Alcom-FT Technical Report Series
ALCOMFT-TR-01-29

Wavelength Rerouting in Optical Networks, or the Venetian Routing Problem*

Alberto Caprara † Giuseppe F. Italiano ‡ G. Mohan §
Alessandro Panconesi ¶ Aravind Srinivasan ||

Abstract

Wavelength rerouting has been suggested as a viable and cost-effective method to improve the blocking performance of wavelength-routed Wavelength-Division Multiplexing (WDM) networks. This method leads to the following combinatorial optimization problem, dubbed Venetian Routing. Given a *directed multigraph* G along with two vertices s and t and a collection of pairwise arc-disjoint paths, we wish to find an st-path which arc-intersects the smallest possible *number* of the given paths. In this paper we prove the computational hardness of this problem even in various special cases, and present several approximation algorithms for its solution. In particular we show a non-trivial connection between Venetian Routing and Label Cover.

Key words: Optical networks, wavelength rerouting, wavelength-division multiplexing, shortest paths, approximation algorithms, label cover.

1 Introduction

Recent excavations in the once-vibrant merchant city of Venice have unearthed documents which show that as early as the Renaissance, complicated optimization problems were studied for state planning purposes. One of the most interesting problems to come

*A preliminary version of this work appears in the *Proc. Third International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, 2000.*

†DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy, e-mail: `acaprara@deis.unibo.it`. Work partially supported by CNR and MURST, Italy.

‡Dipartimento di Informatica, Sistemi e Produzione, Università di Roma *Tor Vergata*, Roma, Italy, email: `italiano@info.uniroma2.it`. Work partially supported by the IST Programme of the EU under contract n. IST-1999-14.186 (ALCOM-FT), by the Italian Ministry of University and Scientific Research (Project “Algorithms for Large Data Sets: Science and Engineering”), and by CNR, the Italian National Research Council, under contract n. 00.00346.CT26.

§Dept. of Electrical and Computer Engineering, National University of Singapore, Singapore. E-mail: `elegm@nus.edu.sg`

¶DSI, Università *La Sapienza*, via Salaria 113, piano terzo, 00198, Roma, Italy, e-mail: `ale@dsi.uniroma1.it`. This work was done when this author was at Universitá Ca' Foscari di Venezia and Universitá di Bologna

||Bell Laboratories, Lucent Technologies, 600-700 Mountain Avenue, Murray Hill, NJ 07974-0636, USA, e-mail: `srin@research.bell-labs.com`. Part of this work was done while this author was at the School of Computing of the National University of Singapore, Singapore 119260, and while visiting the Università “Ca' Foscari” di Venezia, Italy.

to light is the following. As is well-known, Venice is built on an archipelago of several hundred islands in the midst of a lagoon. Even today (miraculously) there are no roads and one has to either walk or go by boat. While today it is possible to walk between any two points inside the city– in modern terminology, the underlying graph is connected– in spite of the many wonderful little bridges, this was not the case back then. The citizenry could make use of a rather sophisticated and pervasive mass transit system, which was a huge source of revenues for the City of Venice, consisting of the characteristic Venetian boats– the gondolas– shuffling back and forth along predefined routes. The Municipality adopted a peculiar ticketing scheme. Tickets would be expensive but each ticket could be used repeatedly, as many times as one wanted in a single day, for the same route. But, to take a gondola covering a different route would require a new ticket. From the point of view of a citizen going from point A to point B the problem was that of minimizing the number of routes. The historical record shows that the municipality was concerned about how to make such reckoning virtually impossible, and that at some point it switched to a system in which routes were unidirectional. Here we show why this was advantageous in order to increase state revenues and how this old problem can shed light on more recent ones.

Old wine, new bottles. We will be concerned with an optimization problem called Venetian Routing (VR), and with some of its relatives. The input to VR consists of a *directed multigraph* G, two vertices s and t of G, respectively called the *source* and the *sink*, and a collection of *pairwise arc-disjoint* paths in G. These paths will be referred to as *special* or *laid-out* paths. The goal is to find an st-path which arc-intersects the smallest possible *number* of laid-out paths. (Recall that G is a multigraph. If e_1 and e_2 are parallel arcs, it is allowed for one laid-out path Q_1 to contain e_1, and for another laid-out path Q_2 to contain e_2. But Q_1 and Q_2 cannot both contain e_1.) Note that the solution to the problem can be assumed to be a *simple* st-path without loss of generality.

Besides state-planning in medieval Venice, this problem arises from wavelength rerouting in Wavelength-Division Multiplexing (WDM) networks. In WDM networks, *lightpaths* are established between pairs of vertices by allocating the same wavelength throughout the path [4]: Two lightpaths can use the same fiber link if they use a different wavelength. When a connection request arrives, a proper lightpath (i.e., a route and a wavelength) must be chosen. The requirement that the same wavelength must be used in all the links along the selected route is known as the wavelength continuity constraint. This constraint often leads to poor blocking performance: A request may be rejected even when a route is available, if the same wavelength is not available all along the route. To improve the blocking performance, one proposed mechanism is *wavelength rerouting*: Whenever a new connection arrives, wavelength rerouting may move a few existing lightpaths to different wavelengths in order to make room for the new connection. Lee and Li [15] proposed a rerouting scheme called "Parallel Move-To-Vacant Wavelength Rerouting (MTV-WR)". Given a connection request from s to t, rather than just blocking the request if there is currently no available lightpath from s to t, this scheme does the following. Let $\mathcal{A}(W)$ denote the set of current lightpaths that use wavelength W and that may be migrated to some other wavelength. That is, for each $Q \in \mathcal{A}(W)$, there is some $W' \neq W$ such that no lightpath with wavelength W' uses any arc of Q. Separately for each W, we try to assign the new connection request to wavelength W. To this end, we solve VR on the sub-network obtained by deleting all wavelength-W lightpaths not lying in $\mathcal{A}(W)$; the set of laid-out paths is taken to be $\mathcal{A}(W)$. (Since all lightpaths with the same wavelength must be arc-disjoint, the "arc-disjointness" condition of VR is satisfied.) If there is no feasible solution to any of these VR instances, the connection request is rejected. Otherwise, let P be an st-path representing the best solution obtained over all W. By migrating each lightpath arc-intersected by P to an available wavelength, we obtain an st-path with minimal disruption of existing routes. In the case of directed multi-ring network topologies, such an approach achieves a reduction of about 20% in the blocking probability [16].

Thus, VR naturally arises in such rerouting schemes, helping in dynamic settings where future requests cannot be predicted and where one wishes to accommodate new requests reasonably fast without significantly disrupting the established lightpaths. VR is also naturally viewed as finding a minimum number of laid-out paths needed to connect s to t. Since these paths have to be disrupted in order to honor the incoming request, this measure is very significant. Lee and Li gave a polynomial-time algorithm for VR on *undirected* networks [15]. This algorithm was later improved in [17]. To the best of our knowledge, no result (including the possible hardness of exactly solving VR) was known for general directed networks.

A natural generalization of VR is to associate a priority with each laid-out path, where we wish to find an st-path P for which the sum of the priorities of the laid-out paths which arc-intersect P is minimized; we call this *Priority* VR or PVR. An important special case of PVR is *Weighted* VR or WVR, where the priority of a laid-out path is its length. This is useful in our wavelength rerouting context, since the effort in migrating a lightpath to a new wavelength, is roughly proportional to the length of the path. Finally, one could look at VR from the complementary angle: Finding a maximum number of laid-out paths that can be removed without disconnecting t from s. Naturally, this variant will be referred to as the GONDOLIER AVOIDANCE problem (GA).

1.1 Our results

In this paper we study the worst-case complexity of VR and of its variants, and present several hardness results and approximation algorithms. In particular, we show an interesting connection between VR and the SYMMETRIC LABEL COVER (SLC) problem [1, 13, 6] which allows us to obtain new upper bounds for the latter. In what follows, let ℓ denote the maximum length (number of arcs) of any laid-out path, and n, m respectively be the number of vertices and arcs in G. Let $\mathcal{QP}$ stand for quasi-polynomial time, i.e., $\bigcup_{c>0} \mathrm{DTIME}[2^{(\log n)^c}]$. For lower bounds, we establish the following:

(1.1) VR is APX-hard even when $\ell = 3$ (actual lower bound is 183/176). If $\ell \leq 2$, the problem is polynomially solvable.

(1.2) When ℓ is unbounded things get worse, for we exhibit an approximation preserving (L-) reduction from SLC to VR which implies that any lower bound for SLC also holds for VR. In particular, using the hardness of approximation result for SLC mentioned in [13, 6], this shows that approximating VR within $O(2^{\log^{\frac{1}{2}-\varepsilon} m})$, for any fixed $\varepsilon > 0$, is impossible unless $\mathcal{NP} \subseteq \mathcal{QP}$.

(1.3) We exhibit an approximation preserving (L-) reduction from SET COVER to VR with unbounded ℓ. This implies two things. First, if $\mathcal{NP} \neq \mathcal{P}$, a weaker hypothesis than $\mathcal{NP} \not\subseteq \mathcal{QP}$, there exists a constant $c > 0$ such that VR cannot be approximated within $c \log n$ if ℓ is unbounded. Second, GA is at least as hard to approximate as INDEPENDENT SET.

We also give several approximation algorithms. All algorithms reduce the problem to shortest path computations by using suitable metrics and (randomized) preprocessing. We remark that almost all of our algorithms actually work for a more general problem than VR defined as follows. The input consists of a multigraph G, a source s and a sink t, and a collection of *disjoint sets of arcs* $S_1, \ldots, S_k$, called *laid-out sets*. The goal is to find an st-path which arc-intersects the smallest possible number of laid-out sets S_i. In other words, we have removed the constraint that the laid-out sets be paths. We refer to this problem as GENERALIZED VR or GVR for short. Clearly, all previously described lower-bounds apply directly to GVR.

Let SSSP(n, m) denote the time complexity of single-source single-sink shortest paths on an n-node, m-arc directed multigraph with non-negative lengths on the arcs. Given an optimization problem, let *opt* denote the optimal solution value for a given instance of the problem. We now summarize our results concerning upper-bounds.

(2.1) For VR, we give a linear-time algorithm for the case $\ell \leq 2$ and an $\lceil \ell/2 \rceil$-approximation algorithm for the general case. The same approach yields: (i) an $\lceil \ell/2 \rceil$-approximation algorithm for PVR running in $O(\text{SSSP}(n, m))$ time, and (ii) an ℓ-approximation algorithm for GVR running in $O(m)$ time.

(2.2) We give an $O(\sqrt{m/opt})$-approximation algorithm for GVR. In view of the reduction described in point (1.2) above, this will yield an $O(\sqrt{q/opt})$-approximation for SLC, where q and *opt* denote the input size and optimal solution value respectively of an SLC instance. This is the first non-trivial approximation for SLC, to our knowledge.

(2.3) We show that for any fixed $\varepsilon > 0$, there is a $2^{O(m^\varepsilon \log m)}$-time $O(\sqrt{m^{1-\varepsilon}})$-approximation algorithm for VR. In particular, this shows a "separation" of VR (and hence SLC) from CHROMATIC NUMBER, as the same approximation cannot be achieved for the latter under the hypothesis that SAT cannot be solved in time $2^{n^{o(1)}}$. We also show that, if $\mathcal{NP} \neq \mathcal{P}$, there can be no L-reduction from CHROMATIC NUMBER to VR.

