Published in *Workshop on Heuristic Methods for the Design, Deployment, and Reliability of Networks and Network Applications (HEUNET 2008)*

A Multilevel Approach for the Routing and Wavelength Assignment Problem

Thomas Fischer
Distributed Algorithms Group
University of Kaiserslautern
Germany

Kerstin Bauer
Embedded Systems Group
University of Kaiserslautern
Germany

Peter Merz
Distributed Algorithms Group
University of Kaiserslautern
Germany

Abstract

In this paper we present a multilevel approach for the static Routing and Wavelength Assignment (RWA) problem. The RWA deals with the problem of assigning paths and wavelengths to requests in optical communication networks. The multilevel approach is a general solution strategy involving stepwise coarsening the original problem instance, solving a simplified instance and expanding the solution back to the original size.

We propose both a multilevel-inspired construction heuristic and a multilevel algorithm using iterated local search for refinement. These algorithms significantly improve previous approaches regarding time consumption and solution quality for large instances.

1. Introduction

The NP-complete [2] *Routing and Wavelength Assignment* problem (RWA) deals with *Wavelength Division Multiplexed* (WDM) optical networks, where communication requests between nodes are routed on optical fiber links.

Given is a graph $G(V,E,W)$ with nodes V, arcs E and wavelengths W. In the physical network, each edge $e \in E$ represents an optical fiber link where each wavelength $\lambda \in W$ is eligible. A request $r = (u^r, v^r, d^r) \in R$ consists of the nodes u^r and v^r and a demand of $d^r \in \mathbb{N}^+$ many units. For each unit of demand, a lightpath (optical path created by allocating the same wavelength throughout a path of links) between these two nodes has to be established (*wavelength continuity constraint*, see Fig. 1). The *wavelength conflict constraint* states that each wavelength on a physical link may be used by at most one lightpath. The RWA can either be defined as a *static* or a *dynamic* problem and different types of *cost functions* can be considered. In this paper we focus on the static RWA, where a set of given requests has to be routed minimizing the number of wavelengths needed.

In [5] a Linear Programming (LP) approach is proposed, where the RWA is represented as a *multicommodity network flow* problem with additional constraints. In [6] a memetic algorithm including a mutation operator, a recombination operator and two local search operators is considered. An alternative approach is splitting the problem into the two subproblems of routing and wavelength assignment. An overview on this approach is provided in [3, 9]. A proposal for a construction algorithm is given by Skorin-Kapov [7]. Requests are processed iteratively by assigning each unit of demand a wavelength and a path depending on bin-packing inspired strategies. Experimental results imply the best strategy is *Best Fit Decreasing* (BFD), where requests are sorted non-increasingly by length (shortest paths in G) and routed in the wavelength with the shortest feasible path.

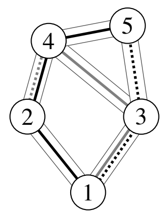

Figure 1: Example for the RWA, using two wavelengths (black, gray) to route requests $\{(1,5,2),(1,4,1),(2,4,1)\}$. The first request uses the black dotted lightpath $\langle 1,3,5\rangle$ and black solid path $\langle 1,2,4,5\rangle$, the second request the gray solid path $\langle 1,3,4\rangle$. The last request uses the gray dotted path $\langle 2,4\rangle$.

In [1] we presented an iterated local search (ILS) algorithm for the static RWA. The ILS consists of a local search (LS), which move requests from less used wavelength to highly used wavelengths with the intention to clear already sparse wavelengths, and a mutation operator, which reroutes path by forcefully clearing the designated links in the target wavelength first. Based on these findings, we developed a multilevel refinement algorithm as promoted by Walshaw [8]. In a multilevel approach, a problem instance is coarsened in multiple steps until a solution can easily be found for this instance. Stepwise extension and refinement of intermediate solutions (e. g. by local search) lead to a solution for the original instance. As the refinement operates on smaller instances (except for the last step), this approach is expected to find acceptable solutions faster than an algorithm operating on the original instance only.

In Sec. 2 we describe both our new construction heuristic and the multilevel ILS approach. Section 3 presents the experimental setup and our results.

