A Distributed Memetic Algorithm for the Routing and Wavelength Assignment Problem

Thomas Fischer, Kerstin Bauer, and Peter Merz

Department of Computer Science, University of Kaiserslautern, Germany
{fischer,k_bauer,pmerz}@cs.uni-kl.de

Abstract. The Routing and Wavelength Assignment Problem deals with the routing of telecommunication traffic in all-optical networks. Extending existing algorithms, we present a memetic algorithm (MA) for the static RWA by introducing a recombination operator and a scheme for distributing the computation. Compared to previously achieved results for this problem, our MA significantly improves the solution quality. We find provably optimal results for previously unsolved problem instances. The distributed variant using epidemic algorithms allows to find solutions of quality comparable to the MA in less real-time.

1 Introduction

The *Routing and Wavelength Assignment* problem (RWA), an NP-complete [1] graph-theoretical problem, deals with *Wavelength Division Multiplexed* (WDM) optical networks, where communication requests between nodes in a network have to be fulfilled by routing them on optical fiber links with given capacities.

A problem instance of the RWA is a physical network represented by a graph $G = (V, E, W)$ with nodes V, edges E, and wavelengths W. The optical fiber links in the physical network are represented by E (here undirected) and on each links each wavelength in W is available. A node in V can be starting point u^r or end point v^r of a connection request $r = (u^r, v^r, d^r) \in R$ with a demand of $d^r \in \mathbb{N}^+$. For each unit of demand a *lightpath* has to be established. A lightpath is a path between two nodes in the physical network utilizing one wavelength on each link. The *wavelength continuity constraint* requires the path to use the same wavelength on every link, the *wavelength conflict constraint* states that no wavelength on a link may be used by more than one lightpath at the same time.

In the RWA's *static* variant, a set of requests is given and one can either minimize the number of wavelengths to route all requests or maximize the number of routed requests for a given set of wavelengths. In the *dynamic* variant, requests turn up over time and one has to maximize the number of routed requests. Here, we focus on minimizing the number of used wavelengths in the static case.

Next, related work and a previous publication is discussed. We present our MA and the recombination operator in Sec. 2. The transformation into a distributed MA is described in Sec. 3. After motivating our experimental setup in Sec. 4, we present our results in Sec. 5.

G. Rudolph et al. (Eds.): PPSN X, LNCS 5199, pp. 879–888, 2008.

1.1 Related Work

Early papers were due to Bala *et al.* [2] and Chlamtac *et al.* [3], where both wavelength constraints were defined and algorithms for the dynamic RWA were presented. A popular approach for the RWA is to split the solution finding process into solving the subproblems of routing and wavelength assignment independently. For the static RWA, Banerjee and Mukherjee [4] used this approach, where the routing part is solved by relaxing the wavelength continuity constraint, solving a fractional multicommodity flow problem, and using randomized rounding to find paths in the graph. For the wavelength assignment a graph coloring problem is solved, where the nodes represent the paths and the edges state whether the incident paths share a common physical link. Another approach for solving the wavelength assignment part was given by Manohar and Shevgaonkar [5], who defined it as an instance of the maximum edge disjoint paths problem, where a maximum-sized subset of all paths in a graph is wanted holding that no two paths share a common link. Iteratively, among the paths with no wavelength assigned, the maximum subset is determined the next unused wavelength assigned to that path set, which is then removed from subsequent iterations. A general overview and classification of algorithms solving both subproblems independently is provided in [6] by Zang *et al.* and in [7] by Choi *et al.*

Sinclair [8] presented a memetic algorithm for the static RWA. Its intricate cost function does not match any of the RWA's objectives as stated above. The algorithm features mutation, recombination, and two local search operations which are applied with varying probabilities. The mutation reroutes a given request on a path randomly chosen from the set of k-shortest paths between the endpoints using the first available wavelength. The recombination performs a crossover where the set of requests is split and both offsprings contain paths and wavelength assignments from one of the halves while the other half only provides the paths using new wavelength assignments. The first local search tries to reroute a request in a wavelength from a high index in a lower-indexed wavelength using a k-shortest path. The second local search operates similarly, but here a target wavelength is chosen first and all conflicting paths are rerouted before rerouting the path from the high-indexed wavelength. Given that the algorithm uses problem-specific operators, the large population size (500) and the vast number of generations (100 000) question its efficiency.