(2.4) Improving on (2.2), we give an $O\left(\sqrt{m/(opt \cdot n^{c/opt})}\right)$-approximation algorithm for GVR, for any constant $c > 0$; this also yields an $O(\sqrt{m/\log n})$-approximation. The algorithm, which is randomized, is based on a technique dubbed "sparsification" which might be useful in other contexts. The algorithm can be derandomized.

(2.5) We give an $O(\min\{opt, \sqrt{m}\})$-approximation for WVR running in $O(m \log m)$ time. Moreover, we show that all approximation results for VR can be extended to PVR if one replaces m and n in the approximation ratio by mw, where w is the maximum weight of a laid out path, which can be assumed to be polynomial in m.

Finally, we perform an experimental analysis of our algorithms. In the analysis we are *not* concerned with their quality of approximation in the worst case. Rather we measure them with respect to what in practical situations is a more relevant performance parameter, namely the blocking probability (the average number of requests that can be honored instead of being rejected). Our tests show that our algorithms can lead to an improved performance at the expense of a moderate computational effort, while at the same time the solutions they found are of good quality, i.e., paths consist of few hops.

1.2 Related work

Recent work of [3], done independently of our work, has considered the following *red-blue set cover problem.* We are given a finite set V, and a collection E of subsets of V. V has been partitioned into *red* and *blue* elements. The objective is to choose a collection $C \subseteq E$ that minimizes the number of red elements covered, subject to all blue elements being covered. Here, $C = \{f_1, f_2, \ldots, f_t\}$ *covers* $v \in V$ iff $v \in f_1 \cup f_2 \cup \cdots \cup f_t$. Letting k denote the maximum number of blue elements in any $f \in E$, a $2\sqrt{k|E|}$-approximation algorithm for this problem is shown in [3]. It is also proved in [3] that the red-blue set cover problem is a special case of GVR, and that there is an approximation-preserving L-reduction from SLC to the red-blue set cover problem, thus getting the same hardness result for GVR as we do in (1.2) above. Furthermore, red-blue set cover is shown to be equivalent to *Minimum Monotone Satisfying Assignment* in [3]. It is not clear if there is a close relationship between red-blue set cover and VR; in particular, the results of [3] do not seem to imply our results for VR, or our approximation algorithms for GVR. Also, a variant of GVR where we have laid-out sets and where we wish to find a spanning tree edge-intersecting the minimum number of laid-out sets, has been studied in [14].

1.3 Preliminaries

The number of vertices and arcs of the input multigraph G will be denoted by n and m, respectively. Given any VR instance, we may assume without loss of generality that every node is reachable from s, and t can be reached from each node, implying

$m \geq n$ if we exclude the trivial case in which G is a path. The maximum length of a laid-out path will be denoted by ℓ, and the number of laid-out paths by p. As G is a multigraph, in principle m may be arbitrarily large with respect to n. In fact, for each pair of vertices u, v, we can assume that there is at most one arc uv not belonging to any laid-out path, whereas the other arcs of the form uv will belong to distinct laid-out paths. Hence, we can assume $m \leq n^2 + \ell p \leq n(n+p)$. Moreover, the fact that the laid out paths are arc disjoint implies $m \geq p$; therefore the size of a VR (or WVR) instance is always $\Theta(m)$. G will always be a (simple) graph in our lower bound proofs. So our lower bounds apply to graphs, while our upper bounds apply to multigraphs as well.

We will only consider optimization problems for which every feasible solution has a non-negative objective function value. Throughout the paper, opt_P will denote the optimal solution value to an optimization problem P. When no confusion arises, we will simply denote it by opt. Given a feasible solution y to an instance x of P, $val(x, y)$ denotes the value that the objective function of P takes on the feasible solution y. Consider a minimization (resp., maximization) problem. Given a parameter $\rho \geq 1$, we say that an algorithm is a *ρ-approximation algorithm* if the value of the solution it returns is at most $\rho \cdot opt$ (resp., is at least opt/ρ). Similarly, given a parameter k and a function $f(\cdot)$, an $O(f(k))$-approximation algorithm is an algorithm that returns a solution whose value is $O(f(k) \cdot opt)$ (resp., $O(opt/f(k))$). We say that this heuristic approximates the problem *within (a factor)* $O(f(k))$.

A problem F is *APX-hard* if there exists some constant $\sigma > 1$ such that F cannot be σ-approximated in polynomial time unless $\mathcal{P} = \mathcal{NP}$. Occasionally we will establish hardness under different complexity assumptions such as $\mathcal{NP} \not\subseteq \mathcal{QP}$, $\mathcal{NP} \not\subseteq \mathcal{ZPP}$, etc.

Given two optimization problems A and B, an *L-reduction* from A to B consists of a pair of polynomial-time computable functions (f, g) such that, for two fixed constants α and β: (a) f maps input instances of A into input instances of B; (b) given an A-instance a, the corresponding B-instance $f(a)$, and any feasible solution b for $f(a)$, $g(a, b)$ is a feasible solution for the A-instance a; (c) $|opt_B(f(a))| \leq \alpha |opt_A(a)|$ for all a, and (d) $|opt_A(a) - val(a, g(a, b))| \leq \beta |opt_B(f(a)) - val(f(a), b)|$ for each a and for every feasible solution b for $f(a)$.

¿From this definition it follows that the relative errors are linearly related, i.e.

$$\frac{|opt_A(a) - val(a, g(a, b))|}{opt_A(a)} \leq \alpha\, \beta\, \frac{|opt_B(f(a)) - val(f(a), b)|}{opt_B(f(a))}.$$

(Thus, the "L" in "L-reduction" stands for "linear".) ¿From this the following well-known fact follows; it will be used repeatedly in the paper.

Fact 1.1 *Given optimization problems A and B and an L-reduction from A to B with parameters α and β, the existence of a polynomial-time $(1+\varepsilon)$-approximation algorithm for B implies the existence of the following types of polynomial-time approximation algorithms for A:*

(i) an $(1 + \alpha\beta\varepsilon)$-approximation algorithm, if both A and B are minimization problems;

(ii) *an* $(1-\alpha\beta\varepsilon)^{-1}$*-approximation algorithm, if A is a maximization problem and B is a minimization problem and if* $\alpha\beta\varepsilon < 1$*;*

(iii) *an* $(1-\frac{\alpha\beta\varepsilon}{1+\varepsilon})^{-1}$*-approximation algorithm, if both A and B are maximization problems and if* $\alpha\beta\varepsilon < 1+\varepsilon$*.*

Notice also that L-reductions map optimal solution into optimal solutions.

2 Hardness of Approximation

Recall that ℓ stands for the maximum length of any laid-out path of the VR-instance.

2.1 Hardness of VR when ℓ is bounded

In this subsection we show that the problem is APX-hard even for the special case in which ℓ equals 3. We will show in Section 3 that the problem is polynomially solvable for $\ell = 1, 2$ and that there is a polynomial time $\lceil \ell/2 \rceil$-approximation algorithm. Our reduction is from the MAX 3-SATISFIABILITY (MAX 3-SAT) problem in which one is given a collection of CNF clauses with exactly three literals per clause, and wants to find a truth assignment to the variables so as to maximize the number of clauses satisfied. To get a reasonable constant we make use of a major result of Håstad [11] which shows that if we can approximate MAX 3-SAT in polynomial time to within $\frac{8}{7} - \varepsilon$ with $\varepsilon > 0$ being any fixed constant, then $\mathcal{NP} = \mathcal{P}$.

Notation. Given a MAX 3-SAT instance, we let p_x and q_x denote the number of positive and negative occurrences of a variable x in it, respectively. For any constant $\lambda \geq 1$, let $MS(\lambda)$ denote the family of MAX 3-SAT instances in which $p_x \leq \lambda q_x$ and $q_x \leq \lambda p_x$ for all variables x.

Theorem 2.1 *For any constant* $\lambda \geq 1$*, there is an L-reduction from* $MS(\lambda)$ *to VR (wherein* $\ell = 3$*), with* $\alpha = \frac{25\lambda+1}{7(\lambda+1)}$ *and* $\beta = 1$*.*

Proof. Let $x_1, \ldots, x_n$ and $C_1, \ldots, C_m$ be the variables and clauses of the given $MS(\lambda)$ instance. To each variable x we associate a *variable gadget* depicted in Figure 1; two parallel paths going from the entry vertex $\text{ENTRY}(x)$ to the exit vertex $\text{EXIT}(x)$. The paths have $M_x := \max\{p_x, q_x\}$ many arcs each. The i-th arc of the upper path corresponds to the i-th positive occurrence of x and has label x^i. The i-th arc of the lower path corresponds to the i-th negative occurrence of x and has label $\bar{x}^i$. Since the number of positive and negative occurrences may differ, some arcs may have no label. To every clause C we associate a *clause gadget* like the one depicted in Figure 2; there are three parallel arcs going from the entry point $\text{ENTRY}(C)$ to the exit point $\text{EXIT}(C)$. If the occurrence of x in C is its i-th positive occurrence, the label of the corresponding arc in the clause gadget is x^i. If the occurrence of x in C is its i-th negative occurrence, the label is $\bar{x}^i$.

These gadgets are strung together as follows. The vertex $\text{EXIT}(x_i)$ is identified with $\text{ENTRY}(x_{i+1})$, for $i = 1, \ldots, n-1$. The vertex $\text{EXIT}(x_n)$ is identified with $\text{ENTRY}(C_1)$,

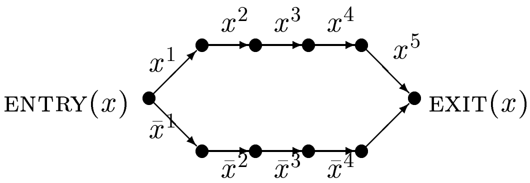

Figure 1: The gadget for a variable x with $p_x = 5$ and and $q_x = 4$.

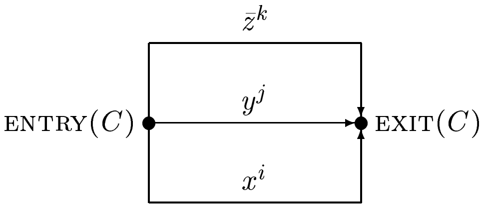

Figure 2: The gadget associated with the clause $C = (x \vee y \vee \bar{z})$, where x's occurrence is the i-th positive occurrence, y's occurrence is the j-th positive occurrence, and z's occurrence is the k-th negative occurrence.

and EXIT(C_i) is identified with ENTRY(C_{i+1}), for $i = 1, \ldots, m-1$. The source vertex s coincides with ENTRY(x_1) while the sink vertex t coincides with EXIT(C_m).

The laid-out paths can have length either 1 or 3. For each positive occurrence of a variable x, say the i-th, we form a path $P(x,i)$ of length 3. Let C be the clause containing the i-th positive occurrence of x. The first arc of $P(x,i)$ is the arc with label x^i of the gadget of the clause C; the second arc is a new arc $f_{x,i}$, joining the head of the first arc to the tail of the third arc; the third arc is the arc with label x^i of the gadget corresponding to the variable x.