2. Algorithms

For the ILS as described in [1], the following implementation-independent estimations on the run-time can be made. We define $n = |V|$ and $D = \sum_{r \in R} d_r$ as the sum of demand of all requests. In the local search operation, for each unit of demand ($\mathcal{O}(D)$) and each wavelength ($\mathcal{O}(D)$) a shortest path is determined ($\mathcal{O}(n^2)$) resulting in a time complexity of $\mathcal{O}(D^2 n^2)$. Each mutation has complexity $\mathcal{O}(D^2 n^3)$, as it reallocates $\mathcal{O}(n)$ paths requiring to search in each wavelength a shortest path. The BFD construction costs $\mathcal{O}(D^2 n^2)$ performing a shortest path search for each unit of demand in each available wavelength.

The impact of D on the time complexity of all algorithms above motivate a multilevel approach using the requests' demand. We define a coarsening step by dividing the demand of each request by the same scaling factor. Expanding a solution to the coarsened instance, the set of paths for each demand in the coarsened instance's solution is copied multiple times (detemined by the scaling factor) in the expanded instance. The strength of the scaling operation is motivated by the experimental findings from [1]: For small instances with $D < 10^4$ the ILS was able to produce (near-)optimal solutions in short time. Therefore, the Scaling Constructor (SC) and the multilevel (ML) approach discussed below will scale down a problem instance by a factor of a^k so that the sum of its demand is below this threshold. The ML performs k expansion operations each multiplying the demand with $a \in \mathbb{N}$, whereas the SC expands to the original instance directly. Scaling step a has to be supplied, k will be derived from the experimental setup. Both approaches are expected to considerably reduce the run-time due to the reduced D.

2.1. Scaling Constructor

The SC (Fig. 3c) is a simplified multilevel algorithm as it performs only one level of coarsening and no refinement is applied to the intermediate solution. The algorithm starts with scaling the demand of the original instance's requests R_{orig} down to R_{scaled} using a scaling factor $f = a^k$ to get $D < 10^4$ (Fig. 3c, line 2). For the scaled instance, an initial solution is constructed using the BFD algorithm. When expanding the scaled instance's solution s_{scaled} to a solution for the original instance (Fig. 3b), paths from s_{scaled} are copied multiple times to the original instance's solution s_{orig}, but a conflict-free wavelength has to be assigned to each path copy. Here, we use a rather straightforward approach and leave finding better assignments to an improvement algorithm applied later. Assuming the number of used wavelengths in s_{scaled} is $|W(s_{\text{scaled}})|$ and for a given request r the j-th path's wavelength is $\lambda(s_{\text{scaled}}, r, j)$, the wavelength λ' assigned to the i-th copy for this path in s_{orig} is

$$\lambda' = i \cdot |W(s_{\text{scaled}})| + \lambda(s_{\text{scaled}}, r, j) \qquad (1)$$

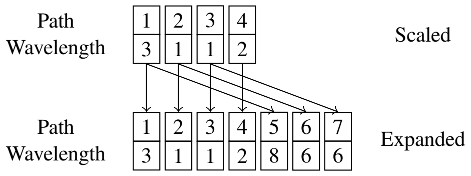

Figure 2: Example for the multiplication of paths for a single request during an expansion step. In this example, $d^r_{\text{scaled}} = 4$, $d^r_{\text{expanded}} = 7$, and $|W(s_{\text{scaled}})| = 5$.

For each unit of demand $1 \leq d \leq d^r_{\text{orig}}$ quotient and remainder of d^r_{scaled}/d determine the copy's number i and path index j, respectively (Fig. 3b, line 6). SCALEREQUESTS guarantees that there exists at least one path in s_{scaled} for each $r \in R_{\text{orig}}$.

2.2. Multilevel Approach

Compared to the SC, the ML is more sophisticated as it performs several steps of expanding, each refining its intermediate solution using ILS. Initially, one large coarsening step is performed scaling the original instance to an instance with $D < 10^4$ by a^k, where a is the scaling factor applied in each iteration step and $k = \lceil \log_a \frac{D}{10000} \rceil$ the number of iterations with a scaled instance. Starting from the deepest coarsening step, the multilevel algorithms iterates through the expansion steps until the original instance is restored. In each iteration $0 \leq i \leq k$, the original requests R_{orig} are scaled to R_{new} by the iteration's scaling factor $f = a^{k-i}$. In the initial step, the solution is constructed using the BFD algorithm, all later steps create a new solution based on a scaled version of the previous step's solution. Then, the ILS algorithm described in [1] is applied to this solution until no improvement can be found for c iterations (convergence criterion) or a global time limit t is reached. For the last iteration operating on the original instance only the global time limit t is used. We limit the ILS for all but the last iteration as we are not interested in optimal solutions for the scaled instances, but to get acceptable solutions for later expansion steps in short time. The global time limit is instance-dependent and must be provided *a priori*; we chose the time limits to be reasonable allowing the algorithms to show some convergence.

