Faster Algorithms for Optical Switch Configuration

S Ravi Kumar*
Dept. of Computer Science
Cornell University
Ithaca, NY 14853.

Alexander Russell†
Dept. of Mathematics
M.I.T.
Cambridge, MA 02139.

Ravi Sundaram‡
Lab. for Computer Science
M.I.T.
Cambridge, MA 02139.

Abstract

All-optical networks using wavelength division multiplexing are increasingly coming to be regarded as the technology of choice for the next generation of wide-area backbone networks. These networks incorporate optical switches that employ the concept of Latin Routers [BH93] for assigning wavelengths to routes. The issue of maximizing wavelength utilization at these switching devices is of great importance since it leads to significant improvements in overall network performance [CB95]. In this paper, we present two fast approximation algorithms – GREEDY *and* MATCH *– for the problem of maximizing wavelength utilization at Latin Routers. These are the first known polynomial-time approximation algorithms for the problem of maximizing the number of entries that can be added to a partially filled Latin Square that achieve non-trivial worst-case performance guarantees. These algorithms are easily implementable and have very small constants in their running times making them eminently suitable for actual use in real-world optical switches. We also provide strong experimental evidence to show that, in practice, these algorithms are near-optimal.*

1 Introduction

1.1 Motivation

Developments in fiber-optic networking technology using *wavelength division multiplexing* (WDM) have finally reached the point where it is being considered as the most promising candidate for the next generation of wide-area backbone networks. These are highly flexible networks capable of supporting tens of thousands of users and capable of providing capacities on the order of gigabits-per-second per user [CNW90, Gre92, Ram93]. WDM optical networks utilize the large bandwidth available in optical fibers by partitioning it into several channels each at a different optical wavelength [BH92, CNW90, IEE93, IK92].

The typical optical network consists of routing nodes interconnected by point-to-point fiber-optic links. Each link supports a certain number of wavelengths. The routing nodes are capable of *photonic switching*, also known as *dynamic wavelength routing*, which involves the setting up of *lightpaths* [CB95, CGK92, ZA94]. A lightpath is an optical path between two nodes on a specific wavelength. The optical switch at a node assigns the wavelengths from an incoming port to an outgoing port. Figure 1 shows an optical switch S. The routing pattern of the optical switch leads itself naturally to a modeling in terms of a matrix. The matrix in Figure 1 shows the corresponding routing pattern in the switch S. A value of k in entry (i, j) of the matrix denotes that wavelength k is routed from input port i to output port j. This assignment is changeable and can be controlled electronically. When there is no wavelength routed from i to j, the corresponding matrix entry is left blank.

There are some constraints on the construction and the behavior of the switches. The first is that the switches have an equal number of input and output ports. Secondly, each entry of the matrix must have *at most* one non-zero value. This is a compromise between hardware complexity and performance because switches that permit routing of *multiple* wavelengths between the same input and output ports require additional hardware that is both complex and costly [CB95]. The third constraint is that of achieving *conflict-free* wavelength routing – the wavelengths assigned to messages at an input port are all distinct and there must be no wavelength conflict at any of the output ports.

These restrictions constrain the simple matrix model of the switch. What we get is in fact a *Latin Square* (LS). Latin Squares were first used to model optical switches in [BH93] (*Latin Routers*). More formal details are given in Section 2. Though the

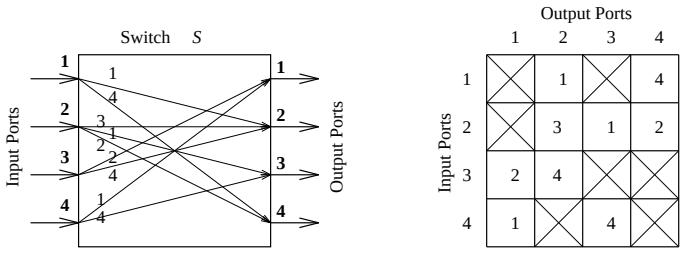

Figure 1: Example of Optical Switch and Port Assignment

number of wavelengths at a given node is limited by the technology (tunability of lasers, amplifier bandwidth, etc.), nevertheless, it is possible to build a transparent wide-area optical network by *spatial reuse* of wavelengths. Figure 2 shows an all-optical wavelength routing network with two 3×3 switches. Though each switch has only 3 input and output ports, it is still possible to set up all the shown interconnections by (re)using wavelength **1** both for the $A - B$ and $C - D$ routes.