Analogously, we have a path $N(x,j)$ for each negative occurrence of x. The first arc comes from the clause gadget and has label $\bar{x}^j$, the second is a new arc $g_{x,j}$, and the third comes from the variable gadget and has label $\bar{x}^j$.

Arcs of the variable gadgets without label form special laid-out paths of length 1.

The overall structure of the construction is therefore that of a layered directed acyclic graph with some additional backward arcs (the $f_{x,i}$'s and $g_{x,j}$'s) going from clause gadgets back to variable gadgets. The reduction can obviously be carried out in polynomial time. To prove that it is an L-reduction we establish some preliminary facts. The first observation is that, as mentioned before, we can assume without loss of generality that any solution to the VR instance is a simple path from s to t (i.e., no loops). In particular, for each variable gadget we either take the upper path, corresponding to the positive occurrences of the variable, or the lower path. For each clause gadget we traverse exactly one arc, corresponding to one variable occurrence. Letting $M := \sum_{i=1}^{n} M_{x_i}$, we now see that any path from s to t has cost of the form

$M + m - k$, where m is the number of clauses of the boolean formula, and k is a function of the path that will be determined next. A cost of M is incurred because, in order to traverse every variable gadget, we must use either the upper or the lower path. To traverse the clause gadgets we will pay a cost of 1 for every clause, unless we can take an arc belonging to a path we have already paid for when traversing the variable gadget. Therefore the reduction establishes a bijection between feasible solutions of the $MS(\lambda)$ and VR instances specified as follows. To an assignment that satisfies clauses $C_1, \ldots, C_k$ there corresponds a path P from s to t that uses only $M + m - k$ laid-out paths, defined as follows: For each variable gadget, take the upper part if the variable is set to True, and the lower part otherwise. For each clause, take an arc of a path you have already paid for, if it exists (i.e., if the clause is satisfied), or an arbitrary arc otherwise (i.e., if the clause is not satisfied).

Conversely, a solution P to VR of cost $M + m - k$ can be mapped in polynomial time into a satisfying assignment satisfying k clauses: A variable is set to True if P traverses the upper portion of a variable gadget, and to False otherwise.

Thus, the two optimal solution values, opt_{VR} for the VR instance defined and opt_{SAT} for the original $MS(\lambda)$ instance, are related as follows:

$$opt_{VR} = M + m - opt_{SAT},$$

and the reduction is an L-reduction with $\beta = 1$. To find the value of α, consider any variable x. Since $p_x \leq \lambda q_x$ and $q_x \leq \lambda p_x$ by the definition of $MS(\lambda)$, we can verify that $M_x \leq \frac{\lambda}{\lambda+1}(p_x + q_x)$. So, $M = \sum_{i=1}^{n} M_{x_i} \leq \frac{\lambda}{\lambda+1} \cdot \sum_{i=1}^{n}(p_x + q_x) = \frac{3\lambda}{\lambda+1} \cdot m$. Then, recalling that $opt_{SAT} \geq \frac{7}{8}m$, we obtain the bound

$$opt_{VR} = M + m - opt_{SAT} \leq \left(\frac{3\lambda}{\lambda+1}\right) m + m - opt_{SAT} \leq \left(\frac{25\lambda+1}{7(\lambda+1)}\right) opt_{SAT},$$

i.e., $\alpha = \frac{25\lambda+1}{7(\lambda+1)}$. □

Corollary 2.1 *The existence of a polynomial-time* $(183/176 - \varepsilon)$*-approximation algorithm for VR, even in the case* $\ell = 3$ *and for any fixed* $\varepsilon > 0$*, implies* $\mathcal{NP} = \mathcal{P}$.

Proof. Suppose indeed that for some constant $\varepsilon < 7/176$, there is a polynomial-time $(183/176 - \varepsilon)$-approximation algorithm for VR, in the case $\ell = 3$. Assuming this, we present a polynomial-time approximation algorithm for MAX 3-SAT.

Suppose we are given a MAX 3-SAT instance. We will first run a preprocessing algorithm that repeatedly: (i) sets certain variables x (appropriately to True or False), and (ii) deletes all clauses that contain the variable x. Let δ be a positive constant that will be specified below; let p_x and q_x respectively denote the number of positive and negative occurrences of a variable x in the *current* (shrunk) formula. The preprocessing is as follows. If there is a variable x with $p_x > (7 + \delta)q_x$, then we set x to True and delete all clauses that contain x; if there is a variable x with $q_x > (7 + \delta)p_x$, we set x to False and delete all clauses that contain x. We repeat this process until we have $p_x \leq (7+\delta)q_x$ and $q_x \leq (7+\delta)p_x$ for all remaining variables x in the remaining formula

F'. We then run the L-reduction of Theorem 2.1 on F' with $\lambda = 7 + \delta$, and apply the claimed $(183/176 - \varepsilon)$-approximation algorithm on the resulting VR instance to get an approximately optimal solution for the remaining formula F'.

Let us analyze the above algorithm. Given an original formula F, we produce a shrunk formula F'; let F'' denote the formula induced by the clauses deleted in the preprocessing. Let *opt*, *opt'* and *opt''* denote the optimal objective function values for the MAX 3-SAT instances F, F' and F'' respectively. Since the set of clauses in F is the disjoint union of those of F' and F'', we have

$$opt \leq opt' + opt''.$$

It is easy to see that our preprocessing algorithm satisfies at least an $a_1 := \frac{7+\delta}{8+\delta}$ fraction of the clauses of F''. Also, Theorem 2.1 and Fact 1.1(ii) show that we satisfy at least an

$$a_2 := \left(1 - \frac{25(7+\delta)+1}{7(7+\delta+1)} \cdot \varepsilon\right)$$

fraction of *opt'*, for the formula F'.

Thus, we satisfy a total of at least

$$a_1 opt'' + a_2 opt' \geq \min\{a_1, a_2\} \cdot (opt' + opt'') \geq \min\{a_1, a_2\} \cdot opt \tag{1}$$

clauses. Setting $\delta = \frac{7-176\varepsilon}{25\varepsilon}$ to equate a_1 and a_2, we get that $\min\{a_1, a_2\} = (8/7 - \delta')^{-1}$, where $\delta' = \delta'(\varepsilon)$ is some positive constant. Thus, (1) shows that we have an $(8/7 - \delta')$-approximation algorithm for MAX 3-SAT, which would imply that $\mathcal{NP} = \mathcal{P}$ [11]. $\square$

2.2 A stronger negative result when ℓ is unbounded

In this subsection we exhibit an L-reduction from the Symmetric Label Cover (SLC) problem to VR. The reduction implies that, unless $\mathcal{NP} \subseteq \mathcal{QP}$, VR is hard to approximate within $O(2^{\log^{\frac{1}{2}-\varepsilon} m})$ for any fixed ε.

The problem SLC is the following. The input of the problem is a bipartite graph $H = (L, R, E)$ and two sets of labels: A, to be used on the left hand side L only, and B, to be used on the right hand side R only. Each edge e has a list P_e of admissible pairs of the form (a, b) where $a \in A$ and $b \in B$. A feasible solution is a label assignment $A_u \subseteq A$ and $B_v \subseteq B$ for every vertex $u \in L$ and $v \in R$ such that, for every edge $e = uv$, there is at least one admissible pair $(a, b) \in P_e$ with $a \in A_u$ and $b \in B_v$. The objective is to find a feasible label assignment which minimizes $\sum_{u \in L} |A_u| + \sum_{v \in R} |B_v|$. Let $q := \sum_{e \in E} |P_e|$ and note that the size of an SLC instance is equal to $\Theta(q)$.

The existence of a polynomial-time $O(2^{\log^{\frac{1}{2}-\varepsilon} q})$-approximation algorithm for SLC, for any $\varepsilon > 0$, implies that every problem in $\mathcal{NP}$ can be solved by a quasi-polynomial-time algorithm, i.e., $\mathcal{NP} \subseteq \mathcal{QP}$ [13, 6]. Here, we exhibit an L-reduction from SLC to VR with $\alpha = \beta = 1$.

Theorem 2.2 *There is an L-reduction from SLC to VR with $\alpha = \beta = 1$.*

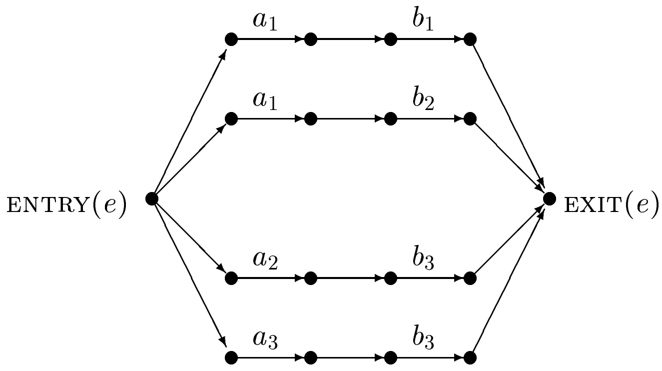

Figure 3: An edge gadget for an edge e whose admissible pairs are (a_1, b_1), (a_1, b_2), (a_2, b_3) and (a_3, b_3).

Proof. For notational convenience, let $A = \{a_1, a_2, \ldots, a_{|A|}\}$ and $B = \{b_1, b_2, \ldots, b_{|B|}\}$. To each edge e of the SLC instance $H = (L, R, E)$, we associate an *edge gadget* shown in Figure 3. The gadget consists of many parallel paths, each formed by 5 arcs, from the entry point ENTRY(e) to the exit point EXIT(e). There is one path for every pair in the list P_e. On the path corresponding to the pair (a, b), the second arc on the path is labeled with a, while the fourth is labeled with b (the other arcs in the paths have no label). These edge gadgets are strung together as follows. Let $e_1, \ldots, e_m$ be the edges of H. For $i = 1, \ldots, m-1$, the exit point EXIT(e_i) is identified with the entry point ENTRY(e_{i+1}). The source vertex s coincides with ENTRY(e_1) while the sink vertex t coincides with EXIT(e_m). Let the resulting graph be denoted by G. The VR instance will be obtained by adding arcs to G, as explained below. Notice that G is a planar, layered, acyclic graph that can be easily laid out on a piece of paper by drawing the edge gadgets from left to right. We use this to define a total ordering between the arcs of G, as follows. An edge a is smaller than an edge b if a belongs to a layer to the left of that of b or, in case they belong to the same layer, if a is above b.

The laid-out paths are constructed as follows. For each vertex $u \in L$ and label $a \in A$, there is a laid-out path P_{ua}. We define the set S of arcs of G that will be contained in P_{ua} as follows. Consider all edges incident with u which have admissible pairs with a as a first coordinate. For each such edge, let S contain the arcs labeled a in the corresponding gadget. Let $w_1, w_2, \ldots w_k$ be these edges ordered according to the total ordering described above. We introduce "backward" edges f_i joining the head of w_i to the tail of w_{i-1}, for $2 \leq i \leq k$. This defines the laid-out path $P_{ua} = w_k f_k w_{k-1} \ldots f_2 w_1$.

Analogously, define "backward" paths of the form $P_{vb} = x_k g_k x_{k-1} \ldots g_2 x_1$, where the x_i's are the arcs with label $b \in B$ belonging to gadgets of edges incident with $v \in R$.