3. Experimental Setup

Benchmark instances for the RWA based on the SNDlib library [4] were presented in [1]. Here, we add new instances with higher demand D (Tab. 1). Whereas scaled variants of `atlanta` and `france` were used before, in this paper we use the original size. Instance `janos-us-ca` was used both in a scaled (factor 10, marked with '▼', to run

```
function ScaleRequests(Requests $R$, scaling factor $f$)
    $R_{\text{scaled}} = R$
    for all $r \in R$ do
        $d^r_{\text{scaled}} \leftarrow \left\lceil \frac{d^r}{f} \right\rceil$
    return $R_{\text{scaled}}$
```

(a) Function ScaleRequests

```
function ScaleSolution(Solution $s_{\text{old}}$, Requests $R_{\text{old}} = \{(u^r, v^r, d^r_{\text{old}})\}$, Requests $R_{\text{new}} = \{(u^r, v^r, d^r_{\text{new}})\}$)
    $s_{\text{new}} \leftarrow \emptyset$
    for all $r \in R_{\text{new}}$ do
        for $d \leftarrow 1, \ldots, d^r_{\text{new}}$ do
            $p \leftarrow \text{Path}(s_{\text{old}}, r, d \bmod d^r_{\text{old}})$
            $\lambda \leftarrow \left\lfloor \frac{d}{d^r_{\text{old}}} \right\rfloor \cdot |W(s_{\text{old}})| + \lambda(s_{\text{old}}, r, d \bmod d^r_{\text{old}})$
            AddPath($s_{\text{new}}, r, d, p, \lambda$)
    return $s_{\text{new}}$
```

(b) Function ScaleSolution

```
function ScalingConstructor(Graph $G$, Requests $R_{\text{orig}} = \{(u^r, v^r, d^r_{\text{orig}})\}$, Scaling $a$)
    $k \leftarrow \left\lceil \log_a \frac{\sum_{r \in R} d^r}{10000} \right\rceil$, $f \leftarrow a^k$
    $R_{\text{scaled}} \leftarrow$ ScaleRequests($R_{\text{orig}}, f$)
    $s_{\text{scaled}} \leftarrow$ CreateInitialSolution($G, R_{\text{scaled}}$)
    return ScaleSolution($s_{\text{old}}$, $R_{\text{scaled}}$, $R_{\text{orig}}$)
```

(c) Function ScalingConstructor

```
function MultiLevelSearch(Graph $G$, Requests $R_{\text{orig}} = \{(u^r, v^r, d^r_{\text{orig}})\}$, Scaling $a$, Time $t$)
    $s, R_{\text{new}}, R_{\text{old}} \leftarrow \emptyset$
    $k \leftarrow \left\lceil \log_a \frac{\sum_{r \in R} d^r}{10000} \right\rceil$
    for $i \leftarrow 0, 1, \ldots, k-1, k$ do
        $f \leftarrow a^{k-i}$
        $R_{\text{old}} \leftarrow R_{\text{new}}$, $R_{\text{new}} \leftarrow$ ScaleRequests($R_{\text{orig}}, f$)
        if $i = 0$ then                          ▷ First Iteration, strongest scaling
            $s \leftarrow$ BFD($G, R_{\text{new}}$)
        else
            $s \leftarrow$ ScaleSolution($s, R_{\text{old}}, R_{\text{new}}$)
        if $i < k$ then                          ▷ ILS on scaled instance
            $s \leftarrow$ ILS($s$, Time($t$) $\vee$ Conv($c$))
        else                                     ▷ ILS on original instance
            $s \leftarrow$ ILS($s$, Time($t$))
    return $s$
```

