Research Article

ILP Model and Relaxation-Based Decomposition Approach for Incremental Topology Optimization in $p$-Cycle Networks

Md. Noor-E-Alam,1 Ahmed Zaky Kasem,2,3 and John Doucette1,3

1 Department of Mechanical Engineering, University of Alberta, 4-9 Mechanical Engineering Building, Edmonton, AB, Canada T6G 2G8
2 Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada T6G 2V4
3 TRLabs, Edmonton, AB, Canada T5K 2M5

Correspondence should be addressed to John Doucette, john.doucette@ualberta.ca

Received 14 May 2012; Accepted 7 October 2012

Academic Editor: Maode Ma

Copyright © 2012 Md. Noor-E-Alam et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

$p$-cycle networks have attracted a considerable interest in the network survivability literature in recent years. However, most of the existing work assumes a known network topology upon which to apply $p$-cycle restoration. In the present work, we develop an incremental topology optimization ILP for $p$-cycle network design, where a known topology can be amended with new fibre links selected from a set of eligible spans. The ILP proves to be relatively easy to solve for small test case instances but becomes computationally intensive on larger networks. We then follow with a relaxation-based decomposition approach to overcome this challenge. The decomposition approach significantly reduces computational complexity of the problem, allowing the ILP to be solved in reasonable time with no statistically significant impact on solution optimality.

1. Introduction

High-availability networks have become integral to our everyday lives, used for banking, financial transactions, voice and data communications, entertainment, and so forth. While much effort has been made to make them as reliable as possible, failures and, more critically, service outages still occur with alarming frequency. The vast majority of such failures are a result of fibre cuts, with most of those failures due to cable digups and similar construction accidents [1].

As the frequency of failures has increased, researchers have developed many approaches for ensuring survivability of the network even in the face of cable cuts or other equipment failures, including a number of mechanisms that allow the network to actively respond to a failure by rerouting affected traffic onto one or more backup routes. Survivability mechanisms are often thought of as being either restoration or protection [2]. Although the differences between the two are often blurred, and some mechanisms can be considered to be either type, the general idea is that restoration techniques are those in which a backup route is formed after failure, while protection techniques are those in which a backup route is formed before failure. Each individual survivability mechanism has its own advantages and disadvantages and requires differing amounts of spare capacity distributed throughout the network to accommodate backup routes.

One particular protection mechanism that has received a lot of attention in recent years is $p$-cycles [3, 4]. A $p$-cycle is a cyclic structure of preconfigured spare capacity, as illustrated in Figure 1, for example, where a six-span $p$-cycle is shown in the leftmost panel. In the event of failure of an on-cycle span (a span that is a part of the cycle), traffic on the failed span can be rerouted around the surviving portion of the $p$-cycle, as illustrated in middle panel. Each unit of capacity on the $p$-cycle can be used to protect one unit of capacity on each on-cycle span. However, in the event of failure of a straddling span (one whose end nodes are on the $p$-cycle but which is itself not a part of the $p$-cycle), traffic on the failed span can be rerouted around the $p$-cycle in either direction, as illustrated in the rightmost panel. In this case, each unit of capacity on the $p$-cycle can be used to protect two units of...
capacity on each straddling span of the \( p \)-cycle. So a single unit capacity copy of the \( p \)-cycle in Figure 1 (i.e., comprising a single unit of protection capacity on each of the six spans of the \( p \)-cycle) is capable of protecting a total of twelve units of capacity, one on each of the six on-cycle spans and two each on the three straddling spans.

As a special case of \( p \)-cycle protection, one can use Hamiltonian cycles to provide protection (a Hamiltonian cycle is a cycle that passes through all nodes in the network) [5, 6]. In the most extreme case, a single Hamiltonian cycle of sufficient size can actually protect all single-span failures in the network, as all spans will either be an on-cycle span or a straddling span for that cycle. While this can result in simplified switching scenarios (nodes need only a switch to a single \( p \)-cycle, rather than localise the failure and make switching decisions for multiple cycles independently), it is merely a special case of \( p \)-cycle protection. The interested reader can refer to prior literature in [5–7].

