Different approaches for solving ring-dimensioning problem

Mladen Kos, Matija Mikac and Domagoj Mikac

Faculty of Electrical Engineering and Computing,
University of Zagreb,
Unska 3, HR-10000 Zagreb, Croatia
e-mail:{kos,domagoj,matija}@tel.fer.hr

Abstract: In this paper, an overview of different methods used for solving a single ring-dimensioning problem is presented. All the algorithms discussed are implemented in our software tool called, RingSolver, and are being used in another tool for designing more complex ring-networks, which is still under development. Since, majority of algorithms are relatively simple to implement, and they provide very good solutions in short execution time it is reasonable to use them when planning and dimensioning larger ring-based networks. Such networks can contain huge number of rings and therefore it is very important to have algorithms that can handle problems fast and effective.

Keywords: ring loading problem (RLP), algorithms

Introduction

In design process of today's high-capacity backbone networks, different network architectures are used. In order to minimize possible data losses and service unavailability in case of network failure, all the architectures must include reliable and fast protection and restoration mechanisms. One of the concepts used are ring-based networks. Such networks are built using self-healing rings (SHR). SHRs are very useful and functional network structures that provide automatic protection and restoration mechanisms. As in all other network design processes, the objective is to find cost-optimal or near-optimal network solution that satisfies all the requirements. Protection requirements are fulfilled by usage of SHRs with automatic protection mechanisms, so that main problem arising in ring-network design is ring-dimensioning problem, sometimes also called as ring loading problem (RLP). Ring-based network is optimized, if each SHR it is build of, is optimized, plus if all SHR interconnections are selected and dimensioned so that overall network has lowest costs. This article discusses only single SHR optimization and dimensioning methods. SHR is said to be optimized if its capacity is minimal possible to carry all the traffic. There are many different algorithms developed for SHR optimization. Our article gives an overview and discusses results obtained by these algorithms. All described algorithms are implemented in our software tool called RingSolver, and are being used in some more complex network design tools under development.

In section 1, ring model used for problem description and solving is described, together with main formulation of ring loading problem. Section 2 describes different ring dimensioning approaches, distinguishes ring-loading problems with and without demand splitting, and explains why there is a need to develop fast and efficient algorithms for solving ring-loading problems. Section 3 includes overview and short remarks about different algorithms introduced by other authors, proposes mathematical formulation of ring loading problem that can be used for solving problem using linear/integer programming. In section 4, results obtained using different algorithms are shown and discussed. Section 5 is used for conclusion, including remarks about usage of implemented algorithms for solving ring-loading problem in more complex applications.

1. Ring model

Our ring model can be defined as follows: let ring R consists of two sets, $V=\{n_1,n_2... n_N\}$, set of nodes and $E=\{e_{12},e_{23}... e_{N1}\}$, set of edges, where N is

total number of nodes/edges in the ring. Each edge e_{ij} represents a link between two neighbor nodes n_i and n_j where $j = (i + 1) \ mod \ N$. Set of traffic demands D consists of traffic demands d_{ij}, $1 \leq i < j \leq N$ that are defined between node n_i and n_j. Each demand is routed through a subset of edges $DE \subset E$, with $K<N$ edges. When defining demand d_{ij} $i<j$ is supposed. Each demand is characterized with amount of traffic (positive integer value), and with its direction – if demand is routed from node n_i to node n_j passing node sequence (n_i, n_{i+1}, ... n_j) it is said to have a "clockwise direction", while if demand from node n_i to node n_j through node sequence (n_i, n_{i-1}... n_1, n_N,.. n_{j+1}, n_j) we say it has a "counter-clockwise direction".
Simple ring configuration is shown in Figure 1.

