Maximum Request Satisfaction in WDM Rings: Algorithms and Experiments

Evangelos Bampas, Aris Pagourtzis, and Katerina Potika

School of Electrical & Computer Engineering,
National Technical University of Athens
Zografou Campus, 15780, Athens, Greece
{ebamp, pagour, epotik}@cs.ntua.gr

Abstract

We study the problem of satisfying a maximum number of communication requests in all-optical WDM rings in which the number of available wavelengths per fiber is limited. We investigate two variations of the problem: with or without prior routing of requests. We consider a number of new and existing algorithmic approaches for these two variations. We perform an experimental comparison of the resulting algorithms, with respect to: (a) the number of satisfied requests and (b) the running time. We end up with interesting observations that reveal merits and deficiencies of the algorithms in hand:

- All algorithms almost always manage to satisfy many more requests than indicated by their worst-case analysis.
- Some algorithms are considerably faster than others with comparable request satisfaction performance. In fact, there are simple, fast algorithms that achieve a competent number of satisfied requests.

We anticipate that our results will prove useful in practice, especially in deciding which routing and wavelength assignment method to choose, taking into account the desired level of accuracy and the affordable time cost.

Keywords: WDM rings, path coloring, request satisfaction, wavelength assignment

1. Introduction

Wavelength Division Multiplexing (WDM) is a dominating technology in contemporary all-optical networking. It allows several connections to be established through the same fiber links, provided that each of the connections uses a different wavelength. A second restriction is imposed by the *wavelength continuity constraint*, which requires that a connection uses the same wavelength from one end to the other in order to avoid the use of wavelength converters which are costly or slow. In practice, the available bandwidth is limited to few dozens, or at most hundreds, wavelengths per fiber and the situation is not expected to change in the near future. It is therefore impossible to serve a large set of communication requests simultaneously.

An approach considered in several papers [12, 15, 6, 10, 11] is that of maximizing the number of requests that can be served at the same time given that the number of wavelengths is limited. Here we study the problem in rings and consider two variations: in the first the routing is pre-determined and only a color assignment is sought while in the second both a routing and a color assignment are sought. In graph-theoretic terms the problems are defined as follows:

Maximum Path Coloring Problem (MaxPC)

Input: a graph G, a set of paths $\mathcal{P}$ and a number of available colors w.

Feasible solution: a set of paths $\mathcal{P}' \subseteq \mathcal{P}$ that can be colored with w colors so that no overlapping paths are assigned the same color.

Goal: maximize $|\mathcal{P}'|$.

Maximum Routing and Path Coloring Problem (MaxRPC)

Input: a graph G, a set of pairs of nodes (requests) $\mathcal{R}$ and a number of available colors w.

Feasible solution: an assignment of paths to a subset of requests $\mathcal{R}' \subseteq \mathcal{R}$ and a coloring of these paths with w colors so that no overlapping paths are assigned the same color.

Goal: maximize $|\mathcal{R}'|$.

We focus on algorithms for the above problems in rings, which are a highly relevant network topology, as has become clear by their extensive use (e.g. SONET rings). Both MaxPC and MaxRPC are $\mathcal{NP}$-hard in rings [15, 11].

In this work we perform an experimental evaluation of a number of algorithmic approaches for these problems in rings. We first propose a new greedy heuristic for both problems which is very fast and easy to implement. We also develop improved variations of approximation algorithms that have been proposed in [15, 10, 11]. We end up with a bunch of seven algorithms for each problem. The comparative study of their performance, in terms of satisfied requests and running time, offers some quite interesting conclusions. All algorithms almost always manage to satisfy many more requests than indicated by their worst-case analysis. There are two simple algorithms that achieve satisfactory solutions very fast. The iterative algorithm usually finds largest solutions, despite the fact that it has the worst theoretical approximation ratio among the more sophisticated algorithms. One of our improved algorithms competes well with the iterative algorithm while being several times faster.

We expect that our results will prove useful in practice, especially in deciding which routing and wavelength assignment method to choose, taking into account the desired level of accuracy and the affordable time cost.

1.1 Related Work

MaxPC in chains is known as the ``k-coloring of intervals" problem which can be solved exactly [4]. For MaxRPC in rings, a $2/3$-approximation algorithm for the undirected problem and a $7/11$-approximation algorithm for the directed problem are given in [10]; for MaxPC in rings a $2/3$-approximation is described in [11]. Wan and Liu [15] present $(1-\frac{1}{e})$-approximation algorithms for MaxRPC in rings and for MaxPC in trees, as well as a constant approximation algorithm for MaxRPC in meshes. Their algorithms employ successive calls to algorithms that solve MaxRPC or MaxPC in instances with one available color (also known as the Maximum Edge-Disjoint Paths problem). Using the same technique, Erlebach and Jansen [7] provide a $(1-\frac{1}{e})$-approximation algorithm for MaxRPC in bounded-degree bidirected trees and a 0.451-approximation algorithm for general bidirected trees. The on-line version of MaxRPC has been studied in [2] where a general technique to obtain a $(\rho+1)$-competitive algorithm for arbitrary number of wavelengths from a ρ-competitive algorithm for one wavelength is presented. (While preparing the camera-ready version of this paper it was brought to our attention that Caragiannis [3] has very recently proven that MaxPC and MaxRPC in rings can be approximated within 3/4; he also shows a 0.7079 ratio for directed MaxRPC. Unfortunately we did not have time to include these algorithms in our comparison.)

