*This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.*

# A New ILP-Based *p*-Cycle Construction Algorithm without Candidate Cycle Enumeration

Bin Wu, Kwan L. Yeung, King-Shan Lui
Dept. of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam, Hong Kong
E-mail: {binwu, kyeung, kslui}@eee.hku.hk

Shizhong Xu
School of Communication and Information Engineering
University of Electronic Science and Technology of China
Chengdu, P. R. China, 610054
E-mail: xsz@uestc.edu.cn

***Abstract*—The notion of *p*-cycle (Preconfigured Protection Cycle) allows capacity efficient schemes to be designed for fast span protection in WDM mesh networks. Conventional *p*-cycle construction algorithms need to enumerate/pre-select candidate cycles before ILP (Integer Linear Program) can be applied. In this paper, we propose a new algorithm which is only based on ILP. When the required number of *p*-cycles is not too large, our ILP can generate optimal/suboptimal solutions in reasonable amount of running time.**

***Keywords-Fast span protection, ILP (Integer Linear Program), p-cycle, WDM (Wavelength Division Multiplexing).***

## I. INTRODUCTION

WDM (Wavelength Division Multiplexing) technology is widely used in optical networks to fully utilize the fiber bandwidth. Nowadays, the capacity of a WDM wavelength can easily reach 10 Gb/s, and hundreds of wavelengths can be multiplexed onto a single fiber for concurrent data transmission. To minimize the amount of data loss upon an accidental failure (such as a fiber cut), it is imperative that the network can survive from the failure and achieve fast optical recovery.

In WDM mesh networks, *p*-cycle (Preconfigured Protection Cycle) [1] can provide fast span (or link) protection with high capacity efficiency. The idea is to organize the spare (or backup) capacity in the network into a set of pre-cross-connected cycles (i.e., *p*-cycles) to protect the working capacity at each span. If a *p*-cycle traverses a particular span, then this span is called an *on-cycle span* of this *p*-cycle. Otherwise, it is called a *straddling span* if both of its two end nodes are on the same *p*-cycle. We define a unity-*p*-cycle as a pre-cross-connection of one unit of spare capacity (or one wavelength) on the on-cycle spans it traverses. Each *p*-cycle refers to a unity-*p*-cycle in this paper. Fig. 1 shows a simple network with bidirectional fiber connections. The two spans 0–1 and 3–4 are straddling spans of the dashed *p*-cycle, and all other spans are on-cycle spans. If the on-cycle span 2–3 fails, the traffic on it can be rerouted to the other side of the *p*-cycle, i.e., path 2–0–4–1–3. If the straddling span 3–4 fails, two backup paths 3–1–4 and 3–2–0–4 are available for protection. We can see that a straddling span is better protected than an on-cycle span, because a *p*-cycle protects two units of traffic for the former but only one unit for the latter. Although no spare capacity is preconfigured at the straddling spans, the reserved capacity on the *p*-cycle is shared to protect both on-cycle spans and straddling spans. This leads to high capacity efficiency

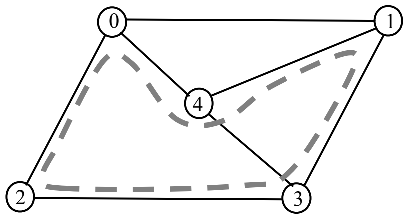


Fig. 1. An example of *p*-cycle protection.

comparable to a span-based mesh restoration scheme [1]. Upon a single span failure, only the two end nodes of the failed span are involved in real time switching. This gives a BLSR [2] ring-like fast recovery speed.

Because of its outstanding performance on both recovery speed and capacity efficiency, *p*-cycle has attracted intensive research interests [3-14] since it was first introduced in 1998 [1]. Recently, *p*-cycle was also extended to path/segment protection at the cost of slower recovery speed [15-16].

For a given network, the problem of *p*-cycle construction is to find a set of *p*-cycles to protect the working capacity on each span (i.e., 100% protection), and minimize the total amount of required spare capacity. Conventional *p*-cycle construction algorithms adopt a two-step approach. The first step enumerates all distinct simple cycles [17] in the network to form a candidate set. The second step determines an optimal set of *p*-cycles from the candidate set by ILP (integer linear program) optimization. However, the size of the candidate set can be very large even in a small network, and it soars exponentially as the network size increases. This makes the algorithm very time-consuming. To address this issue, some heuristic cycle pre-selection algorithms [18-20] are designed to reduce the size of the candidate set, by only selecting a subset of candidate cycles with "high merit" for ILP optimization. Obviously, this also degrades the quality of the solution.