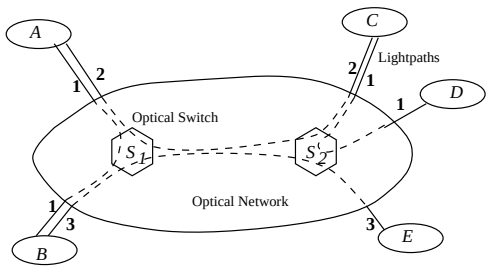

Figure 2: Example of an All-Optical Wavelength Routing Network

* Supported by ONR Young Investigator Award N00014-93-1-0590, NSF grant DMI-91157199, and career grant CCR-9624552. This work was done while the author was visiting M.I.T. E-mail: ravi@cs.cornell.edu.

† Supported by an NSF Graduate Fellowship and grants NSF 92-12184, AFOSR F49620-92-J-0125, and DARPA N00014-92-1799. E-mail: acr@math.mit.edu.

‡ Supported by grants NSF 92-12184, AFOSR F49620-92-J-0125, and DARPA N00014-92-1799. E-mail: koods@theory.lcs.mit.edu.

1.2 Previous Work

The problem of wavelength routing and switch assignments are areas of active research in optical networks. Various policies for wavelength routing, and the corresponding assignments of wavelengths along the routes, have been reported in the literature [CGK92, LL93, RS94, ZA94]. These policies aim at minimizing congestion, average hop-distance, call blocking probability or maximizing traffic throughput, and clear channel density.

Chen and Banerjee [CB95] realized that these policies often lead to scenarios where a significant number of wavelengths, available at the input and output ports of the routers, cannot be used due to potential wavelength conflict. They focus on maximizing the wavelength utilization at a Latin Router in static wavelength routing. They show that achieving improved wavelength utilization also automatically improves overall network performance. Some of the wavelength assignment in routers are predetermined. This could be due to traffic demands, delay limits, fault-tolerance requirements, or other constraints. This scenario corresponds to a partially filled Latin Square. Hence, algorithms to maximize the number of non-zero entries in a partially filled Latin Square are very useful. They present two algorithms, Algorithm 1 – a backtracking algorithm that takes exponential time in the worst case and Algorithm 2 – a heuristic that could potentially modify preassigned wavelengths, thus rendering it unfit for use in most situations of practical interest.

1.3 Our Contributions

In this paper we focus on algorithms that maximize the wavelength utilization at a Latin Router. Reducing the number of unassigned or zero entries (i.e., increasing the density) of the PLS associated with a router is of the utmost practical importance in optical networks as this ensures reduced wastage of the valuable resources of ports and wavelengths. We present two fast approximation algorithms – GREEDY and MATCH – for the problem of maximizing the number of entries that can be added to a partially filled Latin Square. These are the first known polynomial-time approximation algorithms for the Latin Square completion problem that achieve non-trivial worst-case performance guarantees. These algorithms are easily implementable and have very small constants in their running times thus making them eminently suitable for actual use in real-world optical switches. We also provide strong experimental evidence to show that, in practice, these algorithms are, in fact, near-optimal. Unlike Algorithm 2 of [CB95], neither of these algorithms changes preassigned entries. GREEDY possesses the extra advantage of being usable in a dynamic situation where routing requests are online.

In addition, we present BRANCHBOUND – a back-tracking algorithm that finds an optimal solution to the problem of maximizing the number of entries that can be added to a partially filled Latin Square. BRANCHBOUND is a more efficient version of Algorithm 1 of [CB95], that dynamically prunes the search space to save time. The efficacy of this pruning is demonstrated by the fact that we are able to obtain performance results for *large* Latin Routers in our simulations. Without some form of pruning, any attempts at simulation would fail to yield results in a reasonable amount of time.