A generalization of MaxPC to multi-fiber networks has been considered for rings [13] and trees [8], where efficient constant approximation algorithms have been proposed; the problem for general topologies has been studied in [14] and [1].

1.2 Technical Preliminaries

A *chain* is a graph that consists of a single path, while a *ring* is a graph that consists of a single cycle. If we remove an edge e from a ring we get a chain; we call such an edge *separation edge*.

Given a network $G=(V,E)$ and a set of requests we denote by n the number of nodes, and by m the number of requests (paths). For a set of paths $\mathcal{P}$ and an edge e we denote by $L(e,\mathcal{P})$ the *load* of edge e w.r.t. $\mathcal{P}$, that is the number of paths in $\mathcal{P}$ that use e.

A path which is colored with color c is called a *lonely path* if it is the only path which is colored with color c. A request is called a *lonely request* if it is routed and colored so that the corresponding path is a lonely path. Two requests are called

compatible (with each other) if they can be routed so that the corresponding paths are not overlapping. A request compatibility graph $H = (R, E)$ of an instance $(G, \mathcal{R}, w)$ is defined as follows. The set of nodes of H is the set of requests $\mathcal{R}$ and the set of edges E contains all pairs of compatible requests.

2. Algorithms for MaxPC (requests are pre-routed)

In this section we focus on the case in which requests are pre-routed, that is, we study the MaxPC problem. Recall that in this case an instance actually consists of a graph G, a set of paths $\mathcal{P}$ and a number of colors w and the goal is to color as many paths as possible using the given colors, without assigning the same color to overlapping paths.

The problem can be solved exactly in $O(n+w)$ time if the input graph is a chain, using the algorithm of Carlisle and Lloyd [4]. That algorithm has some useful properties. Let L be the maximum load of the input chain graph. The following holds:

Fact: *If $w \geq L$ the Carlisle-Lloyd algorithm colors all paths using exactly L colors. If $w < L$ the algorithm colors a maximum cardinality subset of paths of load exactly w.*

Many of the algorithms presented in the rest of the paper make use of the Carlisle-Lloyd algorithm in order to optimally color chain subinstances.

2.1 Shortest-First Algorithm

We first present a new, greedy algorithm for MaxPC in rings. This algorithm is easy to implement and very fast; nevertheless we will see that it is almost as competent as the more sophisticated algorithms that will follow.

Algorithm 1 MaxPC-*SF* (* Shortest-First *)
Input: a ring $G = (V, E)$, a set of paths $\mathcal{P}$ and a number of available colors w.
Output: coloring of a subset $\mathcal{P}' \subseteq \mathcal{P}$ with w colors.

1. sort paths in $\mathcal{P}$ in order of non-decreasing length;
2. **for** each path $p \in \mathcal{P}$ **do**
   assign to p the smallest color that is available on all edges of p (if such a color exists, otherwise do nothing);

We next show that this simple algorithm always achieves a solution of size at least one-third the size of an optimal solution.

Theorem 1. MaxPC-*SF* is a (1/3)-approximation algorithm for MaxPC in rings.

Proof. Let $\mathcal{P}^*$ be the set of paths colored by an optimal solution and $\mathcal{P}'$ be the set of

paths colored by MaxPC-*SF*. Let also $\mathcal{P}^*_i$ ($\mathcal{P}'_i$) be the subset of $\mathcal{P}^*$ ($\mathcal{P}'$ respectively) that consists of paths colored with the i-th color.

Let D be any set of non-overlapping paths on the ring. The key idea here is that a path $p \notin D$ that is colored by MaxPC-*SF* with color j may prevent, in subsequent steps of the algorithm, at most two paths in D from obtaining color j. The reason is that such paths are at least as long as p (since they are examined by MaxPC-*SF* later than p); therefore, there can be only two of them in D that overlap with p. Hence, if MaxPC-*SF* does not color any path in D, it must be:

$$|D| \leq 2|\mathcal{P}'_j| \qquad (1)$$

Because, otherwise MaxPC -*SF* would have assigned j to at least one path in D. Note that equation (1) holds for all j, $1 \leq j \leq w$.

Let $D_i = \mathcal{P}^*_i \setminus \mathcal{P}'$, that is, D_i consists of paths that were assigned color i in the optimal solution, but were not colored by MaxPC-*SF*. However D_i is also a set of non-overlapping paths and consequently from equation (1), for all i, $1 \leq i \leq w$:

$$|D_i| \leq 2 \min_{1 \leq j \leq w} |\mathcal{P}'_j| \leq 2\frac{|\mathcal{P}'|}{w} \qquad (2)$$

Let us now observe that $\mathcal{P}^* \subseteq \mathcal{P}' \cup \bigcup_{1 \leq i \leq w} D_i$. This implies:

$$|\mathcal{P}^*| \leq |\mathcal{P}'| + w(2\frac{|\mathcal{P}'|}{w}) = 3|\mathcal{P}'|. \qquad (3)$$

This completes the proof.