Subsequent work in the literature has extended the \( p \)-cycle concept to node-encircling \( p \)-cycles (NEPC) [4], path-segment protecting \( p \)-cycles [8], failure-independent path-protecting (FIPP) \( p \)-cycles [9], and two-hop segment node protection \( p \)-cycles [10]. There has also been a number of network design approaches developed in the past decade. Such approaches can be classified as either spare capacity allocation (SCA) or joint capacity allocation (JCA) approaches. In the typical SCA approach, working routing is performed in advance, often via shortest path routing (but not necessarily so), and restoration routing is performed afterwards through some optimal or near optimal method. The objective is to minimize spare capacity requirements such that the network is 100% restorable to single-failure events [11, 12]. In JCA approaches, working routing is performed hand-in-hand with restoration routing; working and spare capacity is jointly minimized such that the network is 100% restorable to single-failure events [13, 14]. In general, JCA is significantly more capacity efficient than SCA. While there have been several heuristic methods developed to design \( p \)-cycle networks [15], we will focus on integer linear programming (ILP) techniques, which can be classified as either arc-path or node-arc models. Arc-path models derive from the original span-restorable network design model in [16], where working and restoration routing is carried out over a set of preenumerated eligible routes. In contrast to the arc-path approach, node-arc models (also called transhipment models) do not utilize eligible restoration routes [17], rather they are assembled on the fly by the ILP solver.

In addition to \( p \)-cycles, many other survivability schemes have been proposed in the literature, including various automatic protection switching (APS) techniques, span restoration, path restoration, and shared backup path protection (SBPP). Comparative analyses were carried out in [2] to compare and contrast major restoration and protection schemes in terms of capacity efficiency. Although it is perhaps the simplest form of protection available, 1 + 1 APS was shown to require considerably greater spare capacity than the other schemes. Span restoration and \( p \)-cycle protection are the next most costly survivability schemes, with span restoration generally serving as a lower bound on \( p \)-cycle capacity requirements (in fact, \( p \)-cycles can be shown to be a special case of span restoration). The end-to-end protection schemes (path restoration and SBPP) are the most capacity efficient, requiring the least amounts of spare capacity. In extreme cases, path restoration can approach zero spare capacity requirements because of stub release (releasing surviving portions of failed working paths to be reused as spare capacity), [2].

In most of the work in the literature, the underlying network topology is known in advance, but there have been several approaches developed that include at least some aspects of an unknown or variable topology in the network design process [17–20]. In [19] and [21], the design methods for tree topologies optimization in communication and data networks did not consider restoration or reliability. In [18, 20–22] biconnected network topologies were considered as a transition from tree topologies. In [17–20], survivability itself was included in the design approach. In these approaches, fixed costs are typically associated with establishment of a span as well as with placement of working and spare capacity on those spans. Fixed establishment costs represent rights-of-way and lease acquisitions, excavation, duct installation, amplifiers, and so forth that are not generally dependent on the capacity or bandwidth of the spans.

Relatively little effort has been made on the investigation of the incremental topology optimization problem. Therefore, the goal of the present work is to develop a JCA \( p \)-cycle network topology optimization ILP formulation that will minimize the overall design cost (capacity and fixed span establishment costs) of a \( p \)-cycle network along with...
its underlying topology such that all single span failures are restorable. Due to the significant computational complexity of this problem (as will be discussed later), we will consider only incremental topology design, where a preexisting initial topology already exists but which is amended through span additions. Even this less complex problem becomes intractable for large networks, and so we further develop a problem-specific relaxation-based decomposition technique to solve this large scale ILP.

The outline of this paper will be as follows: an ILP formulation for topology optimization $p$-cycle design problem is developed in Section 2. It also discusses our experimental setup (i.e., solution approach). Section 3 describes our ILP results. Our problem-specific relaxation-based decomposition technique is proposed and described in Section 4, with results summarized in Section 5. Finally, we wrap up the paper in Section 6 with a concluding discussion.