All the algorithms we present possess provable *worst-case* guarantees. This is very important from a practical standpoint. Heuristics that do not have worst-case performance guarantees tend to be commercially unviable since they cannot provide any insurance against downside risks. And in fact, the results in Figure 8 indicate that our algorithms achieve near-optimal *average-case* performance. The average-case performance is of great significance since these algorithms are meant to be used over and over again in on-line situations.

1.4 Organization

The rest of the paper is organized as follows: Section 2.1 contains a formal description of the problem; Section 2.2 is a description of BRANCHBOUND; Section 2.3 is a description of GREEDY; Section 2.4 is a description of MATCH; Section 3 contains the experimental results showing that MATCH and GREEDY are fast and near-optimal in practice.

2 Algorithms for Switch Configuration

2.1 The Problem

As stated before, the problem of achieving *conflict-free wavelength routing* is modeled by utilizing Latin Routers [BH93]. These are routing devices that employ the concept of a *Latin Square* (LS). Before actually stating the problem we develop some basic concepts and terminology.

Definition 1 *An $n \times n$ LS L is an $n \times n$ matrix with the following properties: (1) $\forall i, j, L_{i,j} \in \{1, 2, \ldots, n\}$, (2) $\forall i, j_1 \neq j_2, L_{i,j_1} \neq L_{i,j_2}$, and (3) $\forall i_1 \neq i_2, j, L_{i_1,j} \neq L_{i_2,j}$*

Stated informally, an $n \times n$ LS is a matrix with entries from $\{1, 2, \ldots, n\}$ such that no row or column contains the same element twice.

Definition 2 *An $n \times n$* Partial Latin Square *(PLS) is an $n \times n$ matrix with entries from the set $\{0\} \bigcup \{1, 2, \ldots, n\}$ (0 is used as a placeholder to denote emptiness) such that each row and each column never contains an element from the set $\{1, 2, \ldots, n\}$ more than once.*

If L is a PLS then a non-zero entry L_{ij} denotes that the wavelength L_{ij} is routed from input port i to output port j. A zero entry denotes an unassigned entry. Thus the state of a Latin Router can be described by a PLS. We also refer to the entries in a PLS/LS as *colors*. We use the terminology *degree of freedom* of a 0 entry in a PLS to refer to the number of colors that can be validly used in that entry. See Figure 3 for an example of a PLS. The color of the entry $(1, 3)$ is 4. The degree of freedom of the $(2, 3)$ entry is 1 since there is only one color that can be validly used in that entry, namely the color 3.

0	0	4	3
2	4	0	1
3	1	0	4
4	3	1	2

1	2	4	3
2	4	3	1
3	1	2	4
4	3	1	2

Figure 3: A 4×4 PLS and its corresponding LS

Definition 3 *The density of an $n \times n$ PLS is the fraction of entries that are non-zero. $\mu(L)$ is used to denote the density of L.*

The PLS in Figure 3 has density $3/4$. An LS is a PLS that has density 1. As noted before, a reduction in the zero entries achieves an increased usage of ports and wavelengths. This motivates the following definitions:

Definition 4 *A PLS L^* is said to* extend *or be an* extension *of a PLS L if L^* can be obtained by altering only zero entries of L. A PLS that cannot be extended is said to be* blocked.

Definition 5 *A PLS is said to be* completable *if it can be extended to an LS.*

See Figure 3 for an LS obtained by extending the PLS. Not all PLSs can be completed (see Figure 4). At this point we have developed sufficient notation to state our problem succinctly:

Given a PLS, find an extension of it with maximum density.

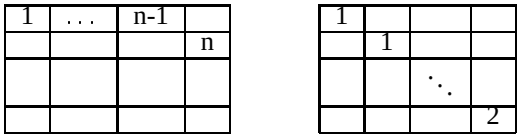

Figure 4: Blocked $n \times n$ PLSs with n entries

The computational complexity of finding a maximum-density extension of a given PLS was unresolved for a long time. Colbourn, finally, proved that the problem of determining whether a given PLS is completable is **NP**-complete [Col84]. Therefore, the natural goal is to find a fast (polynomial time) algorithm that given a PLS, extends it to an LS of as large a density as possible. We formally define the notion of an approximation algorithm.