Equation (2) implies that MaxPC-*SF* behaves much better on the average. For example, if some color j has been used fewer than $|\mathcal{P}'|/kw$ times, for some $k > 1$ then (3) becomes $|\mathcal{P}^*| \leq (1+\frac{2}{k})|\mathcal{P}'|$; that is, the solution returned is near-optimal for large k. Indeed, we will see in Section 4 that MaxPC-*SF* usually achieves quite satisfactory solutions.

A simple implementation of the algorithm costs $O(nmw)$ time.

2.2 Combining Solutions

Algorithm MaxPC-*CombSol* uses two main techniques: the one colors a chain

instance and the other colors pairs of non-overlapping paths. The algorithm in fact combines the two solutions so as to retain some key properties of both. This algorithm is an improved version of the algorithm presented in [11]. The main improvement is an additional last step that takes care of remaining paths that possibly lie on edges where some color is still free.

Algorithm 2 MAXPC-*CombSol* (* Combine Solutions *)
Input: a ring $G = (V, E)$, a set of paths $\mathcal{P}$ and a number of available colors w.
Output: coloring of a subset $\mathcal{P}' \subseteq \mathcal{P}$ with w colors.

1. select as separation edge, an $e \in E$ with minimum load; partition path set $\mathcal{P}$ into two path sets $\mathcal{P}_e$ and $\mathcal{P}_c$, where $\mathcal{P}_e$ contains all paths in $\mathcal{P}$ that pass through e, and $\mathcal{P}_c$ contains the remaining paths;
2. call the Carlisle-Lloyd algorithm [4] for MAXPC in chains on input $(G - \{e\}, \mathcal{P}_c, w)$;
3. find a maximum matching M (of cardinality μ) in $H = (\mathcal{P}_c \cup \mathcal{P}_e, E')$, such that edge $(p, q) \in E'$ if $p \in \mathcal{P}_c$, $q \in \mathcal{P}_e$ and p, q do not overlap;
4. uncolor lonely paths
5. **while** there exists an edge e' in M and free colors remain **do**
   (a) color the two endpoints (paths) of e' with a free color; remove e' from M;
   (b) uncolor lonely paths;
6. **while** free colors remain **do** color an uncolored path in $\mathcal{P}$ with a free color;
   (* Coloring remaining paths if possible *)
7. **for** every color c **do**
   (a) find all uncolored paths that do not overlap with any path colored with c; let $\mathcal{P}_u$ be this set;
   (b) find a maximum subset of non overlapping paths of $\mathcal{P}_u$ and color them with c

It has been shown in [11] that the algorithm presented there achieves an approximation guarantee of 2/3. Consequently, this holds for MaxPC-*CombSol* as well, since the main difference of the two algorithms is the addition of the `Coloring remaining paths' step which may only augment the solution achieved by the previous steps.

Steps 7 and 8 of Algorithm 2 cost $O(nmw)$ time. Therefore, the time complexity of Algorithm MaxPC-*CombSol* is $O(nmw + m^2)$, where $O(m^2)$ is the complexity of the bipartite matching computation, using an algorithm of Ma and Spinrad [9].

Algorithm MaxPC-*CombSol-all*. The selection of the separation edge may be crucial for the average performance of the algorithm. Therefore, we will consider a new version of the algorithm which consists of n executions of MaxPC-*CombSol*, each time with a different separation edge. Consequently, the complexity of MaxPC-*CombSol-all* is $O(nB)$, where B is the complexity of MaxPC-*CombSol*.

2.3 Selecting the Best Solution

The Algorithm MaxPC-*BestSol* solves each instance of the problem with two independent procedures, called Chain Step and Matching Step, and merely chooses the best solution between the solutions of these procedures. The Chain Step performs the same actions as Steps 1 and 2 of Algorithm 2. In addition, if k free colors remain after executing these steps, they are used to color k paths in $\mathcal{P}_c$. The Matching Step

performs the same actions as Steps 3 and 5(a) of Algorithm 2.

It is to be noted that MaxPC-*BestSol* achieves the same worst-case approximation ratio as the more involved algorithm MaxPC-*CombSol*.

Theorem 2. MaxPC-*BestSol* is a (2/3)-approximation algorithm for MaxPC in rings.

The complexity of the algorithm is determined by the bipartite matching computation which can be done in $O(m^2)$ time, using an algorithm of Ma and Spinrad [9].

Algorithm MaxPC-*Chain*. The Chain Step of MaxPC-*BestSol* can be used as an algorithm on its own. Moreover, it can be shown [11] that it achieves an approximation guarantee of $1/2$, since $OPT \le SOL_c + w$ (as discussed above) and $SOL_c \ge w$ (each color is used at least once, or all paths are colored). Its time complexity is $O(n+m)$ [11]. We will refer to this algorithm as MaxPC-*Chain*.

Algorithm MaxPC-*BestSol-all*. The selection of the separation edge e may again play an important role on the average performance of the algorithm, although it does not affect the worst-case approximation ratio. Therefore, we will also evaluate a new version of the algorithm, called MaxPC-*BestSol-all* which consists of n executions of MaxPC-*BestSol*, each time with a different separation edge. Clearly, the complexity of MaxPC-*BestSol-all* is $O(nA)$, where A is the complexity of MaxPC-*BestSol*. Therefore, it takes $O(nm^2)$ time.