In this paper, we propose a new *p*-cycle construction algorithm which only relies on ILP. We formulate the new ILP in Section II, and give numerical results and discussions in Section III. The paper is concluded in Section IV.

## II. ILP FORMULATION

In our ILP, a "cycle" may actually contain one or multiple disjoint cycles and each of them corresponds to a *p*-cycle. In what follows, we use "cycle" and "*p*-cycle" to distinguish the two different notions. If a span can be protected by a particular *p*-cycle, we also say that it can be protected by the

This work is supported by Hong Kong Research Grant Council Earmarked Grant HKU 7150/04E.

corresponding cycle. With the notations defined in Part A below, we summarize the ILP in Part B, and explain the detailed rationale in Part C.

## *A. Notations*

$J$: The maximum number of cycles allowed in the solution.

$j$: Cycle index where $j \in \{1, 2, \ldots, J\}$. Note that a cycle may contain one or multiple disjoint $p$-cycles.

$e_{ab}^{j}$: Binary variable. It takes the value of 1 if span $(a, b)$ is an on-cycle span of cycle $j$, and 0 otherwise.

$z_{a}^{j}$: Binary variable. It takes the value of 1 if node $a$ is on cycle $j$, and 0 otherwise.

$\chi_{ab}^{j}$: Binary variable. It takes the value of 1 if span $(a, b)$ can be protected by cycle $j$, and 0 otherwise.

$d_{ab}$: The total amount of working capacity on span $(a, b)$ after some routing algorithm is applied.

$x_{ak}\big|_{(a,b)}^{j}$: Binary variable. It assumes that we are checking whether span $(a, b)$ can be protected by cycle $j$. It takes the value of 1 if there is a route on cycle $j$ that connects nodes $a$ and $k$, and 0 otherwise.

$y_{ak}^{t}\big|_{(a,b)}^{j}$: Binary variable. It assumes that we are checking whether span $(a, b)$ can be protected by cycle $j$. For $t \neq b$, it takes the value of 1 if there is a route on cycle $j$ that connects nodes $a$ and $t$, and at the same time $(t, k)$ is an on-cycle span of cycle $j$. Otherwise, it takes the value of 0. For $t=b$, $y_{ak}^{b}\big|_{(a,b)}^{j}=0$ is always enforced (See (8)).

$c_{ab}$: The cost of adding a wavelength to span $(a, b)$. If hop-count is used as the cost metric, then $c_{ab}=1$ for each span $(a, b)$. Otherwise $c_{ab}$ may include distance-related costs such as for using amplifiers, plus the cost of any O/E and E/O interfaces, and the amortized cost of OXCs or wavelength converter pools.

$\boldsymbol{V}$: The set of all the nodes in the network.

$\boldsymbol{E}$: The set of all the spans in the network.

## *B. ILP Formulation*

Given a network topology $\boldsymbol{G(V, E)}$, the working capacity $d_{ab}$ (after routing) and the cost $c_{ab}$ for each span $(a, b) \in \boldsymbol{E}$, the set of $p$-cycles for 100% span protection can be obtained by solving the ILP below.

*1) Objective:*

$$\text{minimize}\left\{\sum_{j}\sum_{(a,b)\in \boldsymbol{E}} c_{ab} e_{ab}^{j}\right\}. \tag{1}$$

*2) Capacity constraint:*

$$\sum_{j}\left(2\chi_{ab}^{j} - e_{ab}^{j}\right) \geq d_{ab}, \quad \forall (a,b) \in \boldsymbol{E}. \tag{2}$$

*3) Cycle constraint:*

$$\sum_{(a,b)\in \boldsymbol{E}} e_{ab}^{j} = 2 z_{a}^{j}, \quad \forall a \in \boldsymbol{V}, \quad \forall j. \tag{3}$$