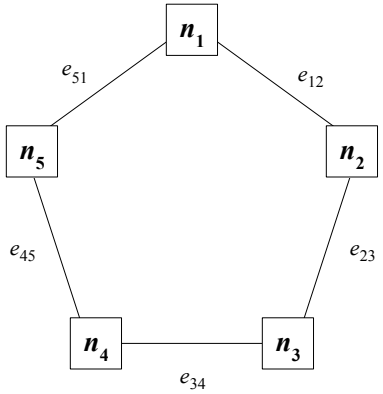

Figure 1: Simple ring configuration with 5 nodes

Depending on possible direction of demands routing, rings are classified into two types – unidirectional and bidirectional rings. In unidirectional rings (USHR) all demands are routed in the same direction (either clockwise or counter-clockwise), while in bidirectional rings (BSHR) demands can be routed in both directions – clockwise and counter-clockwise. Additionally BSHR can be classified into two subclasses – rings without demand splitting and rings with allowed demand splitting. When demand splitting is allowed, one part of demand can be routed in clockwise, and another in counter-clockwise direction.

Link load is a sum of all demands passing through that link. The ring capacity is defined as maximum of all link loads. The ring cost is usually increasingly proportional with its capacity. Therefore, in order to minimize the ring cost we should try to find minimal ring capacity. In other words, optimal routing for given demands should be found, so that maximum link load will be minimized. Problem described previously is usually called Ring Loading Problem (RLP).

It is obvious that when working with USHR, there is only one possible ring capacity (all the demands are routed clockwise or counter-clockwise, resulting the same maximum load through edges). Therefore, it is reasonable to define RLP only for BSHR rings. However, USHR rings are also often used in ring-based network design, especially when building access networks – such protective ring structure replaces standard stars and trees, and it is sometimes combined with dual homing structures.

2. Ring dimensioning

As described earlier, each demand can be routed through the ring in only one direction if splitting is not allowed, or in two directions if demand splitting is allowed. Directions are defined as clockwise and counter-clockwise. Depending on whether splitting of demands is allowed or not, there are two classes of RLP – RLP with demand splitting (RLPW), and RLP without demand splitting (RLPWO). Let us formulate ring dimensioning problem using previously introduced ring model, which consists of three sets: set of nodes $V = \{n_1, n_2, n_3,... n_N\}$, set of edges $E = \{e_{12}, e_{23}, e_{34}, ... e_{N-1N}, e_{N1}\}$ and set of demands $D = \{ d_{ij} : i<j, d_{ij} > 0\}$. Each edge from set E which connects nodes n_i and n_j where $j = i + 1 \bmod N$, has weighted value $C(e_{ij})$, that represent amount of traffic load passing through this edge. Each demand from set D, represents the amount of traffic that has to be carried from node n_i, to node n_j has a routing variable $x(d_{ij})$, $0 \leq x(d_{ij}) \leq 1$. Interpretation of routing variable $x(d_{ij})$ is as follows – if $x(d_{ij}) = 1$ then demand is routed clockwise, if $x(d_{ij}) = 0$ then demands is routed counter-

clockwise. When demand splitting is allowed, variable $x(d_{ij})$ can have any value between 0 and 1, which representing percentage of demand d_{ij} carried in clockwise direction. In other words, amount $d_{ij}*x(d_{ij})$ is routed clockwise, and amount $d_{ij}*(1 - x(d_{ij}))$ is routed counter-clockwise.

In order to mathematically define link load, two subsets of set D for each edge are defined. Set $A(e_{ab})$ is defined as :

$$A(e_{ab}) = \left\{ \forall d_{ij} \in D : (i \le a) \cup (j \ge b) \right\}$$

and set $B(e_{ab})$ is defined as:

$$B(e_{ab}) = D \setminus A(e_{ab})$$

Let us explain these set definitions. Set $A(e_{ab})$ includes all the clockwise routed demands from D that are passing edge e_{ab}. Set $B(e_{ab})$ includes all the other demands.