2.4 Iterative Algorithm

Algorithm MaxPC-*Iter* was proposed by Wan and Liu [15] and works as follows. Given a set of paths $\mathcal{P}$ and w available colors it examines colors one-by-one. For each color c, it computes a maximum subset $\mathcal{S}$ of non-overlapping paths. To achieve this, for each path $p \in \mathcal{P}$ it determines a maximum subset $\mathcal{S}_p$ of $\mathcal{P}$ that can be colored with the same color as p (using e.g. an algorithm for the well known Activity Selection Problem [5, pp. 329-333]), and picks the largest such subset. It then colors paths in $\mathcal{S}$ with color c, removes $\mathcal{S}$ from $\mathcal{P}$ and proceeds with the next color.

Algorithm MaxPC-*Iter* achieves an approximation guarantee of $1-(1-\frac{1}{w})^w > 1-\frac{1}{e} \approx 0.632$; the ratio $1-(1-\frac{1}{w})^w$ is slightly worse (for $w>10$) than the approximation guarantee of $2/3$ achieved by algorithms MaxPC-*BestSol*, MaxPC-*BestSol-all*, MaxPC-*CombSol*, and MaxPC-*CombSol-all*. The complexity of this algorithm is $O(wm^2 \log m)$.

3. Algorithms for MaxRPC (Requests are not pre-routed)

3.1 Shortest-First Algorithm

We present MaxRPC-*SF*, which is a heuristic analogous to the simple heuristic for MaxPC; the difference is that it takes care of the routing too.

Algorithm 3 MAXRPC-*SF* (* Shortest-First *)
Input: a ring $G = (V, E)$, a set of requests $\mathcal{R}$ and a number of available colors w.
Output: routing and coloring of a subset $\mathcal{R}' \subseteq \mathcal{R}$ with w colors.

1. perform shortest-path routing on $\mathcal{R}$, thus obtaining a set of paths $\mathcal{P}$;
2. sort paths in $\mathcal{P}$ in order of non-decreasing length;
3. **for** each path $p \in \mathcal{P}$ **do**
   assign to p the smallest color that is available on all edges of p (if such a color exists, otherwise do nothing);

As for MaxPC-*SF*, it can be shown that MaxRPC-*SF* is a (1/3)-approximation algorithm and that a simple implementation of the algorithm costs $O(nmw)$ time.

3.2 Combining Solutions

Our second algorithm for MaxRPC is the analogue of MaxPC-*CombSol* for the MaxRPC problem.

Algorithm 4 MAXRPC-*CombSol*
Input: a ring $G = (V, E)$, a set of requests $\mathcal{R}$ and a number of available colors w.
Output: routing and coloring of a subset $\mathcal{R}' \subseteq \mathcal{R}$ with w colors.

1. select as separation edge, an $e \in E$ with minimum load (w.r.t. shortest path routing); route requests so that paths avoid e; let $\mathcal{P}_e$ denote the resulting set of paths;
2. call the Carlisle-Lloyd algorithm [4] for MAXPC in chains on input $(G - \{e\}, \mathcal{P}_e, w)$;
3. find a maximum matching M (of cardinality μ) in H, the request compatibility graph of $\mathcal{R}$;
4. uncolor lonely requests;
5. **while** there exists an edge e' in M with at least one endpoint uncolored and free colors remain **do**
   (a) color the two endpoints (requests) of e' with a free color, route the requests accordingly; remove e' from M;
   (b) uncolor lonely requests;
6. **while** free colors remain **do**
   select an uncolored request in $\mathcal{R}$, route it using the shortest path and color it with a free color
7. **for** every color c **do**
   find all uncolored requests that can be routed without overlapping with any path colored with c; route them accordingly; let $\mathcal{P}_u$ be the resulting set of paths; find a maximum subset of non overlapping paths of $\mathcal{P}_u$ and color them with c

It can be shown that MaxRPC-*CombSol* returns a solution which is at least as large as the solution returned by a (2/3)-approximation algorithm for MaxRPC in rings that was presented in [10] (we will also implement that algorithm under the name MaxRPC-*BestSol*; see next subsection). Therefore, MaxRPC-*CombSol* is a (2/3)-approximation algorithm.

Steps 5 and 6 of Algorithm 4 cost $O(nmw)$ time. Therefore, the time complexity of

Algorithm MaxRPC-*CombSol* is $O(nmw + m^3)$, where $O(m^3)$ is the time complexity of the maximum matching computation.

As before, we will also consider an algorithm consisting of n calls of MaxRPC-*CombSol*, each with a different separation edge; we call this algorithm MaxRPC-*CombSol-all*. The complexity of MaxRPC-*CombSol-all* is $O(n^2mw + nm^3)$.

3.3 Selecting the Best Solution

Algorithm MaxRPC-*BestSol* was presented in [10]. The Chain Step is the same as Steps 1 and 2 of Algorithm 4. The Matching Step is the same as Steps 3 and 5(a) of Algortihm 4. As for MaxPC-*BestSol*, we independently call the Chain Step and the Matching Step and choose the best solution between the two solutions.

Similar to MaxPC-*BestSol* we consider the variation of MaxRPC-*BestSol* consisting of n calls of MaxRPC-*BestSol*, each with a different separation edge; we call this algorithm MaxRPC-*BestSol-all*. We also consider the chain step of MaxRPC-*BestSol* as a separate algorithm, called MaxRPC-*Chain*.