(d) Function MultiLevelSearch

Figure 3: Functions

comparative experiments with BFD followed by ILS) and in the unscaled variant (to show the capability of our new approaches). To study the influence of multiplying an instance's demand, we use instance `nobel-us`▲ which is enlarged by factor 50. New instances are `ta1`▼ and `ta2`▼, both scaled down by a factor of 100 due to a demand of $\geq 10^6$. To show that the ML is applicable to smaller instances, too, we include the largest instance from [1] (`zib54`). Time limits as termination criterion are given in Tab. 1 allowing comparisons between algorithms. For our experiments, we set $a = 4$ (scaling) and $c = 5$ (convergence). Our experiments were conducted on a Pentium 4 3.0 GHz system running

Instance	$\lvert V \rvert$	$\lvert E \rvert$	D	UB	Time
`atlanta`	15	22	136 726	25 167	2 h
`france`	25	45	99 830	10 610	2 h
`janos-us-ca`	39	122	2 032 274	262 400	6 h
`janos-us-ca`▼	39	122	203 222	26 181	6 h
`nobel-us`▲	14	21	271 000	34 680	6 h
`ta1`▼	24	51	101 271	6 017	1 h
`ta2`▼	65	108	314 207	19 054	6 h
`zib54`	54	81	12 230	707	5 min

Instances with ▼ or ▲ have been modified, see text for details.
`janos-us-ca`▼'s demand is not 203227 as each request is scaled independently.

Table 1: Properties of our benchmark instances. 'UB' is the best found (either here or from [1]) solution's cost and 'Time' is the time limit for each instance.

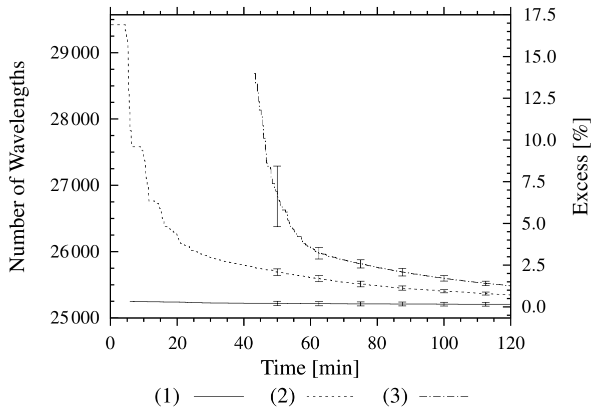

Figure 4: Comparing three setups applied to `atlanta`:
(1) Multilevel approach with ILS in each step,
(2) Scaling Constructor and optimizing with ILS,
(3) BFD constructor and optimizing with ILS (cILS).
Plot includes 99 % confidence intervals.

Linux Kernel 2.6. Algorithms were implemented in Java 6. Each setup was repeated 15 times with different seeds, average values were used for the following discussion.

4. Experimental Results

The results from our experiments are summarized in Tab. 2, where for each instance and both approaches and the conventional ILS (cILS) the elapsed time [s] and excess [%] (above the best known solution) is shown. Results are grouped in snapshot events at the points in time (a) when the BFD constructs its solution ('BFD initialized'), (b) after half of the time ('Half Time') and (c) at reaching the time limit ('Termination').

The general behavior of the three approaches is shown in Fig. 4. Comparing to the two new approaches, the BFD constructor requires the most time to initialize a feasible solution. E. g. for `france`, the BFD takes about $240 \approx 16^2$

	(a) BFD initialized				(b) Half Time				(c) Termination			
Instance	Time	cILS	SC	ML	Time	cILS	SC	ML	Time	cILS	SC	ML
`atlanta`	2365.7	16.49	2.54	0.22	3600.0	3.53	1.75	0.19	7200.0	1.25	0.71	0.15
`france`	1426.2	5.42	0.50	0.25	3600.0	0.38	0.18	0.12	7200.0	0.12	0.07	0.05
`janos-us-ca`	–[1]	–	–	–	10800.0	–	38.72	–	21600.0	–	38.72	–
`janos-us-ca`▼	9473.5	36.33	1.88	0.34	10800.0	28.40	1.77	0.32	21600.0	1.16	1.29	0.29
`nobel-us`▲	7228.2	31.25	4.44	0.72	10800.0	9.16	3.27	0.71	21600.0	3.23	1.28	0.71
`ta1`▼	1020.6	25.41	1.91	1.17	1800.0	2.60	1.13	0.99	3600.0	1.34	0.55	0.81
`ta2`▼	–[1]	–	–	–	10800.0	–	2.05	0.44	21600.0	–	1.06	0.29
`zib54`	25.5	26.45	1.60	1.15	150.0	0.62	0.41	0.46	300.0	0.37	0.23	0.36

Instances with ▼ or ▲ have been modified, see text for details. [1] No result within time limit.