2. ILP Model

In this section, we present and develop our ILP formulation for incremental topology optimization for $p$-cycle network design problem. Prior to topology optimization ILP models generally make use of the node-arc approach as enumeration of eligible restoration routes becomes a challenging combinatorial problem when the underlying topology is not known; a separate set of eligible routes is needed for every combination of selected eligible spans. And while we will utilize a node-arc approach for our ILP with respect to working routing, our overall approach will be a hybrid, with the $p$-cycle selection placement done via an arc-path approach (i.e., selection from amongst an enumerated set of eligible $p$-cycles). There has been a few notable works in recent literature that develop methods for $p$-cycle network design without enumeration of eligible cycles, [23, 24], but these approaches have proven challenging to incorporate into our topology-optimization ILP. In order to formulate our ILP model, we first define the following notation:

$N$: set of all nodes in the network topology, indexed by $n$ or $m$,

$P$: set of all $p$-cycles in the network topology, typically indexed by $p$,

$S$: set of all spans in the network topology, typically indexed by $i$ or $j$. This includes eligible spans as well as existing spans,

$S_n$: set of all spans incident on node $n$, indexed by $i$ or $j$,

$Q$: set of all eligible spans that can be added to the network, indexed by $i$ or $j$,

$D$: set of all demands in the network, indexed by $r$,

$d_r$: the parameter that represents the number of demand units for demand $r$,

$O_r \in N$: the origin node of demand $r$,

$T_r \in N$: the target node of demand $r$,

$c_j$: the incremental cost of adding one unit of capacity on span $j$,

$f_i$: the fixed establishment cost for eligible span $i$,

$x_{ij,p} \in \{0,1,2\}$: a parameter that enumerates eligible $p$-cycles by representing the relationship between span $j$ and $p$-cycle $p$, where $x_{ij,p} = 2$ if it is a straddling span, $x_{ij,p} = 1$ if it is an on-cycle span, and 0 otherwise,

$w_{n,j}^r \geq 0$: a decision variable for the number of working capacity units assigned for demand $r$ on span $j$ and flowing out from node $n$,

$w_{j,n}^r \geq 0$: a decision variable for the number of working capacity units assigned for demand $r$ on span $j$ and flowing into node $n$,

$w_j \geq 0$: an integer decision variable for the total amount of working capacity units assigned to span $j$,

$s_i \geq 0$: an integer decision variable for the number of spare units deployed on span $j$,

$\delta_i \in \{0,1\}$: a binary decision variable that equals 1 if the eligible span $i$ will be used in the design, and 0 otherwise,

$n_p \geq 0$: an integer decision variable that represents the number of copies of $p$-cycle $p$ that will be used in this design,

$M$: a large number (in our case, the summation of all demands plus one).

Note that strictly speaking, the $w_{n,j}^r \geq 0$ and $w_{j,n}^r \geq 0$ decisions variables are integer variables. However, as was shown in [17], as long as the capacity variables themselves are integer, integrality can be relaxed on the underlying flow variables. We then define the problem as follows: minimize

$$\sum_{j \in S} c_j (s_j + w_j) + \sum_{i \in Q} f_i \times \delta_i \quad \forall i \in Q, \forall j \in S. \tag{1}$$

subject to

$$\sum_{j \in S_n} w_{n,j}^r = d_r \quad \forall r \in D, \forall n \in N \mid n = O_r, \tag{2}$$

$$\sum_{j \in S_n} w_{j,n}^r = 0 \quad \forall r \in D, \forall n \in N \mid n = O_r, \tag{3}$$

$$\sum_{j \in S_n} w_{j,n}^r = d_r \quad \forall r \in D, \forall n \in N \mid n = T_r. \tag{4}$$
The objective function in (1) seeks to minimize the total cost of the network, including the variable costs incurred for placing working and spare capacity on all spans, and the fixed costs incurred by adding any additional spans to the existing topology (i.e., selecting one or more of the eligible spans). Equations from (2) to (9) are the node-arc constraints that determine working routing and working capacity placement, similar to the approach in [17]. The constraints in (2) ensure that, for any demand, the total number of working capacity units flowing out from the origin node must equal the number of demand units for this demand, while constraints in (3) ensure that all network flows into the origin node for a particular demand equal zero. Equations (4) and (5) are the related target node constraints. The constraints in (6) ensure the conservation of flow requirement for all transshipment nodes (i.e., not the origin or target nodes) for each demand, while constraints (7) and (8) ensure conservation of flow for all spans (i.e., any traffic flow into a span for a particular demand equals the flow out of that span for that demand). Equation (9) guarantees that the total number of working capacity units deployed on any span will be sufficient to accommodate all of the working traffic routed through it.