As shown in [10], MaxPC-*BestSol* is a $(2/3)$-approximation algorithm. As a corollary, MaxRPC-*BestSol-all* is also a $(2/3)$-approximation algorithm. It was also shown in [10] that MaxRPC-*Chain* is a $(1/2)$-approximation algorithm.

The complexity of MaxRPC-*BestSol* is determined by the maximum matching computation which can be done in $O(m^3)$ time. The complexity of MaxRPC-*BestSol-all* is $O(nA)$, where A is the complexity of MaxRPC-*BestSol*. Therefore, it takes $O(nm^3)$ time.

3.4 Iterative Algorithm

Wan and Liu [15] have also proposed an algorithm, which we will refer to as MaxRPC-*Iter*, for MaxRPC in rings. The algorithm works similarly to MaxPC-*Iter*, except that for each color c and for each request r it examines two paths: one corresponds to clockwise routing of r, the other corresponds to counter-clockwise routing of r. In each case, every other request is routed so as to avoid overlapping with the path assigned to r or it is ignored if no such routing exists. After considering all routings obtained as above, the one that routes a maximum subset $\mathcal{S}$ of requests is chosen and the corresponding paths are colored with the current color c. The requests in $\mathcal{S}$ are then removed from the input and the algorithm proceeds with the next color. Similar to MaxPC -Iter, the iterative algorithm for MaxRPC in rings achieves an approximation ratio of $1-(1-\frac{1}{w})^w \geq 1-\frac{1}{e} \approx 0.632$ and its time complexity is

$O(wm^2 \log m)$.

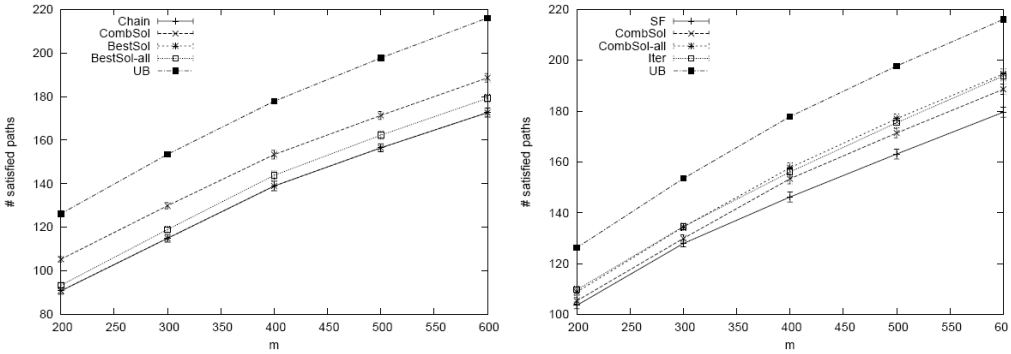

Fig. 1. Performance of algorithms for MAXPC in terms of the number of satisfied paths. $n = 100$, $w = 40$, $m = 200 - 600$, uniform distribution. *Left*: MAXPC-Chain, MAXPC-CombSol, MAXPC-BestSol, MAXPC-BestSol-all, *Right*: MAXPC-SF, MAXPC-CombSol, MAXPC-CombSol-all, MAXPC-Iter.

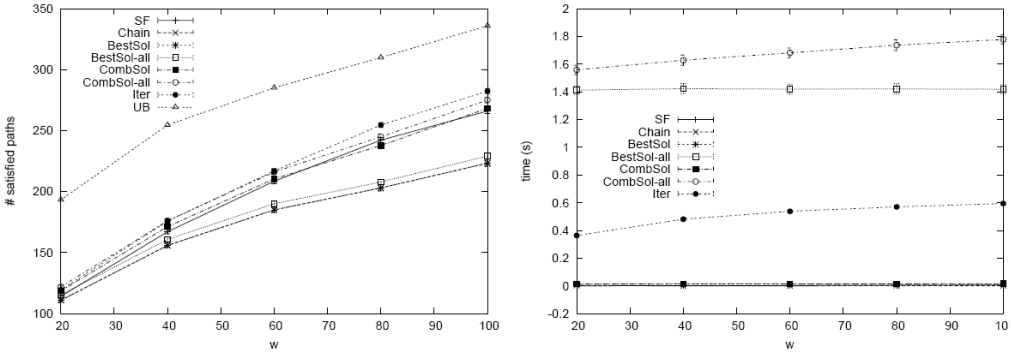

Fig. 2. Performance of algorithms for MAXPC in terms of the number of satisfied paths (*left*) and time (*right*). $n = 100$, $m = 500$, $w = 20 - 100$, Gaussian distribution.

4. Experimental Results

We implemented all algorithms in C++, making use of the LEDA class library of efficient data types and algorithms. All source files were compiled with the Borland C++ 5.5 for Win32 compiler, set to generate fastest possible code. We relied on LEDA routines and classes for graph, array, list and priority queue operations including sorting and finding matchings in general graphs. The experiments were run on a Pentium 4 3.2GHz with 512MB of memory.