Definition 6 *Let L be any PLS. Let* OPT *denote a maximum density extension of L. A polynomial-time algorithm is said to be a ρ-approximation algorithm if, given L, it finds an extension* HEU *such that* $(\mu(\text{HEU}) - \mu(L)) \geq \frac{1}{\rho}(\mu(\text{OPT}) - \mu(L))$.

It is clear that $\rho \geq 1$ and the closer ρ is to 1, the better the quality of approximation.

In the rest of this section, we discuss three algorithms: The first algorithm, BRANCHBOUND, is a backtracking algorithm that does dynamic pruning to find a maximum density extension OPT. The second algorithm, GREEDY, is a 3-approximation algorithm. The third algorithm, MATCH, is a 2-approximation algorithm.

2.2 Branch and Bound Algorithm

The algorithm in [CB95] is the natural backtracking algorithm. It tries to improve the running time by exhaustively trying the empty entries in the increasing order of their degrees of freedom. We give a better backtracking algorithm below. This algorithm, in addition to using the above heuristic, prunes the search space efficiently and is hence faster in practice.

Algorithm Description. Backtracking is a general and proven method for exhaustively going through the space of all possible solutions. BRANCHBOUND does a search of the (potentially $(n+1)$-ary) tree corresponding to the choices of colors for each 0 entry in the PLS. But since the number of possible nodes in the tree is hugely exponential – $\Theta(n^{n^2})$ – BRANCHBOUND employs four techniques to cut down the running time.

Preprocessing. To prevent costly recomputation at each step BRANCHBOUND sets up an elaborate data structure ahead of time. This data structure permits quick table lookup of the set of possible colors that can be validly used in a 0 entry. The data structure also permits a quick lookup of the list of invalidations that would be caused by setting entry $L_{i,j}$ to color k, for all i, j, k.

Implicit Depth first search. The entire algorithm is coded in a conventional higher-level language. To save time spent processing function calls BRANCHBOUND is implemented as an iterative loop that implicitly traverses the tree in depth first fashion.

Degree of Freedom. To reduce the number of backtracking steps the degree of freedom concept is used in a fashion identical to that of Algorithm 1 of [CB95]. The order of 0 entries tried for backtracking is based on the degree of freedom of the entries.

Dynamic Pruning. At any point the algorithm maintains the best solution found so far. At each node it does a quick evaluation (using the data structure) to figure out if the best potential solution in the subtree rooted at that node could better the best solution found so far. If so it continues the search else it aborts the search at that node.

```
BRANCHBOUND(L):
    preprocess L
    d :=↓, L* := L, s := state at the root
    while d ≠ ⊥ do
        case d
            ↓:  if ∃ (possibly better) CHILD(s) then
                    L* := max(L, L*), s := CHILD(s), d :=↓
                else d :=→
            →:  if ∃ SIBLING(s) then
                    s := SIBLING(s), d :=↓
                else d :=↑
            ↑:  if ∃ PARENT(s) then
                    s := PARENT(s), d :=→
                else d := ⊥
    end while
    L = L*
```

Figure 5: The BRANCHBOUND Algorithm

```
GREEDY(L):
    for 1 ≤ i, j ≤ n do
        for 1 ≤ k ≤ n do
            if L_{i,j} = 0 ∧ LEGAL (L, i, j, k) then
                L_{i,j} := k
        end for
    end for
```

Figure 6: The GREEDY Algorithm

A sketch of the algorithm is shown in Figure 5.

Performance. It is clear that BRANCHBOUND obtains a solution that is as good as that of OPT.

Running Time. Since the problem is **NP**-complete, BRANCHBOUND could potentially take $\Omega(n^{n^2})$ time.