*4) Protection constraint:*

$$x_{aa}\big|_{(a,b)}^{j} = 1, \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j. \tag{4}$$

$$y_{ak}^{t}\big|_{(a,b)}^{j} \leq \frac{1}{2}\left(x_{at}\big|_{(a,b)}^{j} + e_{tk}^{j}\right), \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j, \quad \forall k \in \boldsymbol{V} : k \neq a, \quad \forall t \in \boldsymbol{V} : (t,k) \in \boldsymbol{E}. \tag{5}$$

$$x_{ak}\big|_{(a,b)}^{j} \leq \sum_{(t,k)\in \boldsymbol{E}} y_{ak}^{t}\big|_{(a,b)}^{j}, \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j, \quad \forall k \in \boldsymbol{V} : k \neq a. \tag{6}$$

$$y_{ak}^{t}\big|_{(a,b)}^{j} + y_{at}^{k}\big|_{(a,b)}^{j} \leq 1, \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j, \quad \forall (t,k) \in \boldsymbol{E} : t \neq a, k \neq a. \tag{7}$$

$$\sum_{(k,b)\in \boldsymbol{E},\ k\neq a} y_{ak}^{b}\big|_{(a,b)}^{j} \leq 0, \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j. \tag{8}$$

$$\chi_{ab}^{j} = x_{ab}\big|_{(a,b)}^{j}, \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j. \tag{9}$$

$$x_{ak}\big|_{(a,b)}^{j} \leq \sum_{(a,t)\in \boldsymbol{E}} e_{at}^{j}, \quad \forall (a,b) \in \boldsymbol{E}, \quad \forall j, \quad \forall k \in \boldsymbol{V} : k \neq a. \tag{10}$$

## *C. Rationale*

In the above ILP, we do not know how many cycles are sufficient for 100% span protection until a solution is obtained. Theoretically, we can set $J$ sufficiently large, whereas the final solution may contain less number of cycles.

Formula (1) states the objective of minimizing the total cost of all cycles/$p$-cycles. Formula (2) specifies that the set of $p$-cycles must provide 100% protection for the given working capacity $d_{ab}$ on each span $(a, b) \in \boldsymbol{E}$. Note that $2\chi_{ab}^{j} - e_{ab}^{j}$ denotes the number of working capacity units at span $(a, b)$ that can be protected by cycle $j$. If span $(a, b)$ cannot be protected by cycle $j$, we have $\chi_{ab}^{j}=0$, $e_{ab}^{j}=0$ and thus $2\chi_{ab}^{j} - e_{ab}^{j}=0$.

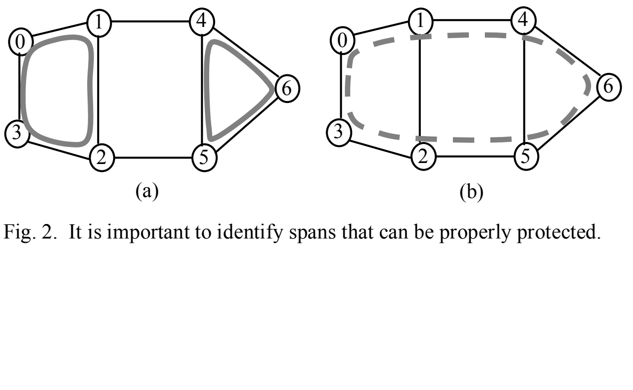


Fig. 2. It is important to identify spans that can be properly protected.

Otherwise, if it can be protected (with $\chi_{ab}^{j}=1$), it may be an on-cycle span (with $e_{ab}^{j}=1$) or a straddling span (with $e_{ab}^{j}=0$). Then, $2\chi_{ab}^{j}-e_{ab}^{j}$ takes the value of 1 for the former and 2 for the latter.

Formula (3) requires a cycle *j* to have either 0 or 2 on-cycle spans incident on any node $a \in \boldsymbol{V}$, depending on whether node *a* is on cycle *j*. With this constraint, multiple disjoint *p*-cycles may coexist in *j*. Fig. 2 gives an example. We assume $d_{ab}=1$ and $c_{ab}=1$ for every span (*a*, *b*), except $c_{14}=c_{25}=1000$. In Fig. 2a, two disjoint *p*-cycles coexist in *j*. If we cannot identify that spans (1, 4) and (2, 5) straddle two disjoint *p*-cycles, they will be mistaken as straddling spans of the same *p*-cycle, and thus can be protected by cycle *j* in Fig. 2a. Accordingly, the total cost is 7, instead of the correct answer 2005 in Fig. 2b (with a single *p*-cycle).

In fact, it is trivial to require that only a single *p*-cycle exists in *j*. Though this can be achieved by adding extra constraints to the ILP, it will dramatically increase its running time. Instead, it is necessary to distinguish straddling spans (of a *p*-cycle) from spans connecting two separate *p*-cycles in *j* (such as (1, 4) and (2, 5) in Fig. 2a). Straddling spans can be properly protected by cycle *j* but spans connecting two separate *p*-cycles cannot. In our ILP, this is achieved by protection constraint 4) (i.e., (4)-(10)) using a recursive process.