For each combination of number of nodes (n), number of paths/requests (m) and number of available wavelengths (w), we randomly generated two sets of 60 instances each. For the first set, we assumed uniform distribution of the endpoints of the paths/requests over the nodes of the ring. For the second set, we assumed normal

distribution with standard deviation approximately $\sigma = \frac{2n}{15}$.

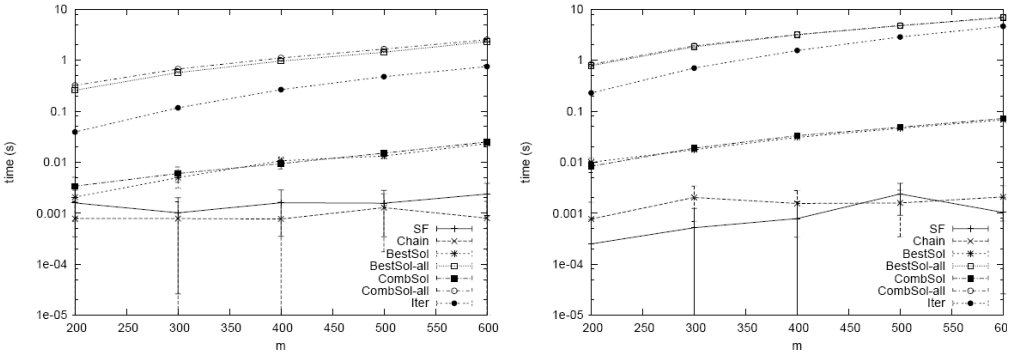

Fig. 3. Time performance of algorithms for MaxPC (*left*) and MaxRPC (*right*). $n = 100$, $w = 40$, $m = 200 - 600$, uniform distribution.

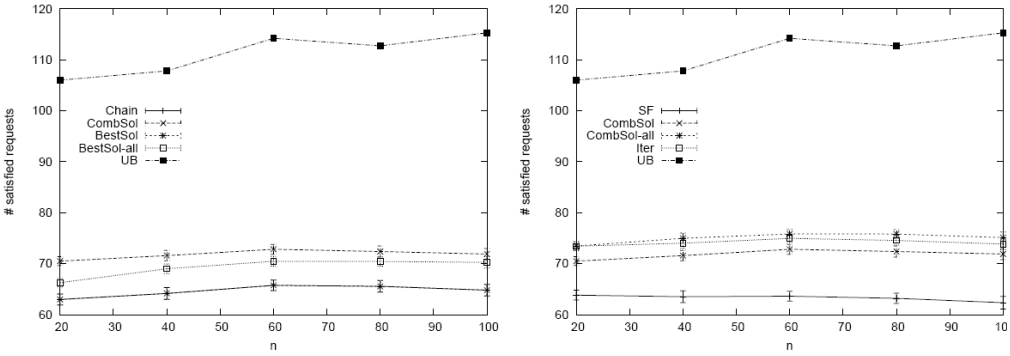

Fig. 4. Performance of algorithms for MaxRPC in terms of the number of satisfied requests. $m = 120$, $w = 16$, $n = 20 - 100$, Gaussian distribution. *Left*: MaxRPC-Chain, MaxRPC-CombSol, MaxRPC-BestSol, MaxRPC-BestSol-all, *Right*: MaxRPC-SF, MaxRPC-CombSol, MaxRPC-CombSol-all, MaxRPC-Iter.

We executed each algorithm on these sets of instances and measured the average execution time and the average number of satisfied paths/requests. Furthermore, for each of these values we calculated a 95 percent confidence interval which is shown on the plots. However, in some cases, this interval is quite small and may not be clearly visible. In the plots where we show the number of satisfied paths/requests, we include a computed upper bound for the sake of comparison.

Note that execution times were measured using the *timer* class of the LEDA package, which does not provide for measuring exact processor time. However, we ran the experiments on a dedicated machine in order to keep background processes at a minimum.

4.1 Computing an Upper Bound on OPT

In order to obtain an estimation of the performance of our algorithms we propose an efficient way to compute an upper bound on the value of an optimal solution. We denote by *length(p)* the number of edges that path p uses. If requests are given instead of paths (MaxRPC problem) we consider for each request $r = (i, j)$ the shortest path p between nodes i and j. We index all paths in non-decreasing order of their *length*. It can be easily proven that the following lemma holds:

Lemma 1. Let k be the smallest number such that $\sum_{i=1}^{k+1} length(p_i) > nw$. Then k is an upper bound on the number of paths (requests) that can be satisfied with w colors.

We could derive an alternative upper bound, using the solution of MaxPC-*Chain*, since any solution can color at most that many paths as those colored by MaxPC-*Chain* plus at most w paths (requests) passing through edge e (respectively, routed using edge e). However, we observed that this upper bound is similar to the one proposed in Lemma 1.

4.2 Discussion

A first observation is that all algorithms perform considerably better than their theoretical guarantee. Indeed, we have included a curve showing the computed upper bound (UB) in our figures and it turns out that all algorithms manage to satisfy a good fraction of an optimal solution, very often better than the theoretically predicted. Taking also into account that the upper bound used may be overestimated it is possible that the actual performance of the algorithms is even better.

The experimental comparison of the algorithms shows that each algorithm for MaxPC has similar behaviour on instances of similar size with the corresponding algorithm for MaxRPC. Note however that the latter is slightly slower (since routing is involved) but usually achieves a higher number of satisfied requests, probably due to the freedom of choosing a more adequate routing for each request.