Skorin-Kapov [9] introduced a construction heuristic called BFD_RWA based on bin-packing algorithms. Requests are sorted non-increasingly by length of their shortest paths and routed in the wavelength with shortest available path.

In [10], we presented a new set of benchmark instances, on which an iterated local search (ILS) algorithm was applied. The ILS consists of a local search (LS) and a mutation operator. The LS's idea is to move requests from less used wavelength to highly used wavelengths with the intention to clear already sparse wavelengths. The mutation operator randomly selects two wavelengths λ_1 and λ_2 where w. l. o. g. the load of λ_2 is smaller than λ_1. A path routed in λ_2 is randomly chosen and moved to λ_1 by using the shortest path between both endpoints. Any conflicting path in λ_1 is forcefully removed and later reinserted

```
1: function InitializePopulation(Graph G, Requests R)
2:     S ← ∅
3:     for i ← 1, ..., n do
4:         s ← BFD_RWA(G, R)
5:         S ← S ∪ {s}
6:     return S

1: function MemeticAlgorithmRWA(Graph G, Requests R)
2:     S ← InitializePopulation(G, R)
3:     while !TerminationReached do
4:         S' ← ∅
5:         for all s ∈ S do
6:             if Random() ≤P[recomb] then
7:                 s̄ ← ChooseIndividual(S, s)
8:                 s' ← Recombinator(G, s, s̄, R)
9:             else
10:                s' ← Mutate(s, R, strength)
11:            s'' ← LocalSearch(s', R)
12:            if s'' < s then
13:                S' ← S' ∪ {s''}
14:            else
15:                S' ← S' ∪ {s}
16:        S ← S'
17:    return S
```

Fig. 1. Memetic algorithm for the RWA

by the construction heuristic's approach. The ILS provides solutions near the lower bound (determined by relaxing the wavelength continuity constraint and solving the resulting multicommodity flow problem) and, for some instances even optimal solutions can be achieved.

It is equivalent having between any node pair either at most one request with demand ≥ 1 or multiple requests with demand $= 1$ each. We use the latter definition to simplify the notation of algorithms presented in this paper.

2 Algorithms

In this paper, we present a recombination operator (Fig. 1) which is used to create a population-based memetic algorithm (MA) for the static RWA. Based on the ILS presented in [10], the MA adds the feature to handle populations of solutions and to recombine two solutions to one offspring solution.

The population is initialized by using the BFD_RWA construction heuristic for each individual. Although this heuristic orders requests by the length of their shortest path, requests are inserted in random order within each subset of equal shortest path lengths, thus resulting in different solutions. Until reaching some termination criterion the MA iterates over the population. Our termination criterion is an instance-dependent time limit allowing to compare different setups

```
1: function RECOMBINATOR(Graph G, Solution a, Solution b, Requests R)
2:     s ← a, R' ← ∅, P_b ← ⋃_{r∈R} p_{G,b}(r)
3:     for λ ∈ {λ_1, ..., λ_{|W|} : u(λ_i) ≥ u(λ_{i+1})} do
4:         for r ∈ R : λ_s(r) = λ do
5:             if p_{G,s}(r) ∈ P_b then
6:                 P_b ← P_b \ {p_{G,s}(r)}
7:             else
8:                 s ← s \ r, R' ← R' ∪ {r}
9:     for all r ∈ R' do                          ▷ for all currently unrouted paths
10:        λ_s(r) ← arg min_{λ∈W} Ψ_{G,s}(r, λ) ≠ ∅
11:        p_{G,s}(r) ← Ψ_{G,s}(r, λ_s)          ▷ route request on the first possible wavelength
12:    return s
```