This reduction establishes a bijection between feasible solutions which preserves the costs. For the proof we need a couple of observations. The first is that it suffices to restrict oneself to feasible solutions of SLC which are minimal. That is, the solution can be constructed by processing each edge once. When an edge $e = uv$ is processed a label a is added to the partial solution A_u so far, and a label b is added to the partial solution B_v computed so far, only if the current set $B_u \times B_v$ does not contain an admissible pair for e.

A second observation is that without loss of generality we can consider only solutions to the VR instance of the following form. To generate the path, we start walking from the source towards the sink. When we enter each arc gadget we choose one of the parallel paths reaching the exit of the gadget. We never use a "backward" arc f_i or g_j. We stop when we reach the sink. We call these the *canonical* solutions.

The reduction establishes a cost-preserving reduction between minimal label covers and canonical solutions to VR. Furthermore, given one, it is possible to obtain the other in polynomial time. This is because a minimal solution is nothing else than a way to prescribe, for each arc e, which label pair to choose. This is exactly what a canonical solution does. □

Noting that, in the L-reduction of Theorem 2.2, the number of arcs of the VR instance defined is $\Theta(\sum_{e \in E} |P_e|)$, we get the following corollary.

Corollary 2.2 *The existence of a polynomial-time $O(2^{\log^{\frac{1}{2}-\varepsilon} m})$-approximation algorithm for VR, for any fixed $\varepsilon > 0$, implies $\mathcal{NP} \subseteq QP$.*

2.3 The hardness of approximating Gondolier Avoidance

Recall that GA, the complementary problem to VR, asks for the maximum number of laid-out paths which can be removed without disconnecting t from s. We start by showing a very simple L-reduction from Set Cover (SC) to VR. Recall that an instance of SC is given by a ground set $E = \{e_1, \dots, e_f\}$ and a family $\mathcal{S} = \{I_1, \dots, I_q\}$ of subsets of E. The objective is to select a minimum-cardinality collection of subsets in $\mathcal{S}$ such that each element in the ground set is contained in at least one subset of the collection.

Theorem 2.3 *There is an L-reduction from SC to VR with $\alpha = \beta = 1$.*

Proof. Given an SC instance, define the following VR instance. For $i = 1, \dots, f$, let $\mathcal{S}_i = \{I_{j_1}, \dots, I_{j_s}\}$ be the family of sets in $\mathcal{S}$ containing element e_i. To each element e_i we associate a gadget formed by $|\mathcal{S}_i|$ parallel arcs going from $\text{ENTRY}(e_i)$ to $\text{EXIT}(e_i)$, the h-th of these arcs being associated with subset I_{j_h}. The gadgets are connected by identifying $\text{EXIT}(e_i)$ with $\text{ENTRY}(e_{i+1})$ for $i = 1, \dots, f-1$. Moreover, the source vertex s coincides with $\text{ENTRY}(e_1)$ while the sink vertex t coincides with $\text{EXIT}(e_f)$.

There is a laid-out path associated with each subset I_j, defined as follows. Consider subset $I_j = \{e_{i_1}, e_{i_2}, \dots, e_{i_{r-1}}, e_{i_r}\}$, with $r = |I_j|$, assuming $i_1 < i_2 < \dots < i_{r-1} < i_r$. The laid-out path associated with I_j has the form $P_j = a_r f_r a_{r-1} \dots f_2 a_1$, where, for $k = 1, \dots, r$, a_k is the arc associated with I_j in the gadget corresponding to element e_{i_k}, while f_k $(2 \le k \le r)$ is an appropriate "backward" arc from the head of a_k to the tail of a_{k-1}. In other words, P_j contains the arcs associated with subset I_j in the gadgets of elements $e_i \in I_j$, and these arcs appear in the path according to decreasing index of elements.

As in the L-reductions presented before, it is easy to see that any simple path from s to t intersecting k laid-out paths corresponds to an SC solution of value k and vice versa. □

The Vertex Cover (VC) problem is the special case of SC where each element of the ground set is contained in exactly two subsets. The problem therefore can be represented by an undirected graph G where vertices correspond to subsets and edges denote nonempty intersections between pairs of subsets. If we complement the objective function of VC, we have the well-known Independent Set (IS) problem. Hence, Theorem 2.3 yields also an L-reduction from IS to GA with $\alpha = \beta = 1$. Recalling the most up-to-date inapproximability results for IS [10], and noting that the number of laid-out paths in the GA instances defined by Theorem 2.3 is equal to the number of vertices in the IS instance, yields the following corollary.

Corollary 2.3 *Let p denote the number of laid-out paths of a GA instance. The existence of a polynomial-time $O(p^{1-\varepsilon})$-approximation algorithm for GA, for any $\varepsilon > 0$, implies $\mathcal{NP} \subseteq \mathcal{ZPP}$. The existence of a polynomial-time $O(p^{1/2-\varepsilon})$-approximation algorithm for GA, for any fixed $\varepsilon > 0$, implies $\mathcal{NP} = \mathcal{P}$.*

A more careful evaluation of the reduction above also shows the following.

Corollary 2.4 *Let m denote the number of arcs of a GA instance. The existence of a polynomial-time $O(\sqrt{m^{1-\varepsilon}})$-approximation algorithm for GA, for any $\varepsilon > 0$, implies $\mathcal{NP} \subseteq \mathcal{ZPP}$. The existence of a polynomial-time $O(\sqrt{m^{1/2-\varepsilon}})$-approximation algorithm for GA, for any fixed $\varepsilon > 0$, implies $\mathcal{NP} = \mathcal{P}$.*

Proof. If $G = (V, E)$ is the original IS instance, then $f = |E|$ and $q = |V|$. For the GA instance defined by the reduction, $m = \Theta(f)$, whereas $p = q$ as noted above. Since we have $|E| = O(|V|^2)$, we have $m = O(p^2)$, i.e., $p = \Omega(\sqrt{m})$ and the claim follows from Corollary 2.3. □

3 Approximation Algorithms

Having shown that even some simple cases of the problem are computationally difficult, we turn to presenting approximation algorithms. Our algorithm are variations of the following idea. We divide the laid-out paths into "long" and "short". To find an st-path we take all "long" paths, for there cannot be too many of these. Then we put a weight of 1 on every arc belonging to each "short" path, a weight of 0 on all other arcs and run a shortest path algorithm (henceforth SPA). This weighting scheme overestimates the cost of using "short" paths and hence they will be used parsimoniously.

In many cases, we will try all the possible values of *opt*. Note that, for VR, we know that $opt \leq n-1$, as the solution may be assumed to be a simple path, and $opt \leq p$, as in the worst case all the laid out paths are intersected by the optimal solution. (In short, we can assume w.l.o.g. that the value of *opt* be known.) The term $\mu := \min\{n-1, p\}$ will often appear in the time complexity of the algorithms we present. We will use a weight function defined as follows. Given a nonnegative integer x, for each arc a of the graph, let

$$z_x(a) := \begin{cases} 1 & \text{if } a \text{ belongs to some laid-out path of length} \le x, \\ 0 & \text{otherwise.} \end{cases}$$

The next fact is obvious but we record it for future reference:

Lemma 3.1 *The number of paths of length $\le x$ intersected by the SPA solution with weights z_x is at most $opt \cdot x$.*

3.1 Constant factor approximation for fixed ℓ

Recall that the maximum length of a laid-out path is denoted by ℓ. We now present an $\lceil \ell/2 \rceil$–approximation, which is especially useful in the common case where ℓ is "small" (in practice, lightpaths will typically only consist of a few arcs).

The first algorithm we present, called A_0, initially assigns weight 1 to each arc that lies in some laid-out path, and weight 0 to all other arcs. Then, for each laid-out path of length at least 2 and consecutive arcs $(u, v), (v, t)$ in the path, A_0 adds a new arc (u, t) with weight 1. Finally, A_0 applies SPA to the graph obtained. At the end, possible new arcs are replaced by the corresponding pair of "real" consecutive arcs. Also, since all arc-weights lie in $\{0, 1\}$, SPA only takes $O(m)$ time here.

The following fact follows from Lemma 3.1, since the new "shortcut" arcs (such as (u, t)) that we add above essentially "reduce" ℓ to $\lceil \ell/2 \rceil$:

Fact 3.1 *Algorithm A_0 returns a solution of value at most $opt \cdot \lceil \ell/2 \rceil$ in $O(m)$ time.*

The following corollaries are immediate consequences of the above fact.

Corollary 3.1 *Algorithm A_0 is an $O(m)$-time $\lceil \ell/2 \rceil$-approximation algorithm for VR.*

Corollary 3.2 *Algorithm A_0 is an $O(m)$-time exact algorithm for VR when $\ell \le 2$.*

It is also easy to see that the above approach yields an $O(SSSP(n, m))$-time $\lceil \ell/2 \rceil$-approximation algorithm for PVR. For GVR, we use the same algorithm, except that we do not add any "shortcut arcs" such as (u, t) above. Lemma 3.1 shows that this is an $O(m)$-time ℓ-approximation algorithm for GVR.

3.2 An approximation algorithm for unbounded ℓ

We next show an approximation algorithm for unbounded ℓ, called A_1. The approximation guarantee of this algorithm is $O(\sqrt{m/opt})$; we then show how this factor of *opt* in the denominator of the approximation bound, can be used to show that VR (and hence SLC) are different from the Chromatic Number problem, in a certain quantitative sense related to approximability.

Let $x := \sqrt{m/opt}$. As *opt* is unknown, algorithm A_1 tries all the $O(\mu)$ possible values *opt*, considering the associated x value. Given the graph G and any nonnegative integer x, let

$$f(G, x) := (\#\text{ of laid-out paths of length} > x).$$

For each value of x, A_1 assigns weight $z_x(a)$ to each arc a of the graph, and applies SPA to the corresponding instance. On output, A_1 gives the best solution found by the SPA calls for the different values of x.

Lemma 3.2 *For any nonnegative integer x: (a) $f(G,x) \leq m/(x+1)$; (b) the solution obtained by applying SPA after having assigned weight $z_x(a)$ to each arc a of the graph has value at most $opt \cdot x + f(G,x)$.*

Proof. Since all laid-out paths are arc-disjoint, the number of paths of length more than x is at most $m/(x+1)$, yielding (a). Moreover, for each x, we know by Lemma 3.1 that the number of paths of length $\leq x$ intersected by the SPA solution with weights z_x is at most $opt \cdot x$. So the total number of paths intersected is at most $opt \cdot x + f(G,x)$, showing (b). □

Theorem 3.1 *(a) Algorithm A_1 runs in $O(\mu m)$ time, and returns a solution of value $O(\sqrt{opt \cdot m})$. (b) A solution of value $O(\sqrt{opt \cdot m})$, with a slightly worse constant than in the bound of (a), can also be computed in time $O(m \log \mu)$.*

Proof. By Lemma 3.2 (a) and (b), the total number of paths intersected is at most $m/(x+1) + opt \cdot x$. This quantity is minimized when x is approximately $\lceil \sqrt{m/opt} \rceil$, and the minimum value is $O(\sqrt{opt \cdot m})$. Since we run SPA for each possible value of opt, the running time follows, for part (a). Part (b) follows by guessing $opt = 1, 2, 4, \ldots$. □

Corollary 3.3 *Algorithm A_1 is an $O(\sqrt{m/opt})$-approximation algorithm for VR.*

The above approach yields an $O(\sqrt{m/opt})$-approximation algorithm for GVR as well.