Clearly, the best algorithms in terms of number of satisfied requests are Max(R)PC-*Iter* and Max(R)PC-*CombSol-all*; however Max(R)PC-*CombSol-all* has the worst time complexity and Max(R)PC-*Iter*, while faster than Max(R)PC-*CombSol-all*, is still quite slow compared to Max(R)PC-*SF*, Max(R)PC-*Chain*, Max(R)PC-*BestSol* and Max(R)PC-*CombSol*; among the latter, Max(R)PC-*CombSol* is competitive. In fact, in terms of performance per time unit spent Max(R)PC-*CombSol* is clearly the best of all algorithms. Max(R)PC-*Chain* seems to be even better with respect to performance/time ratio, providing solutions that can be considered satisfactory very fast; however its performance decreases linearly as the number of wavelengths increases (see Figure 2). In contrast, algorithm Max(R)PC-*SF*, which is also extremely fast, remains relatively competitive even for large number of wavelengths. Finally, Max(R)PC-*BestSol* has practically the same performance as the much faster

Max(R)PC-*Chain* and Max(R)PC-*BestSol-all* has poor performance while being the most time-consuming algorithm.

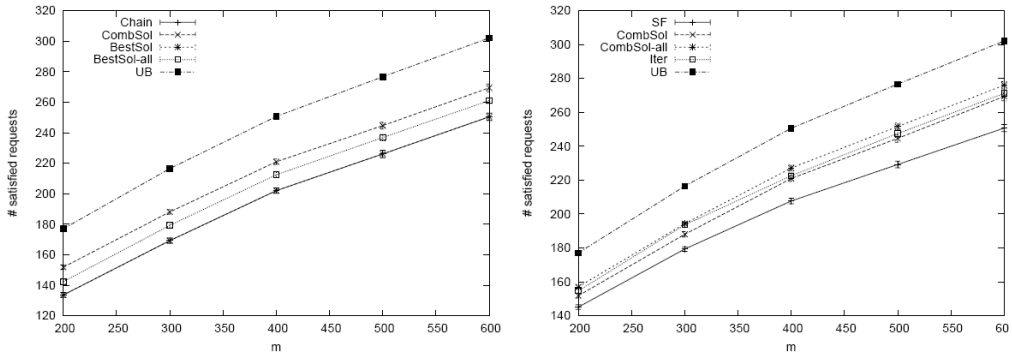

Fig. 5. Performance of algorithms for MAXRPC in terms of the number of satisfied requests. $n = 100$, $w = 40$, $m = 200 - 600$, uniform distribution. *Left*: MAXRPC-Chain, MAXRPC-CombSol, MAXRPC-BestSol, MAXRPC-BestSol-all, *Right*: MAXRPC-SF, MAXRPC-CombSol, MAXRPC-CombSol-all, MAXRPC-Iter.

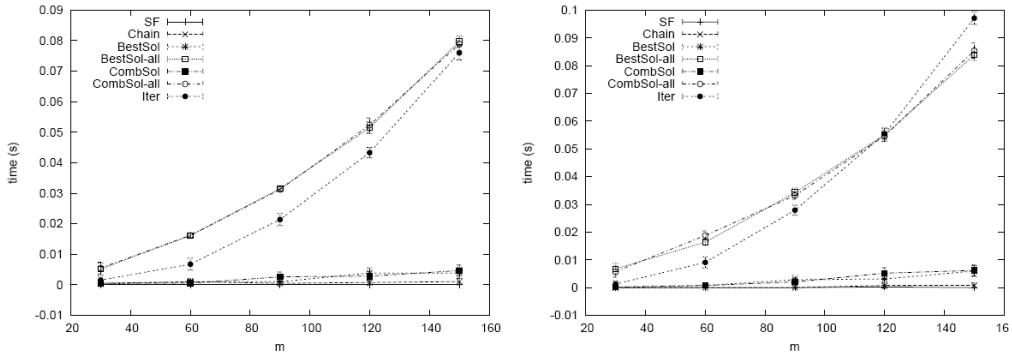

Fig. 6. Time performance of algorithms for MAXRPC. $n = 16$, $w = 8$, $m = 30-150$. *Left*: uniform distribution, *Right*: Gaussian distribution.

It is rather surprising that for large values of w Max(R)PC-*Iter* exhibits a clear superiority although it has the theoretically worst approximation ratio among all the algorithms (with the exception of Max(R)PC-*Chain* and Max(R)PC-*SF*). Figure 2 shows that the superiority of Max(R)PC-*Iter* increases as w increases, but its time complexity depends linearly on w while all other algorithms are practically independent from w (probably with the exception of Max(R)PC-*CombSol-all*). Besides, Max(R)PC-*Iter* exhibits a super-quadratic dependence on the number of requests m, as shown in Figures 3 and 6. A similar dependence on m characterizes also Max(R)PC-*CombSol-all* and Max(R)PC-*BestSol-all*, while all other algorithms seem to have a sub-quadratic dependence on m.

Finally, Figure 6 shows that the time complexity of Max(R)PC-*Iter* has a super-

quadratic dependence on m (the number of requests), the time complexity of Max(R)PC-*Chain* and Max(R)PC-*SF* is almost linear on m and the time complexities of the remaining algorithms are quadratic on m. These differences may become crucial for very large numbers of requests.