To check if span (*a*, *b*) can be protected by cycle *j*, we can check whether there is a route on cycle *j* that connects nodes *a* and *b*. If yes, then both *a* and *b* are on the same *p*-cycle and span (*a*, *b*) can be protected. For simplicity, we use "*a* connects to *b*" to mean that the two nodes are connected by a path on cycle *j*. For example, in Fig. 2a, node 0 connects to node 3, but node 1 does not connect to node 4. In essence, checking the connectivity of two nodes is similar to a routing process along the cycle.

Motivated by some classic routing algorithms such as Dijkstra's and Floyd-Warshall algorithms [21], we can use the recursive process formulated in (4)-(10) to achieve the above goal. Fig. 3 shows the key idea. Checking the connectivity of nodes *a* and *k* in Fig. 3 is equivalent to checking the connectivity of nodes *a* and *t* because (*t*, *k*) is an on-cycle span. Assume that (*s*, *t*) is also an on-cycle span. This further translates to checking the connectivity of *a* and *s*, and so on. If *a* and *k* are on the same *p*-cycle, this recursive process will converge to node *a* which is called the *root* of the recursive process.

For each span $(a, b) \in \boldsymbol{E}$, formulas (4)-(10) check the connectivity of nodes *a* and *b* on cycle *j*. Formula (4) means that the root node *a* always connects to itself. Formulas (5) & (6) are the main body of the recursive process. Specifically, formula (5) defines $y_{ak}^{t}\big|_{(a,b)}^{j}$, which can take the value of 1 only if *a* connects to *t* (with $x_{at}\big|_{(a,b)}^{j}=1$) and (*t*, *k*) is an on-cycle span of cycle *j* (with $e_{tk}^{j}=1$). We call this "*a* connects to *k* via *t*" for short, as shown in Fig. 3. Formula (6) says that, *a* can connect to *k* only if there exists at least one neighbor *t* of *k*, such that *a* can connect to *k* via *t*.

We now use the example in Fig. 4a to show how the recursive process works. Our task is to check whether span (1, 4) can be protected by cycle *j*=1 which contains two disjoint *p*-cycles. Or equivalently, whether $\chi_{14}^{1} = x_{14}\big|_{14}^{1}$ (as defined in (9)) can take the value of 1. Figs. 4c-4h give the details of the recursive process based on (5) & (6). Since $y_{14}^{1}\big|_{(1,4)}^{1}=0$ (because (1, 4) is not an on-cycle span and $e_{14}^{1}=0$), the value of $x_{14}\big|_{(1,4)}^{1}$ in Fig. 4c depends on the values of $y_{14}^{6}\big|_{(1,4)}^{1}$ and $y_{14}^{5}\big|_{(1,4)}^{1}$. We first

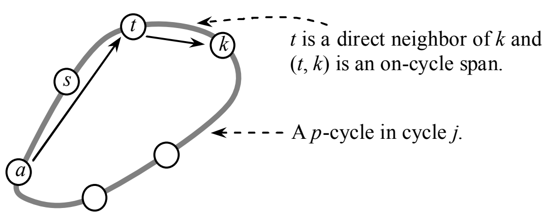
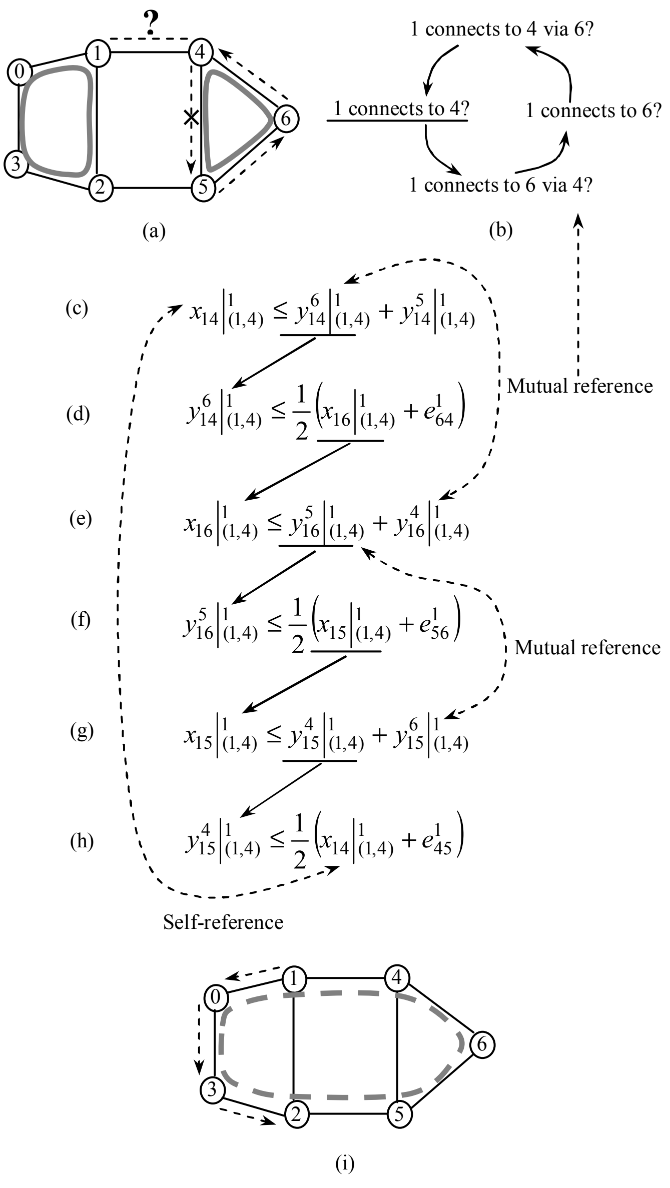