Fig. 2. Recombination operator for the RWA

for the same instance. In each iteration, for each individual the same steps are performed: First, with given probabilities (Sec. 4) either a recombination (s.b.) or a mutation step as introduced in [10] is performed. The mutation strength can be varied, the actually used strategy is $10\%\downarrow_{2\,\%}$ (starting with mutation strength 10 % and decreasing in each iteration by 2 % until reaching the fixed minimum of 1 %) as in [11]. Once a provisional offspring has been created, this individual is improved by the LS used in [10].

For each individual of the next generation, offspring and parent are compared and the better one is kept. Thus, the only interaction between individuals of the same generation is the recombination operator. Setting the recombination probability to 0.0, the memetic algorithm equals to as many independent ILS runs as there are individuals in the population.

The recombination operator allows to combine common features of two parent individuals to one offspring. The offspring only keeps paths existing in both parents using the wavelength assignments from the better one. The remaining requests are later inserted using a construction heuristic. To determine the set of paths common to both parents, a matching problem between paths from both parents has to be solved. Two paths can be matched iff they use the same set of links regardless of the wavelengths assigned to the paths.

Details of the recombination operator are given in Fig. 2. W. l. o. g. parent solution a is better than b as defined by the length-lex ordering from [10]. The offspring solution s is initialized as a copy of a. To determine a matching between paths from a and b, b's set of paths is stored in the multiset P_b. In the recombination's first phase the algorithm iterates on the set of wavelengths sorted non-increasingly by usage. For each request using the current wavelength, it is checked whether the request's path $p_{G,s}(r)$ is element of P_b. If the test holds, a match has been found and the path is removed from P_b to prevent future matchings. Otherwise, the request is removed from s (removed requests are collected in R'). The sorting of wavelengths here keeps paths in already highly used wavelengths more likely than paths in less often used wavelengths. We argued that introducing a gap in already highly used wavelengths by not matching paths in

```
1: function DistributedMemeticAlgorithmRWA(Graph G, Requests R)
2:     s ← InitializeIndividual(G, R)
3:     while ¬TerminationReached do
4:         if {q_1, ..., q_k} = Q ≠ ∅ then
5:             s̄ ← q_1,    Q ← {q_2, ..., q_k}
6:             s' ← Recombinator(G, s, s̄, R)
7:         else
8:             s' ← Mutate(s, R, strength)
9:         s'' ← LocalSearch(s', R)
10:        if s'' < s then
11:            s ← s''
12:        if Random≤P[recomb] then
13:            SendToRandomNeighbor(s)
14:    return s
```

Fig. 3. Distributed memetic algorithm for the RWA

those wavelengths can be less likely exploited for future improvements compared to paths in less often used wavelengths. Finally, the removed requests R' have to be reinserted into s using the concept of the BFD_RWA construction heuristic, where the request is routed in the first wavelength where a feasible path connecting both endpoints is available.

3 Distribution

The MA has been enhanced to a distributed memetic algorithm (DMA), differing from the MA by using populations of n individuals in one algorithm instance, n algorithm instances with one individual each operate independently. Resembling the MA, the DMA's instances exchange individuals regularly over the network using an epidemic algorithm [12].

The DMA is shown in detail in Fig. 3 is similar to the MA (Fig. 1), as the main differences are (a) the DMA operates on a single individual per algorithm instance (b) the exchange and recombination between algorithm instances is managed differently. Each algorithm instance has a receiving queue Q where incoming individuals are stored temporarily (realized by using an asynchronous thread). At the beginning of each iteration it is checked if the queue contains at least one element. If the test holds, the queue's top element is removed and used in a recombination operation. Otherwise, the current solution is mutated. After the obligatory local search and selection, with a given probability the current solution is sent to one randomly selected neighboring algorithm instance. The probability for sending an individual equals to the recombination probability in the original MA. Due to the asynchronous nature of the DMA and the queuing effects, the actual recombination probability may differ.