Required capacity of each edge is calculated as the sum of all traffic routed through that edge:

$$C(e_{ab}) = \sum_{d_{ij} \in A} d_{ij} \cdot x(d_{ij}) + \sum_{d_{ij} \in B} d_{ij} \cdot (1 - x(d_{ij}))$$

Ring capacity (RC) is defined as the largest edge capacity. All the edges in the ring are assigned the same capacity, which is equal to RC, or any higher value available on the market (standard SONET/SDH transmission capacities).

$$RC = \max_{e_{ab} \in E} C(e_{ab})$$

In Figure 2 process of simple ring dimensioning is shown. It can be seen, how demands given in Table 1 are routed (without splitting), and how edge and ring capacity is calculated. Optimal routing solution for this example is as follows: demands d_{14} and d_{36} are routed counter-clockwise, while all other demands are routed clockwise. Capacity of each edge is calculated by summing up all the demands routed through this edge. In this example edge e_{45} has the largest capacity and therefore ring capacity required for this kind of routing is $C(e_{45}) = 29$.

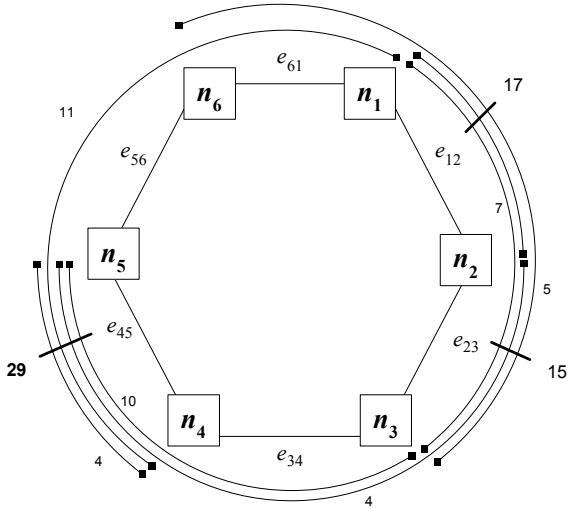

Figure 2: Simple ring dimensioning example

Table 1: Traffic demands for simple ring dimensioning example

Src.	Dest.	Demand
n_1	n_2	5
n_1	n_3	7
n_1	n_4	11
n_2	n_5	4
n_3	n_5	10
n_3	n_6	5
n_4	n_5	4

It is clear, that, different solutions are possible for ring dimensioning, depending on routing of demands. But, since the ring cost is increasingly proportional to ring capacity, it is reasonable to find smallest possible ring capacity that could satisfy all the demands.

By finding smaller ring capacity, the cost of the ring is decreased. This is a reason why RLP is defined as a problem of finding minimum ring capacity.

In order to find an optimal solution for ring dimensioning we should alternate all the possible routings for all demands, and pick the solution giving the best overall ring capacity. That is not a problem for smaller ring networks, but when we have to dimension larger rings such processes could take a lot of time, which is often not acceptable. Therefore, different fast algorithms for solving RLP are developed. It is not always the case that they are providing an optimal solution, but all the solutions are near-optimal solutions, so that those algorithms can be very efficient when used in practice.
For obtaining optimal solutions linear programming approaches can be used – defined problem can be easily formulated as linear program, and solved using available software tools for mathematical programming, such as AMPL/CPLEX [6].

2.1. RLPWO

When demand splitting is not allowed, all the demands can be routed either clockwise or counter-clockwise. In our problem formulation routing direction variable $x(d_{ij})$ can be equal to 1 if demand d_{ij} is routed clockwise, or 0 if demand d_{ij} is routed counter-clockwise. Each demand can be routed in only one of two possible directions, which makes the total number of possible routing solutions $2^{|D|}$, where $|D|$ is number of demands. If we suppose maximum number of demands to be $\binom{n}{2}$, which is the case where all possible demands d_{ij} ($i<j$) are given, we have total of $2^{\binom{n}{2}}$ routing solutions. Our goal is to find the best routing solution out of all possible routing solutions.