Notation (q-exhaustive search). For a given VR instance, suppose we know that opt is at most some value q. Then, the following simple algorithm to find an optimal solution, called *q-exhaustive search*, will be useful in a few contexts from now on: Enumerate all subsets X of the set of laid-out paths such that $|X| \leq q$, and check if the laid-out paths in X alone are sufficient to connect s to t. The time complexity of this approach is

$$O\left(m \cdot \sum_{i=0}^{q} \binom{p}{i}\right) \leq O\left(m \cdot \sum_{i=0}^{q} \binom{m}{i}\right).$$

In particular, if q is a constant, then we have a polynomial-time algorithm.

Some separation results. We now present two "separation" results that indicate that VR is likely to be in an approximation class different from that to which Chromatic Number (CN) belongs. First, we show that, unless $\mathcal{P} = \mathcal{NP}$, there can be no L-reduction from CN to VR. Second, we use Theorem 3.1 to show that VR (and hence also SLC, in view of Theorem 2.2) has certain subexponential-time algorithms which are unlikely to exist for CN (and also for other problems such as Independent Set).

Proposition 3.1 *The existence of an L-reduction from CN to VR implies $\mathcal{NP} = \mathcal{P}$.*

Proof. Let us consider a subclass $\mathcal{S}$ of CN-instances that have this property: a graph G in $\mathcal{S}$ is always 4-colorable and maybe 3-colorable, but it is $\mathcal{NP}$-complete to tell the two cases apart. (Such classes of instances arise naturally from $\mathcal{NP}$-hardness proofs. An example is those graphs obtained via the polynomial-time reduction from 3-SAT to CN[9, 19]. Another example is given by the construction of Holyer showing that that edge-coloring cubic graphs is $\mathcal{NP}$-hard [9, 12].)

Let us assume there is an L-reduction from CN to VR. We shall use it to solve CN on $\mathcal{S}$ in polynomial time. The L-reduction obviously works for $\mathcal{S}$ too. For the VR instances obtained via this L-reduction on CN instances in $\mathcal{S}$, we have

$$opt_{VR} \leq \alpha \cdot opt_{CN} \leq 4\alpha$$

for some constant α that depends on the L-reduction. So, to find an optimal solution for these VR instances, it suffices to do a polynomial-time (4α)-exhaustive search. Once the VR optimum is obtained, it is mapped back to the optimum of the CN instance via the L-reduction (L-reductions map optimal solutions into optimal solutions). $\square$

We now switch to the second result of this subsection. Let N denote the size of an instance of SAT, and s denote the number of vertices in a CN instance. Note that the size S of a CN instance is at most $O(s^2)$. Given any fixed $\varepsilon > 0$ and any oracle that approximates CN to within $O(s^{1-\varepsilon})$, work of [8] presents a randomized polynomial-time algorithm that uses this oracle to correctly solve SAT instances with a probability of at least $2/3$. Thus, if a positive constant δ is such that any randomized algorithm for SAT must have running time $\Omega(2^{N^\delta})$, then there is a constant $\delta' > 0$ such that for any constant $\varepsilon > 0$, there is no $O(2^{S^{\delta'}})$-time $O(\sqrt{S^{1-\varepsilon}})$-approximation algorithm for CN. We now show that the situation is different for VR.

Theorem 3.2 *For any fixed $\varepsilon > 0$, there is a $2^{O(m^\varepsilon \log m)}$-time $O(\sqrt{m^{1-\varepsilon}})$-approximation algorithm for VR.*

Proof. As before, we assume we know *opt*, in the sense that we are allowed to try the procedure below for all the $O(\mu)$ possible values of *opt*. Suppose a positive constant ε is given. Theorem 3.1 shows that in polynomial time, we can find a VR solution of value $O(\sqrt{opt \cdot m}) = O(opt \cdot \sqrt{m/opt})$. Thus, if $opt \geq m^\varepsilon$, we can approximate the problem to within a factor $O(\sqrt{m/opt}) \leq O(\sqrt{m^{1-\varepsilon}})$ in polynomial time. Accordingly, we assume from now on that $opt \leq m^\varepsilon$. In this case, we can do an m^ε-exhaustive search to find an optimal solution, with running time $O(m \cdot \sum_{i=0}^{m^\varepsilon} \binom{m}{i})$, which is $m \cdot m^{O(m^\varepsilon)} = 2^{O(m^\varepsilon \log m)}$. $\square$

3.3 Randomized sparsification: Improvements when *opt* is small

We next show how to improve on the $O(\sqrt{opt \cdot m})$ solution value for VR in case *opt* is "small", i.e., $O(\log n)$. In particular, for any constant $C > 0$, we present a randomized polynomial-time algorithm A_2 that with high probability computes a solution of value at most $O\left(\sqrt{\frac{opt\cdot m}{n^{C/opt}}}\right)$. For instance, if $opt = \sqrt{\log n}$, A_2 computes a solution of value $O\left(\sqrt{\frac{opt\cdot m}{2^{C\sqrt{\log n}}}}\right)$. The difference between A_1 and A_2 is a randomized "sparsification" technique. Note that, due to Corollary 3.3, the following discussion is interesting only if $opt \leq \log n$.

Suppose that for a given constant $C > 0$, we aim to find a solution of value $O\left(\sqrt{\frac{opt\cdot m}{n^{C/opt}}}\right)$. We will assume throughout that n is sufficiently large as a function of C, i.e., $n \geq f(C)$ for some appropriate function $f(\cdot)$. Indeed, if $n < f(C)$, then $opt \leq \log n$ is bounded by a constant, and we can find an optimal solution by a polynomial-time $(\log n)$-exhaustive search.

As in algorithm A_1, algorithm A_2 has to try all the possible $O(\mu)$ values of *opt*, computing a solution for each, and returning on output the best one found. In the following, we will consider the iteration with the correct *opt* value. Let $\lambda := n^{C/opt}$ and $x := \lfloor\sqrt{m/(opt\cdot\lambda)}\rfloor$, where C is the given constant.

If $opt \leq 2C$, algorithm A_2 finds an optimal solution by a $(2C)$-exhaustive search. Hence, in the following we describe the behavior of A_2 for $opt > 2C$. By the definition of x and the fact that $m \geq n$, we have

$$\frac{m}{\lambda x} \geq \sqrt{\frac{opt\cdot m}{\lambda}} \geq n^{1/4}. \tag{2}$$

The *deletion* of a laid-out path corresponds to the removal of all the associated arcs from G. For each possible value of *opt*, algorithm A_2 uses the following "sparsification" procedure: Each laid-out path is independently deleted with probability $(1 - 1/\lambda)$. Then, SPA is applied on the resulting graph, after having assigned weight $z_x(a)$ to each arc a that was not removed. The sparsification procedure and the associated shortest path computation are repeated $O(n^C)$ times, and the best solution found is output.

Theorem 3.3 *Given any constant $C > 0$, Algorithm A_2 runs in polynomial time and returns a VR solution of value $O\left(\sqrt{\frac{opt\cdot m}{n^{C/opt}}}\right)$ with high probability.*

Proof. Let P be an optimal VR solution. We have that

$$\Pr[\text{all laid-out paths intersected by } P \text{ survive}] = (1/\lambda)^{opt} = n^{-C}. \tag{3}$$

Moreover, let Z be the number of laid-out paths of length more than x that remain undeleted. By Lemma 3.2 (a) and linearity of expectation, $E[Z] \leq m/(\lambda x)$. Using a simple Chernoff bound [18], we get

$$\Pr\left[Z \geq \frac{2m}{\lambda x}\right] \leq (e/4)^{m/(\lambda x)} \leq (e/4)^{n^{1/4}}, \tag{4}$$

by (2), where e denotes the base of the natural logarithm as usual.

As mentioned earlier in this subsection, we may assume that n is large enough: We will assume that $n^{-C} - (e/4)^{n^{1/4}} \geq 1/(2n^C)$. Thus, with probability at least $n^{-C} - (e/4)^{n^{1/4}} \geq 1/(2n^C)$, we have that (i) all laid-out paths intersected by P survive, and (ii) $Z \leq 2m/(\lambda x)$. Thus, sparsification ensures that $f(G,x)$ is at most $2m/(\lambda x)$ after path deletion. By Lemma 3.2, running SPA yields a solution of value at most $opt \cdot x + \frac{2m}{\lambda x} = O\left(\sqrt{\frac{opt \cdot m}{n^{C/opt}}}\right)$. The above success probability of $1/(2n^C)$ can be boosted to, say, 0.9 by repeating the procedure $O(n^C)$ times. The time complexity follows immediately from the algorithm description. $\square$

Corollary 3.4 *Algorithm A_2 (with $C = 1$, say) is a randomized $O(\sqrt{m/\log n})$-approximation algorithm for VR. (In other words, Algorithm A_2 always runs in polynomial time, and delivers a solution that is at most $O(\sqrt{m/\log n})$ times the optimal solution with high probability.)*

Proof. By Theorem 3.3, A_2 with $C = 1$, computes a solution of value $opt \cdot O(\sqrt{m/(opt \cdot n^{1/opt})})$. Now, elementary calculus shows that $opt \cdot n^{1/opt}$ is minimized when $opt = \Theta(\log n)$. Thus, $opt \cdot O\left(\sqrt{\frac{m}{opt \cdot n^{1/opt}}}\right)$ is at most $opt \cdot O(\sqrt{m/\log n})$. $\square$

In the next subsection we show that Algorithm A_2 can be derandomized.

3.4 Derandomizing Algorithm A_2

Our aim is to prove the following.

Theorem 3.4 *For any given constant $C > 0$, there is a* deterministic *polynomial-time approximation algorithm for VR that returns a solution of value $O\left(\sqrt{\frac{opt \cdot m}{n^{C/opt}}}\right)$.*

The proof will go through several lemmas and observations. The basic starting point is that randomized algorithms are often robust, i.e., they remain correct even if some small changes are made to the underlying distribution. This opens up a well-known approach to derandomization: We first construct a *small* sample space that approximates a given (larger) sample space sufficiently well. We may then conduct an exhaustive search in the small sample space to find a sample point that is sufficiently good for a randomized algorithm in consideration.

More precisely, we will need the following useful result of [7]. Suppose $Y_1, Y_2, \ldots, Y_N \in \{0, 1, \ldots, a-1\}$ are *independent* random variables with arbitrary individual distributions and joint distribution D. We define a finite set $S \subseteq \{0, 1, \ldots, a-1\}^N$ to be a (d, ε)-approximation for D if, for $(\hat{Y}_1, \ldots, \hat{Y}_N)$ sampled uniformly at random from S, we have for all $I \subseteq \{1, 2, \ldots, N\}$ with $|I| \leq d$ and for all $\gamma_1, \gamma_2, \ldots, \gamma_{|I|} \in \{0, 1, \ldots, a-1\}$ that

$$|\Pr[\bigwedge_{i \in I} (\hat{Y}_i = \gamma_i)] - \prod_{i \in I} \Pr[Y_i = \gamma_i]| \leq \varepsilon. \quad (5)$$

It is shown in [7] that such a set S with cardinality $\mathrm{poly}(2^d, \log N, 1/\varepsilon)$ can be constructed explicitly.