Table 2: Overview on solution quality (elapsed time [s] and excess [%] above best known solution for three snapshot events).

times longer than the SC approach, where $a^k = 16$ for this instance supporting our estimations from Sec. 2, but the BFD finds better initial solutions. However, investing the SC's saved time in the ILS results in superior solutions at the point in time where the BFD finds its initial one. The more sophisticated ML approach significantly amplifies the SC's behavior. Due to the expansion and intermediate ILS steps, this approach takes longer to find a feasible solution for the original instance compared to the SC, but is still much faster than the BFD. Nevertheless, the solution quality significantly outperforms both other approaches. E. g. for `janos-us-ca`▼, the initial feasible solution of ML (ca. 15 min) is not reached by either SC or BFD within the time limit of 6 hours.

For the ML approach, the computation times increase with each expansion step: E. g. for instance `ta1`▼ and $a^k = \{64, 16, 4, 1\}$, the average times increase from 9.95 s to 57.9 s, 505.9 s, and 21882.6 s, respectively.

The inherent symmetries due to multiplying the same (pre-optimized) solution can affect the performance of both SC and ML. For `nobel-us`▲ we observe for initial feasible solutions constructed by ML (< 15 min) virtually no improvement (< 0.1 %) within the given time limit of 6 hours. However, within our repetitions we observe a difference in solution quality spanning about 2 % excess. For `ta1`▼, the SC results in slightly better solutions compared to ML, as the latter one is stronger affected by the symmetries.

Although both the SC and the ML are designed for large instances, they can be applied to small instances without affecting solution quality. As can be seen for `zib54`, the performance of the three approaches is indistinguishable resulting in an average excess between 0.23 % and 0.37 %.

Considering our largest instance `janos-us-ca`, only SC is able to find a valid solution within the time limit. However, multiplying the number of wavelengths with its scaling factor an ML's intermediate solution yields far better results, which were thus used as the best known results. These intermediate results were found after < 1 hour supporting the ML's advantage to both the SC and BFD.

5. Conclusions

We proposed both a multilevel-inspired construction heuristic and a multilevel approach using ILS for refinement for the RWA. Finding initial solutions for large instances in comparably short time, the SC can be used as a replacement for the BFD constructor. Due to initial optimization steps on simplified instances, the multilevel approach already starts in excellent time. Using ILS, the ML results in solutions far better than the BFD constructor followed by ILS.

Future work will focus on the development of recombination operators and distributed algorithms for the RWA.

References

[1] K. Bauer, T. Fischer, S. O. Krumke, K. Gerhardt, S. Westphal, and P. Merz. Improved Construction Heuristics and Local Search for the Routing and Wavelength Assignment Problem. In C. Cotta and J. van Hemert, editors, *EvoCOP 2008*, volume 4972 of *LNCS*. Springer, 2008.

[2] I. Chlamtac, A. Ganz, and G. Karmi. Lightnet: Lightpath Based Solutions for Wide Bandwidth WANs. In *INFOCOM '90*, volume 3, pages 1014–1021, 1990.

[3] J. S. Choi et al. A Functional Classification of Routing and Wavelength Assignment Schemes in DWDM networks: Static Case. In *Proc. OPNET 2000*, pages 1109–1115, 2000.

[4] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *Proc. of INOC 2007*, 2007.

[5] A. E. Ozdaglar and D. P. Bertsekas. Routing and Wavelength Assignment in Optical Networks. *IEEE/ACM Transactions on Networking*, 11(2), 2003.

[6] M. C. Sinclair. Minimum cost routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach. In *Proc. 6th IEE Conf. Telecom.*, 1998.

[7] N. Skorin-Kapov. Routing and Wavelength Assignment in Optical Networks using Bin Packing Based Algorithms. *EJOR*, 177(2):1167–1179, 2007.

[8] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. *Annals of Op. Res.*, 131:325–372, 2004.

[9] H. Zang et al. A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks. *Optical Networks Magazine*, 1:47–60, 2000.