To build a neighborhood relation between algorithm instances (nodes) in distributed setups, we use an epidemic algorithm [12,13], where each node maintains a list of neighboring algorithm instances. For the node neighbor lists' initialization,

one node is selected and its contact information is provided to other nodes. During the execution of the algorithm, from time to time each node chooses a neighbor and both nodes exchanges their neighbor lists. Any received list will be merged with the receiving node's list. The epidemic algorithm's membership protocol settled the neighbor list for each node in less than 5 s in each case providing a stable neighbor list. The receiving queue Q for incoming solutions was limited to 16 elements (twice the largest population size) and any solution received at a node with full queue was discarded.

4 Experimental Setup

In [10], problem instances based on network data from the SNDlib collection [14] were presented. As all but four of these instances have been optimally solved (lower bound reached) and for one the difference between best known solution and the lower bound is only one wavelength, we use the remaining instances `janos-us-ca`, `nobel-us`, and `zib54` for the experiments in this paper. For the SND problem, Atamtürk and Rajan [15] recently presented new benchmark instances, which we converted to RWA instances, too. As the authors use fractional demands in their data, we multiplied them by 100. Using the concept of undirected edges, we added the demands with different directions between the same nodes. For each instance size, the authors provide three networks: (a) average node degree 4 (b) average node degree 8 (c) edge density 75 %. We performed experiments with problem instances of size 15 and 20. The properties of all instances used in our experiments are summarized in Tab. 1.

We considered population sizes of 2, 4 and 8 and recombination rates of 0.2, 0.4, 0.6, 0.8, and 1.0 (recombination only). For comparison, the original ILS was reimplemented by using population size 1 and a recombination rate of 0.0

Table 1. Properties of instances used in our experiments 'Pairs' describes the number of node pairs communicating with each other, 'Requests' summarizes the demand (sum of paths to be established). 'LB' designates the known lower bound from [10] and 'UB' the best result found in any setup in this paper.

Instance	Nodes	Edges	Pairs	Requests	LB	UB	Time [s]
`janos-us-ca`†	39	122	1482	10173	1288	1288	1200
`nobel-us`	14	21	91	5420	670	670	300
`zib54`	54	81	1501	12230	705	705	600
`15.50.75`	15	90	72	9500	–	155	1500
`15.50.deg4`	15	48	72	9500	–	366	450
`15.50.deg8`	15	68	72	9500	–	258	450
`20.50.75`	20	150	137	18210	–	188	3000
`20.50.deg4`	20	82	137	18210	–	439	1000
`20.50.deg8`	20	106	137	18210	–	294	2000

† Instance has been modified, see text for details.

(no recombination). All experiments were repeated 30 times with different seeds. We used up to eight identical cluster PCs (Pentium 4 with 3 GHz running Linux) connected in a switched 100 MBit network. All software was written in Java.

Furthermore, to guarantee comparability between DMA and MA, both algorithms are given the same total time per instance (see Tab. 1) by dividing the time limit per CPU by the population size in the distributed case.