We will also require a useful lemma of [2]. The following result is Lemma A.4 of [2].

Lemma 3.3 ([2]) *Let $t \geq 2$ be an even integer. Suppose $Y_1, \ldots, Y_n$ are independent random variables taking values in $[0,1]$. Let $Y = Y_1 + \cdots + Y_n$ and $\mu = \mathrm{E}[Y]$. Then*

$$\mathrm{E}[(Y-\mu)^t] \leq 2e^{1/(6t)}\sqrt{\pi t}\left(\frac{5}{2e}\right)^{t/2} \cdot (t\mu + t^2)^{t/2}.$$

We present a useful corollary:

Corollary 3.5 *Suppose $Y_1, Y_2, \ldots, Y_N \in \{0,1\}$ are* independent *random variables, and that $Y = Y_1 + \cdots + Y_N$ with $\mu = \mathrm{E}[Y]$. Let $S \subseteq \{0,1\}^N$ be any (d,ε)-approximation for the joint distribution D of the Y_i. Let $(\hat{Y}_1, \ldots, \hat{Y}_N)$ be chosen uniformly at random from S, and define $\hat{Y} = \hat{Y}_1 + \cdots + \hat{Y}_N$. Then for any even positive integer $t \leq d$ and any $b > 0$,*

$$\Pr[|\hat{Y} - \mu| \geq b] \leq \frac{2e^{1/(6t)}\sqrt{\pi t}(5/(2e))^{t/2} \cdot (t\mu + t^2)^{t/2} + \psi}{b^t},$$

where $\psi = \varepsilon(N\mu/t)^t e^{t/\mu}$ if $\mu \geq 1$, and $\psi = \varepsilon(N/t)^t e^t$ otherwise.

Proof. By the standard t-th moment inequality,

$$\begin{aligned}
\Pr[|\hat{Y} - \mu| \geq b] &\leq \frac{\mathrm{E}[(\hat{Y}-\mu)^t]}{b^t} &(6)\\
&= \frac{\sum_{i=0}^t (-1)^{t-i}\mu^{t-i}\sum_{I\subseteq\{1,\ldots,N\}:|I|=i}\mathrm{E}[\prod_{j\in I}\hat{Y}_j]}{b^t}\\
&= \frac{\sum_{i=0}^t (-1)^{t-i}\mu^{t-i}\sum_{I\subseteq\{1,\ldots,N\}:|I|=i}\Pr[\bigwedge_{j\in I}(\hat{Y}_j = 1)]}{b^t}. &(7)
\end{aligned}$$

Consider any set I in (7). Since $|I| \leq t \leq d$, we have by (5) that $\Pr[\bigwedge_{j\in I}(\hat{Y}_j = 1)]$ is within an additive distance of at most ε from $\Pr[\bigwedge_{j\in I}(Y_j = 1)]$. Thus,

$$\mathrm{E}[(\hat{Y}-\mu)^t] \leq \mathrm{E}[(Y-\mu)^t] + \sum_{i=0}^t \mu^{t-i}\sum_{I\subseteq\{1,\ldots,N\}:|I|=i}\varepsilon. \qquad (8)$$

Let us bound

$$H(N,\mu,t) := \sum_{i=0}^t \mu^{t-i}\sum_{I\subseteq\{1,\ldots,N\}:|I|=i}\varepsilon.$$

We have

$$\begin{aligned}
H(N,\mu,t) &= \varepsilon\mu^t\sum_{i=0}^t \mu^{-i}\binom{N}{i}\\
&\leq \varepsilon\mu^t(N/t)^t\sum_{i=0}^t\binom{N}{i}(t\mu^{-1}/N)^i\\
&\leq \varepsilon\mu^t(N/t)^t(1+t\mu^{-1}/N)^N\\
&\leq \varepsilon(N\mu/t)^t e^{t/\mu}.
\end{aligned}$$

If $\mu < 1$, we see that $H(N,\mu,t) \leq H(N,1,t) \leq \varepsilon(N/t)^t e^t$. Lemma 3.3, (6) and (8) now help complete the proof. □

Define t to be the smallest even integer that is at least as high as $9C$, and let $d = \lceil \log n \rceil$; we assume n is large enough so that $t \le d$. (Recall from the beginning of this subsection that we may assume that n is large enough.) Also define $\varepsilon = m^{-9C-4}$. Recall that in Algorithm A_2, each laid-out path is independently deleted with probability $(1 - 1/\lambda)$. Let Y_i be the indicator random variable for the ith laid-out path (under an arbitrary but fixed ordering of the laid-out paths) *not* being deleted, and let D denote the joint distribution of all the Y_i. (Thus, D is the product distribution of the Y_i, where $\Pr[Y_i = 1] = 1/\lambda$ for each i.) Define S to be any (d, ε)-approximation for D; as seen above, the work of [7] provides an explicit construction of such an S of cardinality $\mathrm{poly}(n, m)$. We will now show that it in fact suffices to choose $(\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_p)$ uniformly at random from S, and delete the ith path iff $\hat{Y}_i = 0$; thus, since S is polynomial-sized, we can derandomize the algorithm by exhaustive search over S. The values of x and λ are the same as in Section 3.3.

Recall that Algorithm A_2 is of relevance only in the case where $opt \le \log n$. Let P be an optimal VR solution. Suppose, as mentioned above, that we delete the ith path iff $\hat{Y}_i = 0$. Then, since $opt \le d$, bound (5) shows that in place of (3), we now have

$$\Pr[\text{all laid-out paths intersected by } P \text{ survive}] \ge (1/\lambda)^{opt} - \varepsilon = n^{-C} - m^{-9C-4}. \quad (9)$$

Suppose the laid-out paths of length more than x are those numbered $i_1, i_2, \ldots, i_N$. Define $Z = \sum_j Y_{i_j}$, and $\hat{Z} = \sum_j \hat{Y}_{i_j}$. Note that $\hat{Z}$ is the number of laid-out paths of length more than x that remain undeleted (in our current randomized algorithm which samples from S). Analogous to (4), we wish to upper-bound $\Pr[\hat{Z} \ge 2m/(\lambda x)]$. Let $\mu = \mathrm{E}[Z]$. Let $\psi = \varepsilon(N \cdot \mu/t)^t e^{t/\mu}$ if $\mu \ge 1$, and $\psi = \varepsilon(N/t)^t e^t$ otherwise. Thus, $\Pr[\hat{Z} \ge 2m/(\lambda x)]$ is at most

$$\Pr[|\hat{Z} - \mu| \ge m/(\lambda x)] \le \frac{2e^{1/(6t)}\sqrt{\pi t}(5/(2e))^{t/2} \cdot (t\mu + t^2)^{t/2}}{(m/(\lambda x))^t} + \frac{\psi}{(m/(\lambda x))^t}, \quad (10)$$

by Corollary 3.5.

We now bound the right-hand-side of (10). Recall from Section 3.3 that $\mu \le m/(\lambda x)$, and that $n^{1/4} \le m/(\lambda x)$. Also, $t \in [9C, 9C+2]$ and $\varepsilon = m^{-9C-4}$. Thus, the first summand in the r.h.s. of (10) is at most

$$O(\mu^{t/2} \cdot (m/(\lambda x))^{-t}) \le O((m/(\lambda x))^{-t/2}) \le O(n^{-t/8}) \le O(n^{-9C/8}), \quad (11)$$

where the constant hidden in the $O(\cdot)$ notation depends upon C. As for the second summand in the r.h.s. of (10), we have

$$\psi \le \varepsilon(N \cdot \max\{\mu, 1\}/t)^t e^t \le \varepsilon(Nm/(\lambda x t))^t e^t \le \varepsilon(m^2/(\lambda x t))^t e^t,$$

since $N \le m$. Thus,

$$\psi \cdot (m/(\lambda x))^{-t} \le \varepsilon(me/t)^t = m^{-9C-4}(me/t)^t. \quad (12)$$

Since $t \in [9C, 9C+2]$ and $m \ge n$, we see that for large enough n, the bounds of (11) and (12) add up to at most $n^{-C}/4$. So, by (10), $\Pr[\hat{Z} \ge 2m/(\lambda x)] \le n^{-C}/4$. Thus, by (9), we have that with probability at least $3n^{-C}/4 - m^{-9C-4} \geq 3n^{-C}/4 - n^{-9C-4}$, the events (i) all laid-out paths intersected by P survive, and (ii) $\hat{Z} \leq 2m/(\lambda x)$, hold simultaneously. In particular, these events (i) and (ii) hold simultaneously with positive probability (if n is large enough), which is what we required for the derandomization. This completes the proof of Theorem 3.4.

The same approach and results hold for GVR, by deleting the laid-out sets instead of the laid-out paths.

3.5 Approximation for the weighted cases

Recall that in WVR we have a weight for each laid-out path, which equals its length. We now show a simple $O(\sqrt{m})$-approximation for WVR. A simple algorithm returns a solution of value at most opt^2 for WVR. This algorithm, which we call A_3, guesses the value of opt as $1, 2, 4\ldots$; note that $opt \leq m$. If $opt \geq \sqrt{m}$, we simply include all laid-out paths, leading to an $O(\sqrt{m})$-approximation.

Suppose $opt < \sqrt{m}$. We can restrict attention to laid-out paths of length at most opt. Accordingly, the algorithm: (i) removes all the laid-out paths of length $> opt$; (ii) assigns weight 1 to all remaining arcs that lie in laid-out paths, and weight 0 to arcs that do not lie in any laid-out path; and (iii) applies SPA. Clearly, the path that we get has cost at most opt^2.

Theorem 3.5 *Algorithm A_3 is an $O(m \log m)$-time $O(\min\{opt, \sqrt{m}\})$-approximation algorithm for WVR.*

We conclude this subsection by sketching a reduction from PVR, the weighted version of VR with generic weights assigned to the laid-out paths, to VR. This shows that all the approximation results for VR apply also to PVR if the values of m and n in the approximation ratio are both replaced by mw, where w is the maximum weight assigned to a laid-out path. Moreover, the discussion below points out that we may assume that w is polynomial in m.

In order to reduce PVR to VR we proceed in two steps. First, we make use of a general scaling technique due to Crescenzi et al. [5] which shows that without loss of generality one can consider just polynomially bounded weights for the laid-out paths. More precisely, for any given fixed $\varepsilon > 0$, we can do the scaling so as to loose a factor of at most $(1+\varepsilon)$ in the approximation. The second step is a very simple reduction from PVR with polynomial weights to VR, defined as follows. Let $P = a_1, a_2, \ldots, a_k$, where the $a_i = u_{i-1}u_i$ $(1 \leq i \leq k)$ are arcs, be a laid-out path of weight y. First we replace each arc $a_i = u_{i-1}u_i$ by a path of length y, $A_i = x_0^i x_1^i, x_1^i x_2^i, \ldots, x_{y-1}^i x_y^i$ where $x_0^i = u_{i-1}$ and $x_y^i = u_i$. Then, we form y new laid-out paths by stringing together all arcs $x_{j-1}^i x_j^i$: $P_j = x_{j-1}^1 x_j^1 x_{j-1}^2 x_j^2 \ldots x_{j-1}^k x_j^k$, $1 \leq j \leq y$. It is easy to see that any $s-t$ path intersecting P in the original graph corresponds to a path intersecting $P_1, \ldots, P_y$ in the new graph, and vice versa. Letting w denote the maximum weight of a laid-out path in the original graph, we have that the new graph has $O(mw)$ vertices and arcs. Moreover, the maximum length of a laid out path, ℓ, is unchanged. This shows

Proposition 3.2 *Any polynomial-time $O(f(\ell, m, n))$-approximation algorithm for VR yields a polynomial-time $O(f(\ell, mw, mw))$-approximation algorithm for PVR with maximum weight w.*

3.6 Limits to approximation

We briefly illustrate some limitations to approximating VR by some specific natural techniques.