Simplest, but slowest, and in practice not applicable algorithm, is the one that explores all the possible solutions, and chooses the one with the best result. Such exact algorithm can be used for smaller rings in order to value results obtained using other algorithms. However, it is questionable if usage of such algorithm is reasonable at all.
Another approach for finding optimal RLPWO solution would be the one using linear programming. It was shown that such process is NP-complete in case where only one demand is defined between pair of nodes [1]. After presenting RLP as a linear program, optimal results can be calculated using available commercial tools like AMPL/CPLEX. RLPWO is a special case of RLP where variables ($x(d_{ij})$) can only take the values 0 or 1. Both described solutions are not very effective for practical usage. First solution is very slow but easy for implement, while second solution is faster but unpractical and more complex for implementation.

It is obvious that there is a need for algorithms that should provide all the expected features – they should be relatively fast, easy to implement and finally, solutions obtained using them should be as near as possible to optimal solutions. Basically, there are two common approaches for all the algorithms – greedy approach and weighted approach. As its name says, greedy algorithms are quite myopic, and they consider only temporary optimal solutions. They do not consider a possibility that temporary non-optimal solution could give better final solution than temporary optimal solution does. Therefore, there are no guaranties that solutions obtained using greedy algorithm will be near optimal – there are cases where solutions are exact the same as optimal solution, but also there are cases where solutions are not even near the optimal solution.
Unlike greedy algorithms, weighted algorithms define a certain weight for each edge depending on different criterion. Weighted algorithms generally give better solutions than greedy algorithms. Example of weighted algorithm is DualAscent heuristic described in [1].

In the following example of greedy algorithm, we will calculate an optimal routing for simple ring example introduced in previous section. We are given seven demands (Table 2) defined on ring built of six nodes. Simple greedy algorithm will be used.

Table 2: Sorted demands of gready algorithm example

Src.	Dest.	Demand
n_1	n_4	11
n_3	n_5	10
n_1	n_3	7
n_3	n_6	5
n_1	n_2	5
n_2	n_5	4
n_4	n_5	4

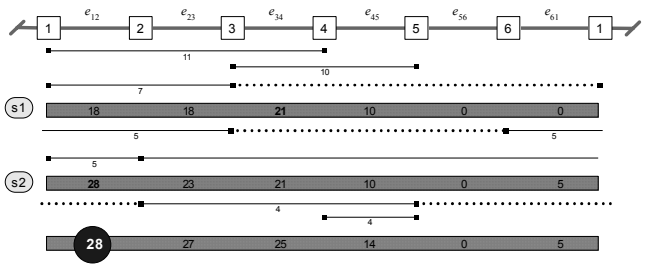

Figure 3: Example of greedy RLP approach

First, sort all demands in decreasing order. The largest demand (first in sorted list) will be routed in direction where it passes the minimal number of edges. After that, other demands are routed in such manner that increment in ring capacity is smaller than if the demand has been routed in another direction. If the ring capacity increment is the same regardless routing direction, then some additional conditions are considered. In this case, let us route a demand so that smaller number of edges will be used. As depicted in Figure 3, there are two decisions to be made – *s1* and *s2*. *s1* occurs when demand d_{36} is being routed. If demand would be routed clockwise maximum edge capacity would be at e_{34}, equal to 26. But, if we route it counter-clockwise maximum edge capacity is 23, at edges e_{12} and e_{23}. Next demand, d_{12} is routed clockwise – if it would be routed counter-clockwise, maximum edge capacity would be the same, but here decision was made based on smaller number of edges influenced if demand is routed clockwise. Another decision *s2* is to be made when d_{25} is routed. When routed clockwise it does not impact edge e_{12}, so that ring capacity after finishing dimensioning is equal to 28.

The question that arises here is, how would different sorting of demands impact final solution? What if demand d_{12} would be routed before d_{36}, since both demands have the same traffic amount? In that case, ring capacity would decrease and at the end of dimensioning its value would be 27. It is obvious, that greedy algorithm is not reliable, because it depends on many parameters. For this reason we have implemented many greedy 'sub-algorithms' that take into account different additional considerations. In our example, even optimal solution can be found (Figure 4) – if first demand (d_{14}) is routed counter-clockwise instead of clockwise, it is possible to route other demands based on the same principles and to achieve optimal ring capacity value 25. In this routing solution, demands d_{14}, d_{36} and d_{45} are routed counter-clockwise, and other demands are routed clockwise.