Equations (10) and (11) are the arc-path $p$-cycle placement constraints like those in the original $p$-cycle paper, [26]. Constraints in (10) ensure that, for each failed span, the total number protection routes available from $p$-cycles deployed in the network will be sufficient for restoring the working capacity on each span; each copy of a $p$-cycle copy can restore one working capacity on each of its on-cycle spans and two units of working capacity on each of its straddling spans. Constraints in (11) place sufficient spare capacity to accommodate all deployed $p$-cycles. Finally, the constraints in (12) force all span selection variables to equal one if the associated span is assigned any working and/or spare capacity.

3. Experimental Methodology

We used a set of seven test case networks of 10 nodes, 15 nodes, 20 nodes, 25 nodes, 30 nodes, 35 nodes, and 40 nodes. The base networks we used herein (i.e., defining the existing topologies) are the most sparse members of the network families from [2], while their so-called master networks (i.e., those with average nodal degree $d = 4.0$) represent the set of eligible spans for each of our respective networks. The set of demands for each of those networks were also used herein; each node pair in a network exchanges a number of lightpaths drawn from a uniform random distribution. While one might argue that demands in reality are not known in advance with any precision and are not static, this treatment of demands is common in the literature, as the demands used can represent upper limits on the expected demands.

Eligible $p$-cycles were enumerated via a custom-designed C++ algorithm that performed a depth-first search type of algorithm to enumerate at least the shortest 10 thousand possible cycles that can be drawn in the graph to protect each single span failure, including eligible spans.

We solved all instances of the problem on an 8-processor ACPI multiprocessor X64-based PC with Intel Xeon CPU X5460 running at 3.16 GHz with 32 GB memory. The ILP models were implemented in AMPL [27] and solved with the CPLEX 11.2 solver [28]. We used a CPLEX mipgap setting of 0.001, which means that all test cases solved to full termination are provably within 0.1% of optimality.

4. Preliminary Result Analysis

Figures 2, 3, 4, 5, 6, 7, and 8 show the relationship between total network design cost and the number of eligible restoration routes with various establishment cost multipliers. Each square, diamond, and triangular data point represents the normalized total cost (working and spare capacity plus fixed span establishment costs) of the network indicated with the specified number of eligible spans and with the specified span establishment cost multiplier. The cost multiplier is the ratio of the spans’ fixed establishment cost to its per-unit capacity cost (i.e., it equals $f/c_i$); the same cost multiplier is applied uniformly on all spans in the network. In our case, we used cost multipliers of 10, 20, and 50, denoted in the charts as low, medium, and high, respectively. We remind the reader that the fixed establishment costs represent rights-of-way costs associated with the span’s fibre facility route, installation of the conduit and fibre cables, and all other one-time costs that might be incurred to establish a new span. The network design cost curves for the medium and high establishment cost factors are not shown for the three larger networks, as problem
complexity becomes exceedingly problematic for these test cases (see further discussion below).

As we expect, the ILP model is better able to perform working and restoration routing and allocate the associated working capacity and $p$-cycles as we introduce more eligible spans, so that overall capacity costs are reduced as the eligible span set gets larger. The rate of the cost reductions varies from network to network, but the trend appears to be that cost reductions slow as the number of eligible spans becomes large. The interpretation here is that as we provide the network with a greater and greater number of eligible spans to select from, it becomes more difficult for the network to make use of these eligible spans.