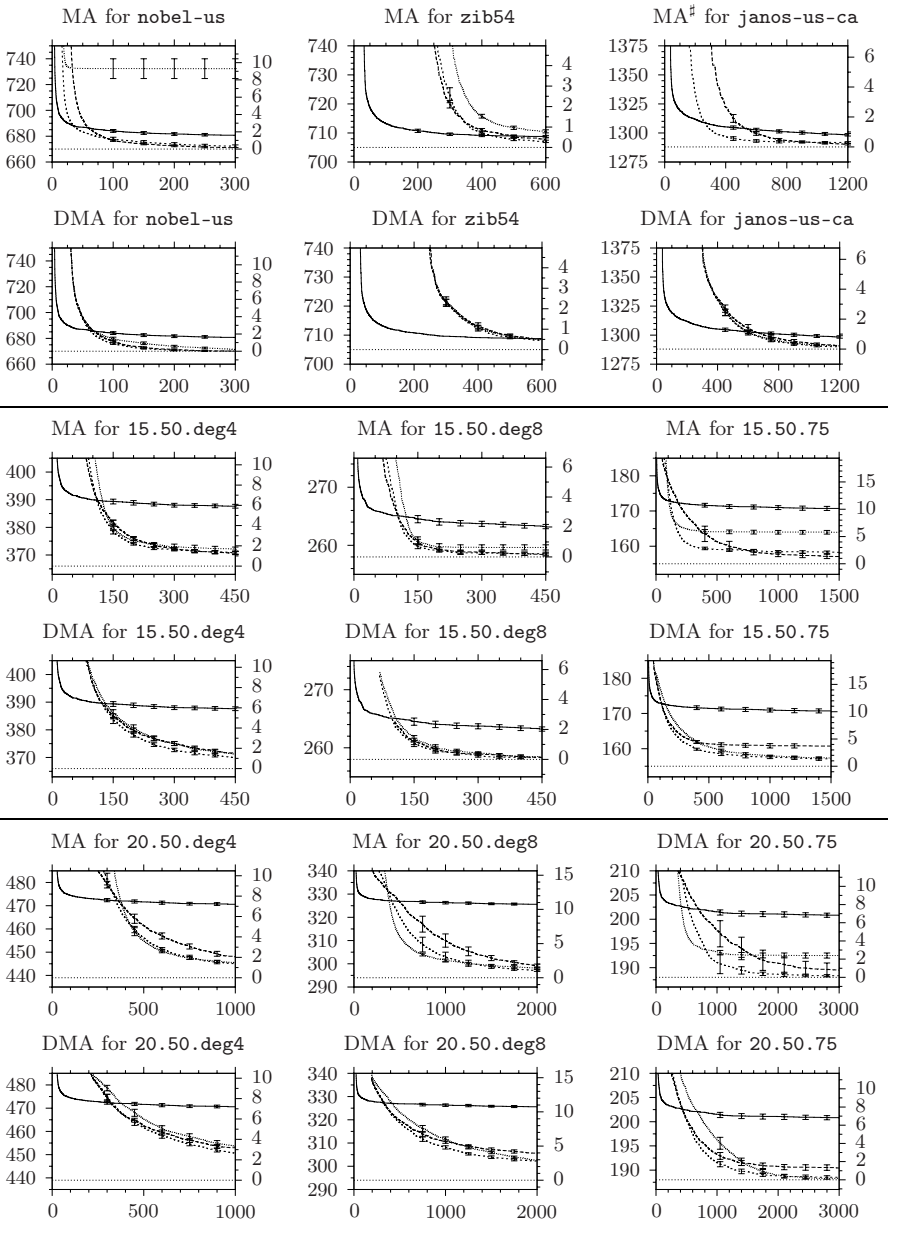
5 Experimental Results

Experiments using the MA and DMA were conducted as described above. In Fig. 4 the performance of both the distributed and non-distributed MA variants operating on 8 individuals for every benchmark instance is visualized. Except for instance `zib54` and a pathological case for `nobel-us` (discussed below), both variants of our MA find significantly better results compared to the original ILS. For the three instances already discussed in [10], our memetic algorithm was able to find optimal solutions for `janos-us-ca` in 4 MA setups and 93 DMA setups, for `nobel-us` in 28 and 402 setups, respectively, and for `zib54` in 36 and 82 setups, respectively, out of 450 runs each.

Regarding population size, small sizes are less capable of maintaining sufficient diversity within the population than larger populations. Interestingly, this effect is stronger for non-distributed than for distributed setups. The DMA's population stays more diverse, as the DMA's queue Q (see Fig. 3) provides a memory of older and thus more different solutions. E. g. for `20.50.deg4`, the percentage of paths with the same edges regardless of used wavelengths (*relative similarity* between two solutions) of both parents for a recombination in the MA is on average over time and all repetitions $> 98.5\,\%$ for all recombination rates, whereas for the corresponding DMA setup the similarity is $< 95.0\,\%$. For recombination rate 1.0 in the non-distributed case, the relative similarity decreases from 99.9 % to 95.5 % and 91.2 % for population sizes of 2, 4, and 8, respectively.