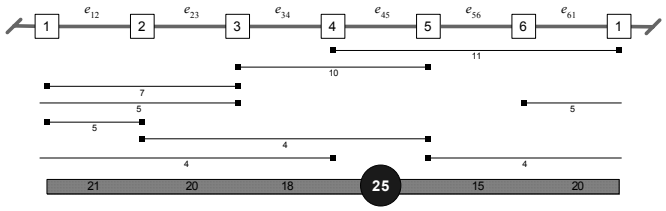

Figure 4: Greedy RLP approach with different initial routing

One could notice, that greedy solutions are not very efficient and small change in algorithm can produce totally different solution. For this reason, many optimized algorithms were developed by different authors [1][2][3][4][5], each of them giving excellent results in certain scenarios.

All algorithms were also implemented and compared in our software tool, and obtained solutions are discussed later.

One approach in finding RLPWO routing solutions is based on solutions obtained from algorithms used for RLPW. In practice (see results section) algorithms that use RLPW solutions in order to find RLPWO solution were found very useful and very effective. One of them is 2-OPT developed by [3], that is based on EXACT RLPW algorithm.

2.2. RLPW

Ring loading problem with demand splitting can be formulated as a relaxation of RLPWO. When RLPW is defined, linear program is the same as for RLPWO, with exception of definition of variable $x(d_{ij})$. Different than in RLPWO, variable $x(d_{ij})$ can take any value from 0 to 1 (including 0 and 1). Traffic portion $d_{ij}*x(d_{ij})$ will be routed clockwise, and $d_{ij}*(1 - x(d_{ij}))$ will be routed counter-clockwise.

Intuitively, RLPW looks more complex than RLPWO, because there are more possible solutions. For instance if we look back on greedy RLPWO approach example, we can see that for each demand one-step forward examination is made whether routing in one direction gives better results (lower ring capacity) than routing in another direction. But, since we know temporary edge capacities, we can determine how much additional traffic can be routed through each edge without unnecessary increasing ring capacity. After calculating that traffic amount we can easily determine $x(d_{ij})$. Of course, additional constraints can be set, for instance only certain splitting modes can be allowed (integer splitting) etc.

Fast and exact algorithms for finding optimal RLPW are much like greedy algorithms. Simply, temporary ring capacity is explored and demands are split so that new ring capacity has smallest increase. Example of such algorithm is EXACT algorithm described in [3].

3. Problem solutions

Previous section gave an introduction and principle description of both RLP problems arising. Now, let us make a short overview of different algorithms developed by other authors, including simple descriptions. All described algorithms are implemented and included in our software tool called RingSolver (Figure 5). This tool imports data about ring structure (nodes, edges, demands) and calculates ring capacity using different algorithms, providing additional results analysis.

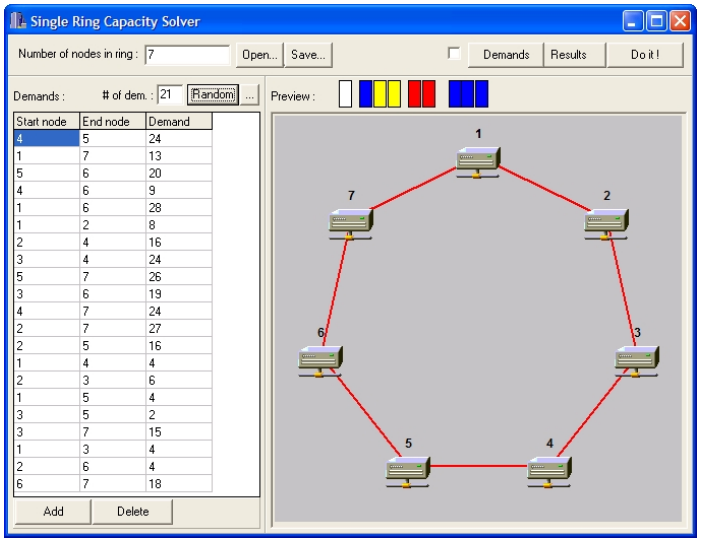

Figure 5: RingSolver tool - main screen

All the solutions got using this tool, together with solutions calculated based on linear program definition of RLP, are discussed later in next section.