2.3 Greedy Algorithm

Algorithm Description. GREEDY is a straightforward algorithm that fills in the 0 entries in a greedy manner. GREEDY considers each 0 entry in the PLS, fills it with any legal color, and moves on to the next 0 entry. The algorithm is presented in Figure 6. The algorithm can be seen to be quite local in nature when making decisions about the choice of empty entries to fill and choice of the color to fill in the chosen empty entry. Thus it is ideal for use in a dynamic situation where requests are online. LEGAL(i, j, k) is a predicate that is true when k is a valid color for entry (i, j).

Performance.

Theorem 1 GREEDY *is a 3-approximation algorithm.*

The interested reader can find the proof in [RRS96] that GREEDY is a 3-approximation algorithm. The crux of the proof depends on the observation that the decision by GREEDY to place color k in entry $L_{i,j}$ can at most affect the strategy of OPT in three ways – to

```
MATCH(L):
    for 1 ≤ k ≤ n do
        let G_k = (V, V', E_k) with
            V := {1, ..., n}, V' := {1', ..., n'}
            E_k := {(i, j') | L_{i,j} = 0 ∧ LEGAL(L, i, j, k)}
        M_k := MAXIMUM-MATCHING (G_k)
        for (i, j') ∈ M_k do
            L_{i,j} := k
    end for
```

Figure 7: The MATCH Algorithm

place k in some other empty entry in row i, or to place k in some other empty entry in column j, or to place some other color k' in $L_{i,j}$. Thus, in the worst-case, GREEDY can fill in m entries in the PLS when OPT could have filled in $3m$ entries.

It was also shown in [RRS96] by a probabilistic argument that any $n \times n$ PLS can always be filled to density one-half. This automatically gives us that GREEDY performs much better (i.e., is a $(2+\epsilon)$-approximation algorithm, for small ϵ) when the initial PLS is sparse (i.e., has only $o(n^2)$ entries filled).

Running Time. By preprocessing the given PLS, we can construct bit vectors (of length n) representing allowable colors for each row and column. Once we have this data structure, picking a legal color to fill in a 0 entry can be done in $O(n)$ time. Updating the data structures takes $O(1)$ time. Thus, the entire algorithm is very simple and can be implemented in $O(n^3)$ time.

Additional Heuristic. We also provide an additional heuristic (without analysis) which seems to improve the algorithm a little, as indicated by our experiments. This is also partially motivated by the concept of degree of freedom. Instead of examining the 0 entries in an arbitrary order, the algorithm chooses to examine them in the order of increasing degrees of freedom. This is done by computing the degree of freedom for each 0 entry in the PLS, sorting the 0 entries in ascending order, and using the sorted order to fill in a greedy manner. The time taken by this algorithm (GREEDY+) still remains $O(n^3)$.

2.4 Matching Algorithm

Algorithm Description. Unlike GREEDY the decision to fill in a 0 entry in the PLS with a particular entry is made in a global fashion in MATCH. The essence is to consider each color k of the PLS and do the following: construct a (bipartite) graph on $\{1,\ldots,n\} \cup \{1',\ldots,n'\}$ with an edge (i,j') whenever $L_{i,j}$ is empty and k is a legal color. We then obtain a maximum matching (a collection of non-intersecting edges) of this graph and the edges selected for matching dictate that those 0 entries should be filled with color k. The algorithm is presented in Figure 7.

Performance.

Theorem 2 MATCH *is a* 2*-approximation algorithm.*

The interested reader can find a proof in [RRS96] that MATCH is a 2-approximation algorithm. We give here a gist of the main idea involved. The argument is based on the observation that for each color k, the number of 0 entries OPT can fill with k is at most the cardinality of the maximum matching of the graph constructed for k. The decision by MATCH to place k in $L_{i,j}$ can at most affect the decision of OPT to place k' in $L_{i,j}$ and k elsewhere. Since MATCH decides according to the maximum matching, the number of entries it fills with color k is at least as many as that of OPT. Thus, in the worst-case, MATCH can fill in m entries in the PLS when OPT could have filled in $2m$ entries.

Running Time. For each color, we can build a n^2 node bipartite graph and run the maximum matching algorithm MAXIMUM-MATCHING of Hopcroft-Karp [HK73], which runs in $O(n^{2.5})$ time. Thus, the total algorithm can be implemented in $O(n^{3.5})$ time.