We can also note that the establishment cost factor does not appear to have a significant bearing on the behaviour of the relationship between network costs and the number of eligible spans. For each network, the differences between the three curves themselves (corresponding to the low, medium, and high establishment cost factors) is primarily due to the fact that the sum of the selected spans’ fixed costs will be larger with a higher establishment cost factor (i.e., the second summation in the objective function), irrespective of the actual number of selected spans. In addition, as will be discussed later, the differences between the three curves are partially a function of the differences in the spans selected by the solver for the various cost factors. However, since the higher establishment cost factors generally result in selection of fewer eligible spans (see the discussion below), this will have a negative effect on the design costs at higher establishment costs. The total network design costs tend to become closer (i.e., the differences between them become less, relatively speaking) as the networks become larger, though this is primarily due to the fact that the capacity costs represent a proportionally greater share of the overall network cost as the networks become larger. In hindsight, this suggests that perhaps our establishment cost multipliers are likely too small to adequately demonstrate the effect that is seen in smaller networks. This should not be interpreted as suggesting that the objective function itself is flawed, rather, there will be a degree of uncertainty in establishment cost factors that will need to be selected based on observed (i.e., actual) costs and perhaps also artificially through a desire to drive rich or sparse topologies (through low cost multipliers and high cost multipliers, resp.).

In any case, the ILP effectively permits a network designer to select an optimal set of span additions (i.e., incremental topology optimization) on which to design a $p$-cycle network. Strictly speaking, this problem is NP-hard [29, 30], but like many NP-hard problems, specific instances are solvable in reasonable times. That is the case for small instances of this problem. However, we can also observe in Figure 2 through Figure 8 that the solution runtimes become prohibitively high for large test case network instances and
eligible spans for the 35-node network.

Figure 6: Total network costs and CPU time versus the number of eligible spans for the 30-node network.

Figure 7: Total network costs and CPU time versus the number of eligible spans for the 35-node network.

generally also increase with the number of eligible spans provided to a network. In those figures, the curves with cross (×) data points (read against the right hand side y axes) show the runtime required by the solver when running the corresponding low establishment cost factor results to optimality. Each data point represents the actual processor time used in total amongst all 8 processors (as recorded by CPLEX) to solve the ILP for the indicated test network with the indicated number of eligible spans using the low establishment cost factor (though in most cases, only a single processor was utilized).

As one can observe, runtimes are quite short (from fractions of a second to a few minutes) for the smaller test case networks, but become exceedingly high for larger test case networks reaching nearly 200 thousand seconds (more than two days) for the 40-node network with 78 eligible spans. While there is a general increasing trend in runtimes as we provide a greater number of eligible restoration routes, we can notice that they often exhibit an irregular nature. Although it would be interesting if some useful insight could be gained from this observation, the cause is simply due to peculiarities in the network topologies and the nature of the solution approach. For instance, when the 15-node test case network is solved with 5 eligible spans (Figure 3), inclusion of that 5th eligible span results in the enumeration of a specific set of eligible p-cycles which happens to be more computationally complex to solve than the test case with only 4 eligible spans or with 6 eligible spans. It might also be interesting to note that the number of branch-and-bound nodes produced by CPLEX’s internal algorithm rises quite substantially in test cases corresponding to those instances with irregularly high runtimes, suggesting that simple peculiarities in the branch-and-bound tree contribute to these high runtimes. We suspect that the highly irregular nature of CPU times for those test-case networks was due to a complex interaction of the large number of spans in the network and topological effects (addition of a single span can often provide an obviously beneficial routing option that the solver takes advantage of). Such instances of the problem can create much tighter LP relaxations than other instances, and/or algorithms used by CPLEX’s internal branch-and-bound procedures might be better suited to some of those specific cases. As a result, these instances see fewer branch-and-bound nodes when solving the ILP problem. It is these artifacts (i.e., the irregular nature of the runtime increases) in smaller to midsize test case networks, and, more importantly, the extremely high runtimes in the large test case networks that motivate us to develop an alternative solution method for the p-cycle network topology optimization problem, as discussed later in Section 5.

A closer look at the numbers of spans selected by the solver adds some additional interesting insights. As we can observe in Figures 9, 10, 11, and 12, the number of spans selected decreases quite as we increase the span establishment cost factor (i.e., as spans become more expensive, relative to the per-unit capacity costs). However, for larger networks there is almost no variation initially (i.e., as we provide only a few eligible spans), regardless of span establishment factor, but a small degree of variation arises when we provide a greater number of eligible spans. While this may initially seem indicative of some underlying phenomenon, the truth of the matter is that we happened to have selected span establishment factors that produce a lopsided objective function that is dominated by the span capacity costs in the larger networks. In hind sight, a smarter approach would have been to set higher stand establishment factors for these larger networks, so that the objective function is more...
balanced, with respect to the span capacity costs and the fixed establishment costs. With the span establishment factors we have used, the solver sees little disincentive to select quite a large number of the eligible spans (i.e., there is only a small cost to add extra spans, relative to the reductions in capacity that result).