As already explained, greedy algorithms for RLPWO can provide majority of different solutions, depending on mechanisms used in cases where no direct decision can be made. Our tool includes several greedy algorithms.

Since all of them are quite fast, all the calculations are always performed and best solution is picked to represent greedy solution. Greedy algorithms described in [1] are also implemented.
For solving RLPWO two advanced algorithms are included - DualAscent algorithm as a weighted approach algorithm [1], and 2-OPT algorithm [3]. Algorithm 2-OPT is based on RLPW solution obtained using EXACT algorithm. It is very effective and fast.

For solving RLPW two algorithms are included - EXACT algorithm, and INDES algorithm [2]. While EXACT provides exact solution without constraints, INDES is used to find integer splitting demand solution. As it can be seen from results, solutions obtained using them are almost the same.

In order to find a solutions where all edges in ring network have almost the same utilization few Split algorithms are implemented. It can be seen that solutions they provide are not even near optimal, but all edges are almost equally utilized, meaning that traffic flow through all of them approximately the same. Optimized solutions often provide solutions in which some edges are under-loaded.

4. Results

In this section the results obtained by using all the previously described algorithms are presented. Results are compared with results calculated using AMPL/CPLEX tool for solving mathematical problems. Also, on smaller rings optimal solution for RLPWO is calculated using slow 'algorithm' that explores all the possible solutions. All the algorithms are implemented and executed in our software tool RingSolver (Figure 6). Traffic demands are supposed to have uniform distribution, within predefined minimal and maximal values. The obtained results are shown in Table 3 and Table 4. Table 3 has three sections. First section shows results for small rings, those with number of demands equal to the number of nodes. In second section of the table the results for middle-size rings are shown. Middle-size ring are those with number of demands approximately two times smaller than maximal number of demands, which is $\binom{n}{2}$, where n is number of nodes in the ring. In third section results for large rings with maximal possible number of demands, are shown. For each ring, all the calculations were made 10 times. Demands were chosen randomly with uniform distribution in interval between 1 and 30. Average value of these 10 calculations is shown in the table.

Figure 6: Results window of RingSolver software tool

First column in the table represents the number of nodes and edges in the ring. Remaining columns contain results calculated using described algorithms. Column 3 shows the results obtained by using AMPL/CPLEX tool. These results are optimal for RLPWO, and are used for validation of other results. RLPW results are also shown. Since we have different versions of Greedy and DualAscent algorithm implemented in our tool, where each version gives slightly different results in different scenarios, in the table are shown only the best results from each class of algorithms.

Let us discuss results shown in Table 3. It is obvious that both RLPW and RLPWO rings have smaller capacity than USHR rings. It does not

necessarily means that they are cheaper than USHRs, because USHR equipment is usually cheaper than equivalent BSHR equipment.