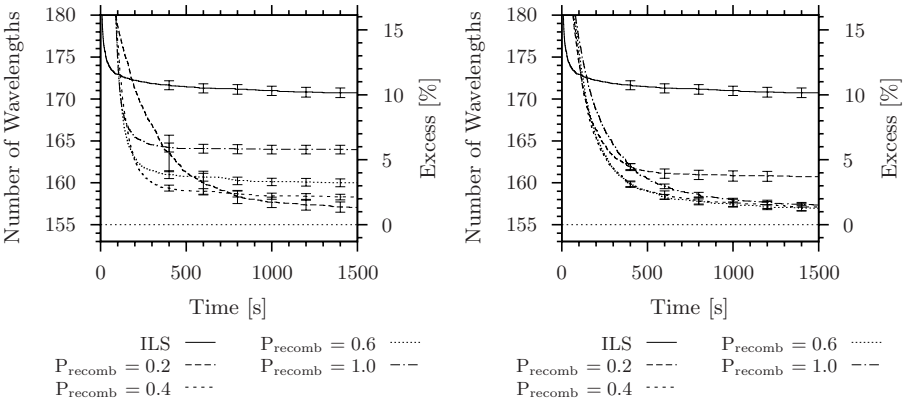
Regarding recombination rates, three different patterns can be observed: (a) all rates perform similarly (b) recombination rate 1.0 performs significantly worse (c) recombination rate 0.2 performs significantly worse. Case (b) occurs most often for the MA, whereas case (c) is more common for the DMA.

We discuss the setup with `15.50.75` (Fig. 5) with population size 8 in detail. The two plots show the run-time behavior of the non-distributed and distributed setup, respectively, for selected recombination rates in comparison to the original ILS and the best known solution. Furthermore, instance `15.50.75` represents recombination patterns (b) and (c), respectively. For each setup, the average number of generations, the effective recombination rate, and the average similarity of recombination partners are summarized below the plots. The number of generations decreases with increasing recombination rate as a recombination operation requires more computation time than a mutation operation. The effective recombination rate determined by counting the number of recombinations is exactly the expected value for the non-distributed case. For the DMA, the effective rates are lower as recombinations are only performed if an individual's queue Q



Line type − represents the ILS, types ---, ····, and ····· represent recombination rates 0.2, 0.4, and 1.0, respectively. The horizontal line at each plot's bottom is the best found solution's quality. 99 % confidence intervals are given.

Fig. 4. Overview on the performance for each instance both for the non-distributed and distributed variant operating on 8 individuals. Axis labels are the same as in Fig. 5. ♯ Setups with 1.0 recombination rate are out of scale.



(a) non-distributed MA **(b)** Distributed MA

	Non-distributed				Distributed			
	0.2	0.4	0.6	1.0	0.2	0.4	0.6	1.0
# Gen.	2214.3	2317.6	2189.4	1959.8	1812.6	1619.6	1414.7	1151.2
$P^{\text{eff}}_{\text{recomb}}$	0.200	0.400	0.600	1.000	0.196	0.391	0.582	0.886
sim [%]	94.4 %	96.9 %	97.2 %	97.6 %	91.1 %	93.8 %	94.4 %	94.6 %

Fig. 5. Detailed analysis of experimental results for setups with instance `15.50.75` with 8 individuals. The horizontal line at each plot's bottom is the best found solution's quality. 99 % confidence intervals are given.

is non-empty. Due to varying time requirements for mutation and recombination and short-time effects, queues may be filled beyond the capacity limit for a few generations and thus incoming solutions get discarded. For high recombination rates, this effect is stronger explaining the lower effective recombination rates. Due to this effect pattern (b) does not occur for distributed setups.