First, in terms of approximation algorithms via linear programming (LP) relaxations, there is a natural integer programming formulation for VR, and a corresponding LP relaxation. We have an integer variable $y_Q \in \{0, 1\}$ for each laid-out path Q, and a "capacity" $x_e \in \{0, 1\}$ for each arc e. Given an st-path P, x_e is supposed to be 1 iff e lies in P, and y_Q is 1 iff P arc-intersects Q. We clearly have the constraints that for each laid-out path Q and all $e \in Q$, $x_e \leq y_Q$. The integer programming formulation is to install these capacities and send a unit flow from s to t respecting these capacities, so as to minimize $\sum_Q y_Q$. The LP relaxation is to let the variables x_e, y_Q lie in $[0, 1]$. Unfortunately, one can show instances where the optimal solution is as high as $\Theta(\ell)$ times the LP's optimal solution. We sketch the proof of this gap for odd ℓ; a minor modification works for the case of even ℓ. Consider the digraph with vertex set $\{s, t, u_0, u_1, \ldots, u_\ell\}$. The arcs are of three types:

- a single laid-out path $Q = (u_0, u_1, \ldots, u_\ell)$;
- arcs $(s, u_0), (s, u_2), \ldots, (s, u_{\ell-1})$, and
- arcs $(u_1, t), (u_3, t), \ldots, (u_\ell, t)$.

It is easy to see that any st-path must edge-intersect the laid-out path Q. However, the above LP relaxation has a feasible solution of value $2/(\ell + 1)$: Set $y_Q = 2/(\ell + 1)$, and send one unit of flow on each of the paths

$$(s, u_0, u_1, t),\ (s, u_2, u_3, t), \ldots, (s, u_{\ell-1}, u_\ell, t).$$

Furthermore, even some natural pre-processing (e.g., as in our $\lceil \ell/2 \rceil$-approximation for VR), still leaves a gap of $\Theta(\ell)$.

Second, SPA-based algorithms cannot in general give solutions much better than those here. This intuition can be formalized by constructing an infinite set of instances such that, for any "reasonable" weight function f on the arcs, SPA returns a solution of value $\Theta(\ell) = \Theta(\sqrt{m})$, whereas the optimal solution has value 1. (For WVR, these values are respectively $\Theta(\ell^2)$ and ℓ.) By "reasonable" we mean that f can be any function of the length of the laid-out path and of the position of the arc (first arc of the path, second arc of the path, etc).

We consider a class of instances defined by a parameter $k \geq 2$. Besides vertices s and t, G contains $k+1$ vertex-disjoint laid-out paths $Q, P^1, \ldots, P^k$ of length $\ell = 2k-1$. Path Q visits vertices $q_1, q_2, \ldots, q_{2k}$, whereas path P^i, $i = 1, \ldots, k$, visits vertices $p_1^i, p_2^i, \ldots, p_{2k}^i$. In addition to the arcs in the laid-out paths, G contains arcs (s, q_{2k-1}),

(q_{2k}, q_{2k-3}), (q_{2k-2}, q_{2k-5}), $\ldots$, (q_6, q_3), (q_4, q_1), (q_2, t); as well as arcs (s, p_1^1), (p_2^1, p_3^2), (p_4^2, p_5^3), $\ldots$, $(p_{2k-2}^{i-1}, p_{2k-1}^{i})$, (p_{2k}^{k}, t).

Suppose one uses any "reasonable" weight function to the arcs of G, according to the definition above. Then, the optimal VR and WVR solutions are associated with path $(s, q_{2k-1}), (q_{2k-1}, q_{2k})$,
$(q_{2k}, q_{2k-3}), (q_{2k-3}, q_{2k-2}), \ldots, (q_4, q_1), (q_1, q_2), (q_2, t)$, and have cost 1 and $2k-1$, respectively. This is also an optimal solution for SPA, but a solution of the same value is given by the path $(s, p_1^1), (p_1^1, p_2^1), (p_2^1, p_3^2), \ldots, (p_{2k-2}^{i-1}, p_{2k-1}^{i}), (p_{2k-1}^{i}, p_{2k}^{i}), (p_{2k}^{i}, t)$, whose VR and WVR values are respectively k and $(2k-1)k$.

4 Experimental Results

In this section we discuss the outcome of a simulation in which the effectiveness of the algorithms A_0 and A_1 is tested. Starting with an initial random graph we generate a sequence of requests in the following way. We use a queue called the *event-queue*, storing events of two possible kinds: *request* and *release*. Associated with every event there is an *event-time* denoting the time at which the event is triggered (processed). At any instant of time the event-queue is sorted by event-time so that the next event to be processed always appears at the front end of the queue. Each request-event has information about the source node, the destination node and the event-time, while a release-event contains information about the source node, the destination node, the route and wavelength of the lightpath, and the event-time.

Initially the event-queue has one request-event for each node, with the event-time equal to 0. Thereafter events are processed according to the following algorithm:

1. Remove the event from the front end of event-queue. Call it e.

2. **if** e is a *request* **then**

   (a) set current-time equal to the event-time of e;

   (b) Let s_e and d_e be the source and destination nodes of e, respectively;

   (c) Generate a new *request* event e' with source s_e as follows. Generate an inter-arrival time $t_{e'}$ using an exponential distribution with mean $1/r$ (r a parameter) and set the event-time of e' equal to current-time plus $t_{e'}$. Generate the destination node $d_{e'}$, choosing it uniformly at random among all vertices except s_e. Insert e' in the event-queue in the sorted order based on the event-time.

   (d) Process e (see below).

   (e) **if** request e is honored **then** generate a new *release* event e'' with source s_e as follows: (a) Generate a holding time $t_{e''}$ using an exponential distribution with mean $1/r$ (r a parameter); (b) Choose a destination d uniformly at random among all nodes except s_e; (c) Set the event-time of e'' equal to the current-time plus $t_{e''}$. Insert e'' in the event-queue in the sorted order based on the event-time.

3. **else** (e is a *release* event) release the lightpath associated with e by updating the status information of the edges of the path.

4. **repeat** the above steps m times ($m = 10^5$, a parameter).

We now specify how a request is processed. When a request arrives, it is first checked whether a free source-destination path is available. In case there is no such path, algorithm A_0 (respectively, A_1) is used to look for a path intersecting a (hopefully) small number of already laid-out paths. A path is allowed to be intersected only if it can be migrated to another wavelength. Prior to establishing a new path, the intersecting paths are migrated. If neither a free nor an available laid-out path is found, the request is not honored and rejected.

We measure the following relevant performance parameters of algorithms A_0 and A_1:

- The *blocking probability*, that is the fraction of requests which fail to be honored, plotted against the arrival rate of the requests. This is perhaps the most important performance parameter;

- The average number of intersected paths, plotted against the request arrival rate;

- The average number of hops, i.e., the average path length of the requests that are honored, plotted against the arrival rate;

- The average connection request processing time, measured in CPU ticks, again plotted against the arrival rate of the requests.

A WDM network with W wavelengths is represented as a layered graph each corresponding to a wavelength, called a *wavelength graph.* All the laid-out paths on a wavelength graph are arc-disjoint.

As said, the performance of the algorithms is evaluated on randomly generated networks. A randomly generated network is characterized by N and p, where N is the number of nodes and p is the edge probability, i.e., for any two given nodes u and v, a "coin" is tossed and if the outcome is head, an event which has probability p of happening, the edge is included in the network. A small value of p typically generates a sparsely connected network, while if p is large the network is likely to be densely connected. In our experiments $N = 32$ and $p = 0.1, 0.2$. For each experiment we have generated 50 graphs, for each value of p, and for each graph we have generated in the order of $m = 10^5$ connection requests.

Fig. 4 through Fig. 7 plot the performance of the algorithms for random graphs with $N = 32$ and $p = 0.10$. The blocking probability by different algorithms is plotted in Fig. 4 as a function of arrival rate per node. The arrival rate is measured in Erlangs. We consider three algorithms: no-intersection, A_0, and A_1. The no-intersection algorithm does not allow a new path to intersect with any of the laid-out paths. It can observed that the performance gain by A_0 and A_1 is significant. The figure shows that more than 20% of the connections rejected by no-intersection algorithm have been successfully routed by A_0 and A_1. As the load increases, the above percentage of improvement slowly decreases. This is because, under heavy load, the number of laid-out paths that qualify for migration is small.

Fig. 5 plots the average number of intersecting paths by a new path by A_0 and A_1 for different arrival rates per node. The plots show that A_0 performs better than A_1. This is because, A_1 fails to identify that two edges belong to the same path. However, for A_0 it is possible by way of crossover edges (shortcut arcs). It also shows that arrival rate does not have much effect on the number of intersecting paths.

Fig. 6 plots the average number of hops used by an intersecting path by A_0 and A_1 for different arrival rates per node. The curves show that A_1 selects shorter hop paths than A_0. The reason is that A_1 performs an exhaustive search. The plot also shows that as the arrival rate increases the hop count decreases. This is because, under heavy load conditions, longer paths do not qualify for intersection, as their migration might not be possible.

The actual running time to find a path is measured in terms of CPU ticks (1 CPU tick = 10 milliseconds). The time required by A_0 and A_1 for different arrival rates per node are plotted in Fig. 7. Algorithm A_1 requires more time than A_0, since it considers several possible solutions before choosing the best one.

Fig. 8 through Fig. 11 plot the performance of the algorithms for a random graph G_2 with $N = 50$ and $p = 0.20$. The connectivity of G_2 is denser than that of G_1. The plots demonstrate the usefulness of the algorithms A_0 and A_1. Due to dense connectivity, the performance on G_2 in terms of blocking probability is better than G_1. For the same reason, the average number of intersected paths is slightly more and the average number of hops per intersected path is slightly less for G_2 than that for G_1. Also, the average connection processing time for G_2 is less than that for G_1.

Thus we see that with a modest amount of computational effort, our algorithms can lead to better resource utilization in WDM networks, via wavelength rerouting.

Acknowledgments. We thank Sanjeev Arora, Sanjeev Khanna, Goran Konjevod, Madhav Marathe, Riccardo Silvestri and Luca Trevisan for helpful discussions. Part of this work was done at the University Ca' Foscari, Venice.

References

[1] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *Journal of Computer and System Sciences*, 54:317–331, 1997.

[2] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proc. IEEE Symposium on Foundations of Computer Science*, 276–287, 1994.

[3] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 345–353, 2000.