Table 3: RLP results obtained with RingSolver and AMPL/CPLEX

		RLPWO				RLPW	
N,K	USHR	CPLEX	Greedy	DAscent	2OPT	EXACT	INDES
Small rings							
5,5	57,1	41,8	41,8	45,2	42,9	36,4	36,6
10,10	107,8	66,5	67,4	78,4	70,6	61,5	61,9
15,15	143,6	90,2	92,6	103,9	93,3	88,3	88,6
20,20	181,6	114,0	118,4	139,0	119,1	111,7	112,1
40,40	329,0	192,6	207,0	240,7	198,3	189,6	189,7
Middle-size rings							
5,8	79,3	50,4	50,8	52,4	51,1	45,4	45,7
10,22	209,6	120,2	128,8	139,0	122,3	116,4	116,8
15,52	468,8	255,6	273,7	283,7	263,8	255,8	256,1
20,95	819,5	442,2	487,3	481,0	449,2	439,9	440,5
40,390	3239,6	**	1821,7	1793,8	1717,9	1706,3	1706,6
Large rings							
5,10	96,2	59,3	59,8	61,7	60,7	53,6	54,0
10,45	418,6	218,2	238,6	233,5	222,6	215,1	215,4
15,105	917,4	486,2	504,5	504,5	494,3	484,1	484,6
20,190	1575,6	821,6	878,3	858,9	830,1	820,5	820,8
40,780	6201,4	**	3303,7	3250,5	3189,4	3177,0	3177,5

** Result cannot be obtained with a student version of AMPL, which is limited to 300 variables

RLPWO algorithms provide different solutions, and we cannot point out any of them saying it is always providing the best solutions. But, based on the results shown here, some rules can be determined. Greedy algorithms provide best solutions only with the small rings with small number of demands. In other cases 2OPT algorithm provides the best solution - solutions vary from 1% to 6%. It is generally because it is based on the EXACT algorithm and then degrades the result by eliminating possible demand splitting. DualAscent approach can be used in some cases, but only because it his complexity smaller than 2OPT algorithm. DualAscent can improve solutions calculated using greedy algorithms, especially in the rings with more demands.
RLPW algorithms are optimal - EXACT gives exactly optimal solution, while INDES uses only integer splitting, therefore providing slightly degraded solution from EXACT.

In Table 4, results for demands distributed uniformly between 1 and 100 are presented. All the regularities appearing in Table 3 are even more obvious in Table 4. DualAscent algorithm on large networks gives better solutions than simple Greedy algorithm.

Table 4 RLP results for bigger demands values

	RLPWO		
N,K	Greedy	DAscent	2OPT
20,20	404,7	439,9	401,8
10,22	397,6	426,2	394,6
15,52	885,6	914,9	834,3
20,95	1548,8	1560,9	1443,8
10,45	753,0	734,2	706,6
15,105	1574,5	1545,1	1545,1
20,190	2877,2	2708,1	2708,1

5. Conclusion

In this article an overview of different methods used for solving single ring dimensioning problem is presented. All the algorithms discussed here are implemented in our software tool called, RingSolver, and are being used in

another tool for designing more complex ring-networks, which is still under development. Since, majority of algorithms are relatively simple to implement, and they provide very good solutions in short execution time it is reasonable to use them when planning and dimension larger ring-based networks. Such networks can contain huge number of rings and therefore it is very important to have algorithms that can handle problems fast and effective.

Additional research and improvement of algorithms is required, since it is obvious from the results that there is no universal algorithm, which is superior in all scenarios. Even though results are quite close to optimal (within 6%), still no RLPWO algorithm can reach the results obtained by AMPL/CPLEX. For the RLPW case, EXCAT algorithm gives optimal results in very short execution time.

Literature

[1] S.Cosares, I. Saniee, "An optimization problem related to balancing loads on SONET rings", Telecommunication Systems, 3 (1994), pp. 165-181.

[2] C.Y.Lee, S.G.Chang, "Balancing loads on SONET rings with integer demand splitting", Computers Ops. Res., Vol. 24, No. 3. pp. 221-229, 1997.

[3] Y-S.Myung, H-G.Kim, D-W.Tcha, "Optimal load balancing on SONET bidirectional rings", Operations Research, Vol. 45, No.1, January-February 1997.

[4] Wu, T.H, "Fiber Network Service Survivability", Artech House, Massachusetts, 1992.

[5] A. Schrijver, P. Seymour, P. Winkler, "The Ring Loading Problem", SIAM J. Descrete Math., Vol. 11, No. 1, pp. 1-14, February 1998.

[6] AMPL: A Modeling Language for Mathematical Programming, www.ampl.com