5. Relaxation-Based Decomposition Technique

In order to be better able to solve the \( p \)-cycle network topology optimization problem in large test case networks, we now propose and develop a problem-specific relaxation-based decomposition technique for the ILP developed above. From the investigation of a hard ILP instance, it is sometimes observed that the computational complexity arises from a set of constraints or integrality properties of specific sets of variables. For the first scenario, we can dualize these hard constraints and create an easy subproblem [25, 31], and the solution of this subproblem can be used to solve the main problem. Our proposed technique is different from this approach in a sense that we decompose the original problem into two easy subproblems by relaxing the integrality property of some variables rather than relaxing the set of constraints. While most advanced solvers, including CPLEX, utilize some form of relaxation-based approaches to speed up solution of ILP problems, such general approaches often have difficulty properly selecting the best specific relaxations and subproblem decompositions. With some insights into the problem at hand, insights that a general approach might not have, we can decompose the ILP problem into two subproblems. First, we use a partially relaxed version of the original, which is more easily solved. We can then use the solution from that problem to set fixed values for a subset of integer variables and resolve the original with that subset of integer variables acting as parameters. This relaxation-based technique is known as relax-and-fix-based decomposition (RFBD) [32]. While the solution is not guaranteed to be optimal in the RFBD technique, the proper selection of the integer variables to relax in the first subproblem and of the integer variables for which we can fix their values in the second subproblem can permit near-optimal solutions. As with most near-optimal algorithms, quality of the solution (in terms of both the objective function value and the runtime improvement) will depend on careful selection of those subsets of variables.
With the particular ILP problem we developed above, we felt that if we could use a partially relaxed version of the problem to first identify which specific span additions to select, then we could fix that topology and solve the original unrelaxed problem with a known topology. We therefore decompose our problem as follows:

Step 1: relax working capacity \( w_j \), spare capacity \( s_j \), and \( p \)-cycle placement variables \( n_p \) and solve the original ILP problem. In other words, all of the integrality requirements on those decision variables are removed and the ILP solved.

Step 2: fix all span establishment variables \( \delta_i \) to the values obtained in Step 1. In other words, take the resulting values for all span establishment variables as solved in Step 1, and convert those variables to parameters with the same values.

Step 3: solve the original ILP, resetting integrality requirements in all relevant variables (but where all \( \delta_i \) variables are fixed to the values in Step 2).

The main rationale for the above decomposition approach is that the span establishment variables are binary, and so fractional values would have very little meaning; if \( \delta_i = 0 \), then the span is not selected, and if \( \delta_i = 1 \), then the span is selected, but \( \delta_i = 0.5 \), for instance, is difficult to interpret in a manner that has any real physical meaning. The three sets of variables noted in Step 1, however, can be permitted to take on fractional values, and the solution can still impart some physical meaning. For instance, \( w_j = 7.8 \) would mean that 7.8 units of working capacity are placed on span \( j \), which might not strictly be feasible (one cannot place a fractional unit of capacity) but is still conceptually understandable. In addition, the span establishment variables will still be driven to \( \delta_i = 0 \) or \( \delta_i = 1 \) whether the \( w_j, s_j \), and \( n_p \) variables are integer or not. Then when we resolve the ILP in Step 3, with the span establishment variables fixed in Step 2, the resultant ILP is equivalent to the basic \( p \)-cycle network design problem.