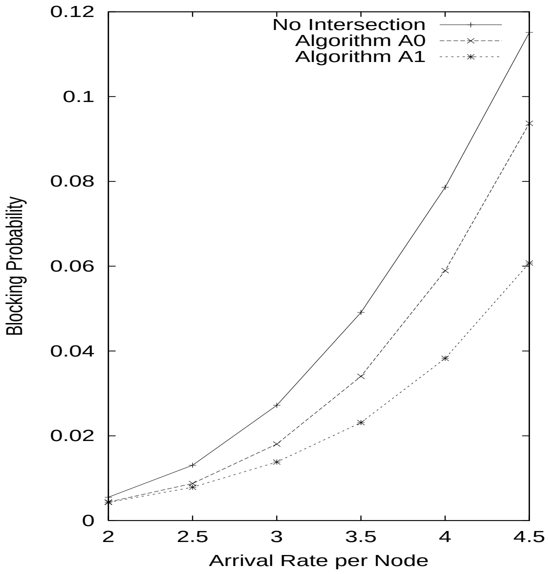

Figure 4: Blocking probability vs. arrival rate per node for a 50-node random graph with $p = 0.10$.

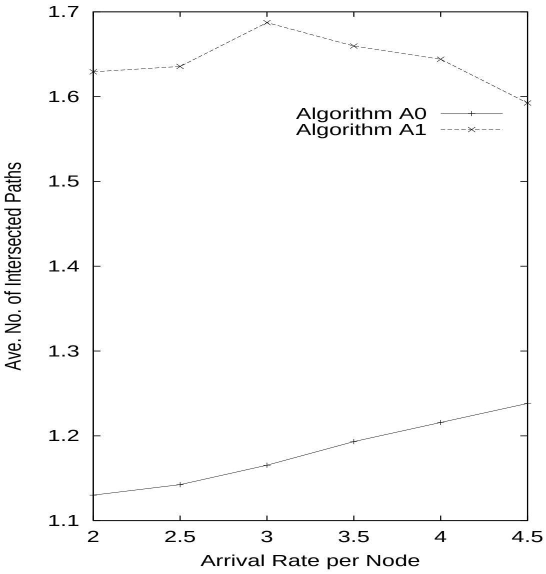

Figure 5: Average number of intersected paths vs. arrival rate per node for a 50-node random graph with $p = 0.10$.

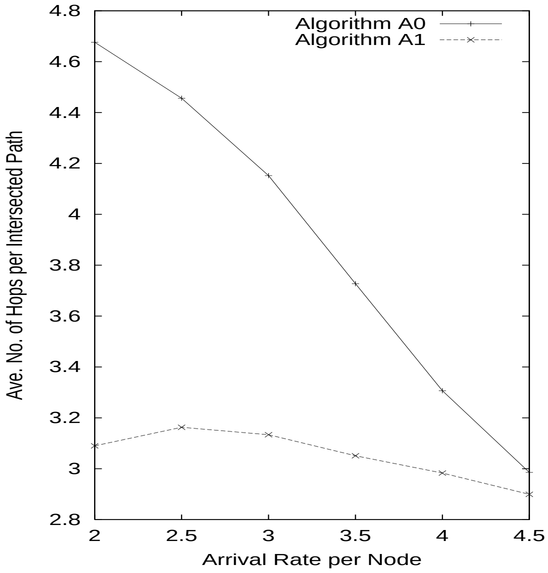

Figure 6: Average number of hops per intersected path vs. arrival rate per node for a 50-node random graph with $p = 0.10$.

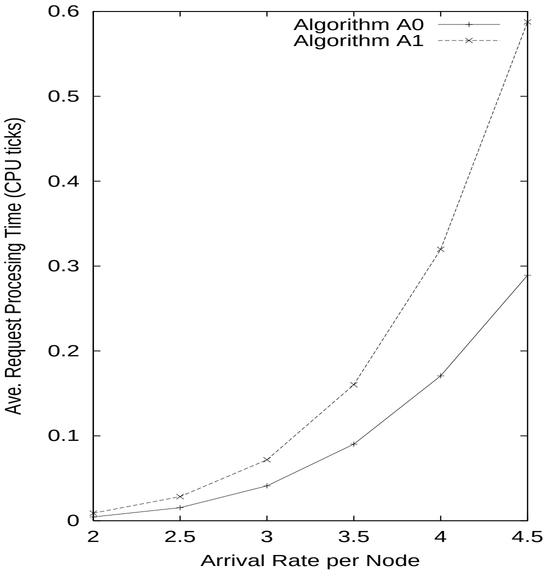

Figure 7: Average connection request processing time in CPU ticks for a 50-node random graph with $p = 0.10$.

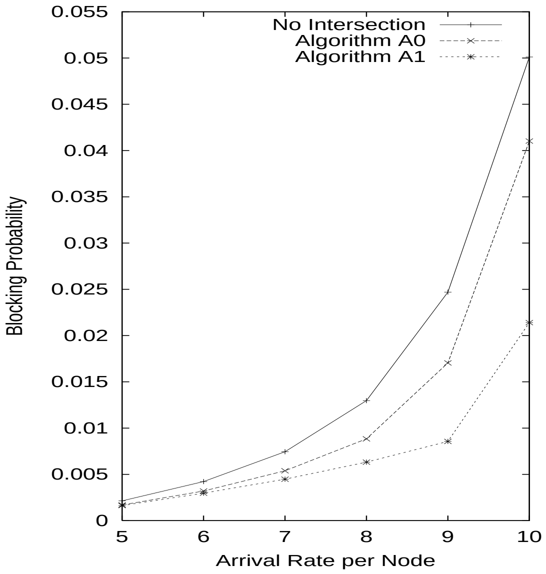

Figure 8: Blocking probability vs. arrival rate per node for a 50-node random graph with $p = 0.20$.

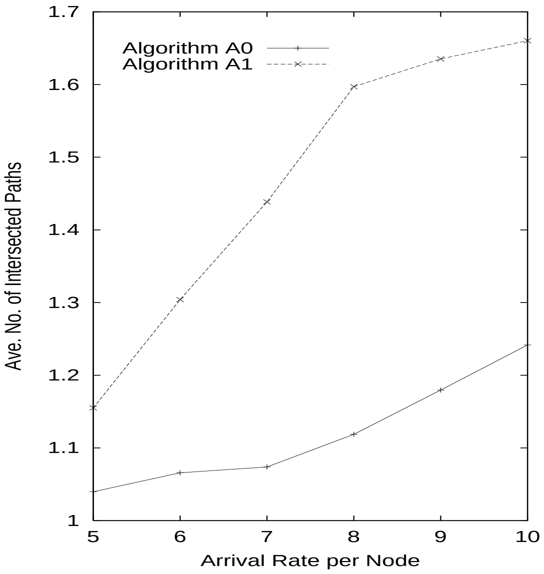

Figure 9: Average number of intersected paths vs. arrival rate per node for a 50-node random graph with $p = 0.20$.

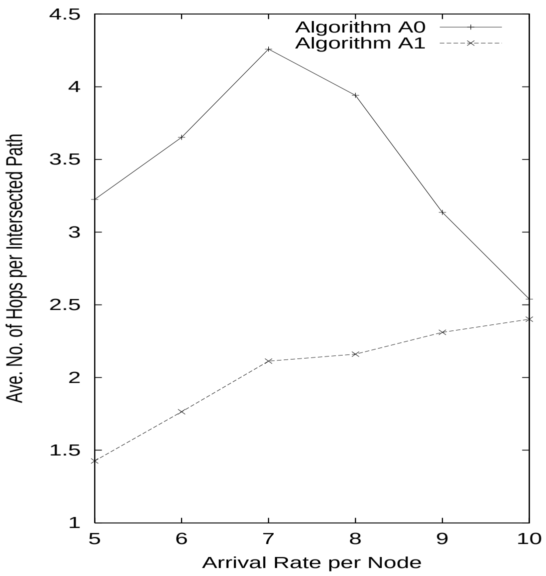

Figure 10: Average number of hops per intersected path vs. arrival rate per node for a 50-node random graph with $p = 0.20$.

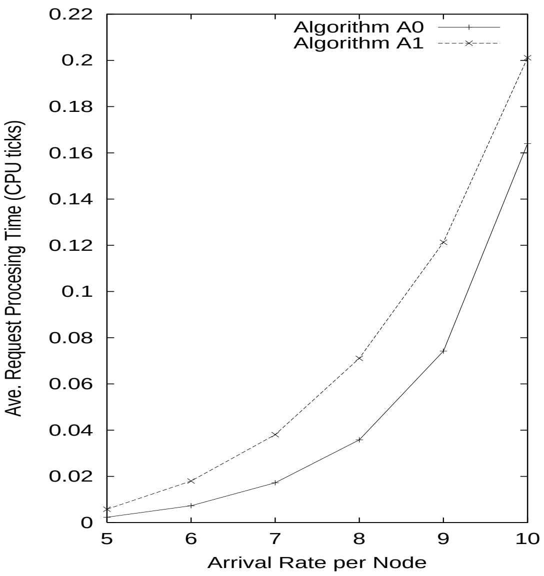

Figure 11: Average connection request processing time in CPU ticks for a 50-node random graph with $p = 0.20$.

[4] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: An approach to high bandwidth optical WANs. IEEE Trans. on Communications, 40:1171–1182, 1992.

[5] P. Crescenzi, R. Silvestri, and L. Trevisan. To Weight or not To Weight: Where is the Question? In *Proc. 4th Israel Symposium on Theory of Computing and Systems*, 68–77, 1996.

[6] Y. Dodis and S. Khanna. Designing networks with bounded pairwise distance. In *Proc. ACM Symposium on Theory of Computing*, 750–759, 1999.

[7] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Veličković. Efficient approximations for product distributions. *Random Structures & Algorithms*, 13: 1–16, 1998.

[8] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57:187–199, 1998.

[9] M. R. Garey and D. S. Johnson. Computers and Intractability. W.H. Freeman & co. San Francisco 1979.

[10] J. Håstad. Clique is Hard to Approximate within $n^{1-\varepsilon}$. To appear, *Acta Mathematica.*

[11] J. Håstad. Some optimal inapproximability results. In *Proc. ACM Symposium on Theory of Computing*, 1–10, 1997.

[12] I. Holyer. The NP-completeness of Edge Colouring. *SIAM Journal on Computing*, 10:718–720, 1981.

[13] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: The approximability of minimization problems. ECCC tech report TR96-064, available at `http://www.eccc.uni-trier.de/eccc-local/Lists/TR-1996.html`. Also see: *Proc. IEEE Conference on Computational Complexity*, 282–296, 1997.

[14] S. O. Krumke and H.-C. Wirth. On the minimum label spanning tree problem. *Information Processing Letters*, 66:81–85, 1998.

[15] K. C. Lee and V. O. K. Li. A wavelength rerouting algorithm in wide-area all-optical networks. *IEEE/OSA Journal of Lightwave Technology*, 14:1218–1229, 1996.

[16] G. Mohan and C. Siva Ram Murthy. Efficient algorithms for wavelength rerouting in WDM multi-fiber unidirectional ring networks. *Computer Communications*, 22:232–243, 1999.

[17] G. Mohan and C. Siva Ram Murthy. A time optimal wavelength rerouting for dynamic traffic in WDM networks. *IEEE/OSA Journal of Lightwave Technology*, 17:406–417, 1999.

[18] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[19] L. J. Stockmeyer. Planar 3-colorability is NP-complete. *SIGACT News*, 5:19–25, 1973.