Additional Heuristic. We also provide an additional heuristic (without analysis) which seems to improve the algorithm a little. This is also motivated by the degree of freedom parameter. Instead of an arbitrary order to examine the colors, we examine in the order of increasing cardinality of matching. At each step of the algorithm, we select the color that has the least cardinality matching. The time taken by this algorithm (MATCH+), however, increases to $O(n^{4.5})$.

Another point in favor of our algorithms is the fact that the notion of row/column/color are all used in an interchangeable fashion. Hence, our algorithms can be adapted in a dynamic setting where, for instance, the preassigned entries are specified row by row, column by column, or color by color.

3 Experimental Results

To study the applicability of our approximation algorithms in practice, we performed simulation studies. We are mainly concerned with the quality of the output (i.e., the number of filled entries in the final PLS) and the time taken for execution. We then compare the performance against BRANCHBOUND (which is OPT, as far as the quality of the result is concerned). The simulations were conducted on 4×4 to 9×9 PLSs that were randomly filled with initial densities from 20% to 80%. The performance of various algorithms are shown in Figure 8. The table shows the output densities obtained and execution time (in ms) averaged over several trials (100 trials). The simulations were written in C++ and run on a Sparc-20. The results in the tables can be taken in at a glance by inspecting the graphs in Figure 9. From our results,

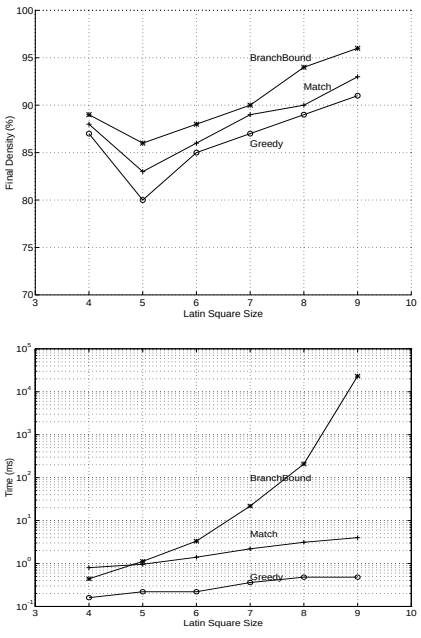

Figure 9: Performance of Algorithms when Initial Density = 50%

it is clear that the approximation algorithms with their heuristics

Algorithm		4 × 4 PLS				5 × 5 PLS				6 × 6 PLS			
	Initial%	20	40	60	80	20	40	60	80	20	40	60	80
GREEDY	Final%	96	87	85	83	91	85	86	86	90	87	85	85
	Time	0	0.08	0.08	0.12	0.08	0.08	0.32	0.12	0.24	0.08	0.12	0.12
GREEDY+	Final%	82	89	87	83	92	87	85	86	93	91	87	86
	Time	0.36	0.32	0.16	0.24	0.48	0.36	0.12	0.08	0.56	0.36	0.16	0.24
MATCH	Final%	100	87	87	83	94	91	87	86	98	91	86	85
	Time	1	0.52	0.76	0.36	1.24	1.08	1	0.68	1.88	1.64	1.4	1.08
MATCH+	Final%	100	96	87	83	100	90	87	86	100	92	86	85
	Time	1.36	1.44	1.04	0.6	3.24	2.72	2.2	1.2	6.28	5.4	4.32	2.56
BRANCHBOUND	Final%	100	100	87	83	100	93	87	86	100	95	88	86
	Time	2.52	1.12	0.32	0.16	9.84	2.64	0.68	0.12	75.04	14.68	2.04	0.16

