values

The following runs concern the NSFNET network, described in [13]. This network is made of 14 nodes and 21 symmetric links and carries a set I of 268 requests. In order to increase the size of the problem, we multiplied each request by a factor of 20 and fixed the number of fibers per link to 5. Figure 3 presents the maximal flow values obtained by Algorithm 1 for $\epsilon = 0.05$ and $\epsilon = 0.1$ and the optimal value computed

by CPLEX. When $\epsilon = 0.1$, the value of flow is 10% less than the optimal (gap of 518 flow unities). On the other hand, the flow value should be 5% less than the maximal flow when $\epsilon = 0.05$, which is not the case here. Indeed, if ϵ is too small, numeric accuracy problems arise and degrade the tightness of approximation. These problems, already pointed in [4], are due to the simultaneous manipulation of numbers with very heterogeneous scales.

Figure 4 presents the running time for the computation of maximal flow on the NSFNET network with the set of request I multiplied by a factor κ from 1 to 20, and Figure 5 gives the ratio between these running times and the time required for I. Figure 4, as well as Figure 5, shows a logarithmic dependency to the size of the instance: for a factor of 20, the time is 4 times higher. Note that when the maximal flow is reached, about 60 wavelengths according to Figure 3, the ratio Time(κ.I)/Time(I) keeps constant.

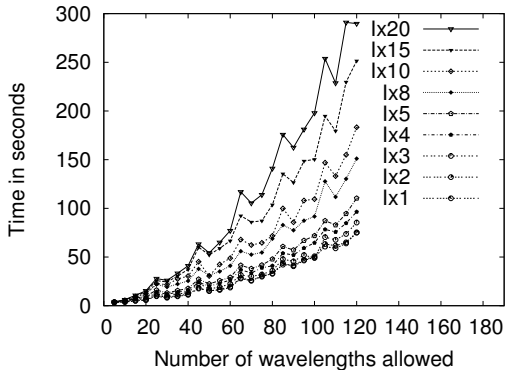

Figure 4: Running times for κI on NSFNET, $\epsilon = 0.05$

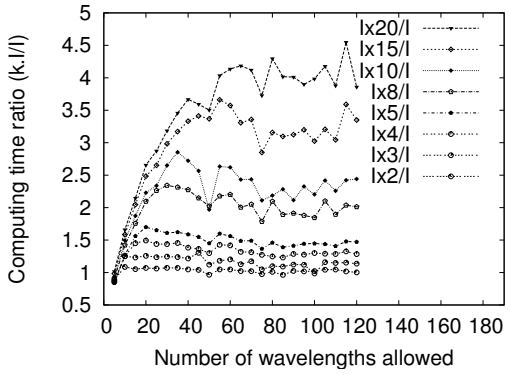

Figure 5: Ratio Time(κ.I)/Time(I), $\epsilon = 0.05$

5 Perspectives

In this paper, we have proposed an improvement of the most efficient known algorithm to approximate fractional multicommodity flow. We have shown that using a dynamic algorithm for updating the shortest path trees allows for a significant speed up of the process, even if a precise evaluation of the complexity gain is not possible.

When focusing on approximating the Routing and Wavelength Assignment in WDM optical networks, we mixed this approach with specialized shortest path algorithms on our layered structured model. Consequently, we have shown a dramatic complexity gain compared to the former use of linear program solvers. Therefore, it is now possible to address the design of WDM optical networks of higher and more realistic size and capacity (like DWDM networks), which appeared as a limitation of the approach presented in [2].

Nevertheless, further challenges are coming ahead. The implementation of this algorithm yielded numeric accuracy problems which could avoid a practical use if high precision is required. We are currently investigating rescaling methods applied to the length function to cope with these difficulties.

Another approach is also investigated. Starting from a multicommodity flow violating the capacities, this approach would transform it smoothly into a valid solution using Lagrangian relaxation and gradient descent methods.

References

[1] BERTIMAS, D., AND TSITSIKLIS, J. N. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[2] COUDERT, D., AND RIVANO, H. Lightpath assignment for multifibers WDM optical networks with wavelength translators. In *IEEE Globecom'02* (Taiwan, 2002). OPNT-01-5.

[3] FERREIRA, A., PÉRENNES, S., RICHA, A. W., RIVANO, H., AND MOSES, N. S. Models, Complexity and Algorithms for the Design of Multifiber WDM Networks. In *IEEE ICT'03* (Papeete, French Polynesia, 2003).

[4] FLEISCHER, L. Approximating fractional multicommodity flows independent of the number of commodities. *SIAM J. Discrete Math. 13*, 4 (2000), 505–520.

[5] FORD, L., AND FULKERSON, D. *Flows in Networks*. Princeton University Press, 1962.

[6] FRIGIONI, D., MARCHETTI-SPACCAMELA, A., AND NANNI, U. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms 34*, 2 (2000), 251 – 281.

[7] GARG, N., AND KONEMANN, J. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science* (1998), pp. 300–309.

[8] KRISHNASWAMY, R., AND SIVARAJAN, K. N. Algorithms for Routing and Wavelength Assignment Based on Solutions of LP–Relaxations. *IEEE Communications Letters 5*, 10 (Oct. 2001), 435–437.

[9] KUMAR, V. Approximating arc circular colouring and bandwidth allocation in all-optical ring networks. In *APPROX'98, LNCS 1444* (1998), K. Jansen and J. Rolim, Eds.

[10] LI, G., AND SIMHA, R. On the Wavelength Assignement Problem in Multifiber WDM Star and Ring Networks. *IEEE Infocom 3* (2000), 1771–1780.

[11] MARGARA, L., AND SIMON, J. Wavelength assignment problem on all-optical networks with k fibres per link. In *ICALP'00, LNCS 1853* (2000), U. Montanari, J. Rolim, and E. Welzl, Eds., pp. 768–779.

[12] RAGHAVAN, P. Probabilistic construction of deterministic algorithm: Approximating packing integer programs. *Journal of Computer and Systems Sciences 38* (1994), 683–707.

[13] SWAMINATHAN, M., AND SIVARAJAN, K. Practical routing and wavelength assignment algorithms for all optical networks with limited wavelength conversion. In *IEEE ICC* (New-York City, 2002). IO4 - 4.