In our recombination, parents pass on common path and wavelength assignments to their offspring thus requiring to insert new path and assignments to get a feasible solution (partial restart). Low similarity yields large partial restarts which may degrade the offspring's solution quality, for too high similarity the restart is too small to escape local optima. This model is well supported by our data as depicted in Fig. 5. In the non-distributed case, the final solution quality decreases with increasing similarity of the parents. In the distributed case, all recombination rates except rate 0.2 have the same parent similarity of about 94 % and the same solution quality. Only the recombination rate 0.2 has a significantly lower similarity and performs considerably worse.

6 Conclusions

In this paper we presented a memetic algorithm including a recombination operator for the static RWA which significantly improved former results. Furthermore,

the MA can be efficiently distributed in a real network allowing to find results similar to the single CPU variant given the same total time summed over all participating CPUs. Future work will focus on combining our MA and DMA with the multilevel approach [11] for large instances.

References

1. Chlamtac, I., Ganz, A., Karmi, G.: Lightnet: Lightpath Based Solutions for Wide Bandwidth WANs. In: INFOCOM 1990, vol. 3, pp. 1014–1021 (1990)
2. Bala, K., Stern, T.E., Bala, K.: Algorithms for Routing in a Linear Lightwave Network. In: IEEE INFOCOM, vol. 1, pp. 1–9 (1991)
3. Chlamtac, I., Ganz, A., Karmi, G.: Lightpath Communications: An Approach to High Bandwidth Optical WANs. IEEE Trans. Comm. 40(7), 1171–1182 (1992)
4. Banerjee, D., Mukherjee, B.: A Practical Approach for Routing and Wavelength Assignment in Large Wavelength-Routed Optical Networks. IEEE J. Sel. Areas Comm. 14(5), 903–908 (1996)
5. Manohar, P., Manjunath, D., Shevgaonkar, R.K.: Routing and Wavelength Assignment in Optical Networks From Edge Disjoint Path Algorithms. IEEE Communications Letters 6(5), 211–213 (2002)
6. Zang, H., Jue, J.P., Mukherjee, B.: A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks. Optical Networks Magazine 1, 47–60 (2000)
7. Choi, J.S., Golmie, N., Lapeyrere, F., Mouveaux, F., Su, D.: A Functional Classification of Routing and Wavelength Assignment Schemes in DWDM networks: Static Case. In: OPNET 2000, pp. 1109–1115 (2000)
8. Sinclair, M.C.: Minimum cost routing and wavelength allocation using a genetic-algorithm/heuristic hybrid approach. In: Proc. 6th IEE Conf. Telecom. (1998)
9. Skorin-Kapov, N.: Routing and Wavelength Assignment in Optical Networks using Bin Packing Based Algorithms. EJOR 177(2), 1167–1179 (2007)
10. Bauer, K., Fischer, T., Krumke, S.O., Gerhardt, K., Westphal, S., Merz, P.: Improved Construction Heuristics and Iterated Local Search for the Routing and Wavelength Assignment Problem. In: van Hemert, J., Cotta, C. (eds.) EvoCOP 2008. LNCS, vol. 4972, pp. 158–169. Springer, Heidelberg (2008)
11. Fischer, T., Bauer, K., Merz, P.: A Multilevel Approach for the Routing and Wavelength Assignment Problem. In: Köppen, M., Raidl, G. (eds.) HEUNET 2008. IEEE Comp. Soc. Press, Los Alamitos (2008)
12. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. In: Schneider, F.B. (ed.) ACM PODC, pp. 1–12. ACM Press, New York (1987)
13. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Transactions on Computer Systems 25(3) (2007)
14. Orlowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0–Survivable Network Design Library. In: Proc. INOC 2007 (2007)
15. Atamtürk, A., Rajan, D.: Partition inequalities for capacitated survivable network design based on directed p-cycles. Discrete Optimization 5(2), 415–433 (2008)