The branch-and-bound technique is widely used to solve ILP models, as there is no known polynomial time algorithm. This technique applies an intelligent enumeration scheme that can cover all possible solutions by only evaluating a small number of solutions [33]. However, in the worst-case situations, it requires complete enumeration, and in practice, the solver can require a very long time to reach optimality. For our model above, if the solver needed to enumerate all possible nodes in the branch-and-bound tree, the total number of nodes is \( m_1 |Q| + m_2 |Q| + m_3 |Q| + 2|P| \), where \( m_1, m_2, \) and \( m_3 \) are number of integer working capacity variables, spare capacity variables, and \( p \)-cycle variables. More specifically, \( m_1 = |S|, m_2 = |S|, \) and \( m_3 = |P| \). However, in our proposed decomposition method, even in the worst-case scenarios, the total number of possible nodes in the branch-and-bound tree for Step 1 is only \( 2|P| \) due to the relaxations of the integrality property of the first three variables. Furthermore, the resulting model in Step 2 becomes much easier to solve, as very tight LP relaxations arise due to the “known” values of the span establishment variables. As a result the proposed method reduces the overall computational complexity of the model significantly, which we’ll discuss in more detail in a moment.

### 6. Results for Decomposition Method

To test the performance of this above technique, we selected the most computationally complex instance of the problem for each network (though we skip the 10-node and 15-node networks, as their solutions are already trivial). More specifically, we tested the decomposition technique on the 20-node network with 18 eligible spans, the 25-node network with 20 eligible spans, the 30-node network with 26 eligible spans, the 35-node network with 24 eligible spans, and the 40-node network with 37 eligible spans. And as stated earlier,
The explanation is that the decomposition approach provides a less-costly solution more costly than the full ILP, and in three of the five cases, the decomposition technique are at worst only 0.016% gap. In fact, the heuristic solution was slightly better than the objective function value remained within the optimality gap setting of 0.1%, which means that the two approaches effectively provide equivalent solutions.

Table 1: Comparison of normalized total cost between decomposition method and exact method.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Normalized total cost</th>
<th>Percentage difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>20n40-39s</td>
<td>1</td>
<td>1.000137</td>
</tr>
<tr>
<td>25n50-46s</td>
<td>1.76266</td>
<td>1.762942</td>
</tr>
<tr>
<td>30n60-57s</td>
<td>2.834284</td>
<td>2.833077</td>
</tr>
<tr>
<td>35n75-70s</td>
<td>3.634334</td>
<td>3.633292</td>
</tr>
<tr>
<td>40n80-78s</td>
<td>4.468206</td>
<td>4.466802</td>
</tr>
</tbody>
</table>

our ILP models were implemented in AMPL and solved with the CPLEX 11.2 solver. We used a CPLEX mipgap setting of 0.001, which means that all test cases solved to full termination are provably within 0.1% of optimality.

Figure 13 compares the CPU runtimes of the decomposition approach with the original ILP solution. The general trend is that runtime improvements are greater in larger test cases. As we can observe, we see only moderate runtime improvements for the midsize networks (a 21% reduction in the 20-node test case) but significantly greater runtime improvements in the largest networks (a 99.99% reduction in the 40-node test case).

Of course, the tradeoff when implementing a heuristic approach is often a reduction in optimality of the resulting solution, so we also need to compare the solutions we obtain with the decomposition approach with the solutions from Section 4. As we can see from Table 1, the solutions obtained with the decomposition technique at worst only 0.016% more costly than the full ILP, and in three of the five cases the decomposition approach provides a less-costly solution than the full ILP. This is counterintuitive, as the cost of the full ILP solution should serve as the lower bound on the cost of solutions obtained via our heuristic approach. The explanation is that the differences are smaller than the mipgap setting of 0.1%. In fact, we can note that in all cases, the difference between the solutions obtained from the decomposition approach and the full ILP are within the optimality gap setting of 0.1%, which means that the two approaches effectively provide equivalent solutions.

7. Conclusion

We have developed a new ILP model for incremental topology optimization in a p-cycle network that is capable of selecting an optimal subset of eligible spans to add to an existing p-cycle network. While the ILP proves to be relatively easy to solve for small test case network instances, it is computationally complex to solve for larger networks. We then developed a relaxation-based decomposition heuristic that significantly reduces runtime of the ILP in large networks, while having no statistical impact on optimality. In the most computationally complex instance, the ILP runtime of over 184 thousand seconds (more than two days) was reduced to less than 2300 seconds (less than an hour), while the objective function value remained within the optimality gap. In fact, the heuristic solution was slightly better than the full ILP (though again, we note that it was not provably better since the difference was smaller than the optimality gap).

References