5. Conclusions

We have presented various algorithms for the MaxPC and MaxRPC problems in rings and have demonstrated results concerning the achieved practical performance.

To evaluate the experimental results we take into consideration the number of satisfied requests as well as the time performance. Taking into account both measures we first remark that Max(R)PC-*CombSol* is probably the algorithm of choice for practical purposes, since it achieves one of the best performances, in respect to the number of satisfied requests, and at the same time its time requirements are relatively low. Of course Max(R)PC-*Iter* and Max(R)PC-*CombSol-all* produce better solutions than Max(R)PC-*CombSol*, especially as w increases. Undoubtedly Max(R)PC-*Iter* performs better than all other algorithms for very large w but it gets much slower at the same time, because it depends linearly on w.

From a practical point of view, we can assume that the number of nodes and the number of wavelengths are fixed, thus it is important to consider the behavior of algorithms with respect to the number of requests m. The superiority of Max(R)PC-*CombSol* is even more clear in this case considering its time performance. The next best choice seems to be Max(R)PC-*Iter* which exhibits a ``middle'' time performance.

Max(R)PC-*BestSol* and Max(R)PC-*BestSol-all* are not at all competitive because they fail to provide better solutions than much faster algorithms like Max(R)PC-*SF*, Max(R)PC-*Chain* and Max(R)PC-*CombSol*. The new greedy heuristic Max(R)PC-*SF* is a decent choice whenever time is crucial, since it achieves relatively large solutions while being one of the fastest of algorithms. Table 1 summarizes the above observations.

Algorithm	Request Satisfaction	Time Consumption
MAX(R)PC-*SF*	2	5
MAX(R)PC-*Chain* [4]	1	5
MAX(R)PC-*BestSol* [10]	1	5
MAX(R)PC-*BestSol-all*	2	1
MAX(R)PC-*CombSol* [11]	4	5
MAX(R)PC-*CombSol-all*	5	1
MAX(R)PC-*Iter* [15]	5	3

Table 1. Ranking of the algorithms: 1=poor, 5=excellent.

Directions for further research include fine-tuning of some parts of the algorithms. For example, it would make sense to set a threshold on the number of iterations of Max(R)PC-*Iter* and combine it with some other strategy for the remaining colors; this

could result in more acceptable running times even for large values of w.

Acknowledgement: Research supported in part by the General Secretariat of Research and Technology of Greece through ΠΕΝΕΔ 2003 programme, contract nr. 03ΕΔ285, co-funded by the European Social Fund (75%) and national resources (25%) and also by the National Technical University of Athens through "Protagoras" grant.

References

1. M. Andrews and L. Zhang. Complexity of wavelength assignment in optical network optimization. In Proceedings of the 25nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2006), 2006.
2. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Ros´en. On-line competitive algorithms for call admission in optical networks. Algorithmica, 31(1):29–43, 2001.
3. I. Caragiannis. Wavelength management in WDM rings to maximize the number of connections. In Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science (STACS 07), Lecture Notes on Computer Science. Springer, 2007.
4. M. Carlisle and E. Lloyd. On the k-coloring of intervals. Discrete Applied Mathematics, 59:225–235, 1995.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. mcgraw, second edition, 2001.
6. T. Erlebach and K. Jansen. Maximizing the number of connections in optical tree networks. In Proceedings of the 9th Annual International Symposium on Algorithms and Computation (ISAAC '98), LNCS 1533, pages 179–188, 1998.
7. T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. SIAM Journal on Discrete Mathematics, 14(3):326–355, 2001.
8. T. Erlebach, A. Pagourtzis, K. Potika, and S. Stefanakos. Resource allocation problems in multifiber wdm tree networks. In Proc. of the 29th Workshop on Graph Theoretic Concepts in Computer Science, LNCS 2880, pages 218–229. Springer-Verlag, 2003.
9. T.-H. Ma and J. Spinrad. Avoiding matrix multiplication. In 16th Workshop on Graph Theory, LNCS vol. 484, LNCS 484, pages 61–71, 1990.
10. C. Nomikos, A. Pagourtzis, and S. Zachos. Minimizing request blocking in all-optical rings. In Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), 2003.
11. C. Nomikos, A. Pagourtzis, and S. Zachos. Satisfying a maximum number of pre-routed requests in all-optical rings. Computer Networks: The International Journal of Computer and Telecommunications Networking, 42(1):55–63, 2003.

12. C. Nomikos and S. Zachos. Coloring a maximum number of paths in a graph. Presented at the ICALP'97 Workshop on Algorithmic Aspects of Communication (July 11–12, 1997, Bologna, Italy), 1997.
13. K. Potika. Maximizing the number of connections in multifiber wdm chain, ring and star networks. In Proc. of NETWORKING 2005, LNCS 3462, pages 1465–1470, Springer-Verlag, 2005.
14. M. Saad and Z. Luo. On the routing and wavelength assignement in multifiber WDM networks. In Proceedings of Globecom 2002), 2002.
15. P.-J.Wan and L. Liu. Maximal throughput in wavelength-routed optical networks. In Multichannel Optical Networks: Theory and Practice, volume 46 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 15–26. AMS, 1998.