Algorithm		7 × 7 PLS				8 × 8 PLS				9 × 9 PLS			
	Initial%	20	40	60	80	20	40	60	80	20	40	60	80
GREEDY	Final%	87	87	85	85	89	88	86	88	91	89	86	85
	Time	0.24	0.28	0.08	0	0.25	0.32	0.32	0.2	0.6	0	0.32	0.12
GREEDY+	Final%	90	90	87	85	89	91	88	88	91	92	89	85
	Time	0.72	0.6	0.24	0.32	1	0.76	0.44	0.28	1.8	1	0.96	0.2
MATCH	Final%	96	92	88	85	100	94	89	88	97	95	88	85
	Time	3	2.72	2.16	1.64	4	3.4	2.76	2.16	5.6	4.7	3.48	2.76
MATCH+	Final%	99	96	87	85	100	95	88	88	100	95	89	85
	Time	10.32	8.88	6.96	3.28	17	16.3	11.48	5.84	27.4	22.7	17.6	7.72
BRANCHBOUND	Final%	100	98	90	85	100	98	90	88	100	99	91	85
	Time	1227	153	7.04	0.4	589	12k	16.16	0.4	246k	401k	144.56	0.48

Figure 8: Experimental Evaluation of the Algorithms

perform extremely well in practice. As expected, BRANCHBOUND runs fast when the density is large (since there are less choices). This suggests a combined approach in practice – using the approximation algorithms at low densities and BRANCHBOUND at higher densities.

4 Conclusions

In this paper we present three algorithms to complete a Latin Router in which some of the entries may have been preassigned. Two of the algorithms are fast polynomial time approximation algorithms with provable worst-case bounds. The third is an exhaustive search-based algorithm that uses pruning to dynamically shrink the search-space. These algorithms have applications to the problem of lightpath assignment in wide area wavelength routing optical networks. We also present simulation results which show that on average the two approximation algorithms are near-optimal, in addition to being very fast.

Acknowledgements

We thank Madhav Marathe (Los Alamos), Ram Ramanathan (BBN Systems), and Suresh Subramaniam (U. of Washington/M.I.T. Lincoln Labs) for their suggestions. We also thank Chien Chen (Stevens Tech) for providing the C code for their implementation.

References

[BH92] R. A. Barry and P. A. Humblet. Bounds on the number of wavelengths needed in WDM networks. In *LEOS '92 Summer Topical Meeting Digest*, pp. 21–22, 1992.

[BH93] R. A. Barry and P. A. Humblet. Latin routers, design and implementation. *IEEE/OSA J. of Lightwave Technology*, pp. 891–899, 1993.

[CB95] C. Chen and S .Banerjee. Optical switch configuration and lightpath assignment in wavelength routing multihop lightwave networks. In *INFOCOM*, pp. 1300–1307, 1995.

[CGK92] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: An approach to high bandwidth optical WANs. *IEEE Trans. on Communication*, 40(7):1171–1182, 1992.

[CNW90] N. K. Cheung, K. Nosu, and G. Winzer, editors. *IEEE JSAC: Special Issue on Dense WDM Networks*, vol. 8, 1990.

[Col84] C. J. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8:25–30, 1984.

[Gre92] P. E. Green. *Fiber-Optic Networks*. Prentice-Hall, 1992.

[HK73] J. E. Hopcroft and R. M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. on Computing*, 2(4):225–231, 1973.

[IEE93] IEEE/OSA. *J. of Lightwave Technology, special issue on Broad-Band Optical Networks*, vol. 11, 1993.

[IK92] M. Irshid and M. Kavehrad. A wdm cross-connected star topology for multihop lightwave networks. *J. of Lightwave Technology*, pp. 828–835, June 1992.

[LL93] K.-C. Lee and V.O.K. Li. Routing and switching in a wavelength convertible optical network. In *INFOCOM*, pp. 578–585, 1993.

[Ram93] R. Ramaswami. Multi-wavelength lightwave networks for computer communication. *IEEE Communications Magazine*, 31(2):78–88, 1993.

[RRS96] S. Ravi Kumar, A. Russell, and R. Sundaram. Approximating latin square extensions. In *COCOON*, pp. 280–289, 1996.

[RS94] R. Ramaswami and K.N. Sivarajan. Optimal routing and wavelength assignment in all-optical networks. In *INFOCOM*, pp. 970–979, 1994.

[ZA94] Z. Zhang and A. Acampora. A heuristic wavelength assignment algorithm for multihop WDM networks with wavelength routing and wavelength reuse. In *INFOCOM*, pp. 534–543, June 1994.