Fig. 3. Node *a* connects to node *k* via its neighbor *t* ($y_{ak}^{t}\big|_{(a,b)}^{j}=1$).



?
0
1
2
3
4
5
6
(a)
1 connects to 4 via 6?
1 connects to 4?
1 connects to 6?
1 connects to 6 via 4?
(b)
(c) $x_{14}\big|_{(1,4)}^{1} \le y_{14}^{6}\big|_{(1,4)}^{1} + y_{14}^{5}\big|_{(1,4)}^{1}$
(d) $y_{14}^{6}\big|_{(1,4)}^{1} \le \frac{1}{2}\left(x_{16}\big|_{(1,4)}^{1} + e_{64}^{1}\right)$
(e) $x_{16}\big|_{(1,4)}^{1} \le y_{16}^{5}\big|_{(1,4)}^{1} + y_{16}^{4}\big|_{(1,4)}^{1}$
(f) $y_{16}^{5}\big|_{(1,4)}^{1} \le \frac{1}{2}\left(x_{15}\big|_{(1,4)}^{1} + e_{56}^{1}\right)$
(g) $x_{15}\big|_{(1,4)}^{1} \le y_{15}^{4}\big|_{(1,4)}^{1} + y_{15}^{6}\big|_{(1,4)}^{1}$
(h) $y_{15}^{4}\big|_{(1,4)}^{1} \le \frac{1}{2}\left(x_{14}\big|_{(1,4)}^{1} + e_{45}^{1}\right)$
Mutual reference
Mutual reference
Self-reference
(i)

Fig. 4. Mutual reference, self-reference and the recursive process.

consider $y_{14}^{6}\big|_{(1,4)}^{1}$. From Figs. 4d & 4e, we can see that the value of $y_{14}^{6}\big|_{(1,4)}^{1}$ further depends on the values of $y_{16}^{5}\big|_{(1,4)}^{1}$ and $y_{16}^{4}\big|_{(1,4)}^{1}$, because (6, 4) is an on-cycle span in Fig. 4a and thus we have $e_{64}^{1}=1$ in Fig. 4d. However, at this point we should not refer back to node 4 again (or $y_{16}^{4}\big|_{(1,4)}^{1}$ in Fig. 4e). Otherwise we will be trapped by the logical loop shown in Fig. 4b, which is called *mutual reference* between nodes 4 and 6. In such a mutual reference, the connectivity of node pairs {1, 4} and {1, 6} depends on each other, and thus both will be mistaken as connected. Note that the ILP will treat any two nodes as connected if their connectivity is undefined, because this can help to minimize the objective in (1). Therefore, we should prevent such mutual references, and let the nodes on the cycle be sequentially referred to in a unidirectional manner, as shown by the dashed arrows in Fig. 4a. This is achieved by (7) in the ILP, which allows either $y_{ak}^{t}\big|_{(a,b)}^{j}$ or $y_{at}^{k}\big|_{(a,b)}^{j}$ (but not both) to take the value of 1.

Assume $y_{16}^{4}\big|_{(1,4)}^{1}=0$ in Fig. 4e for avoiding the mutual reference between nodes 4 and 6. Then, the recursive process continues as in Figs. 4e-4h. Note that $e_{56}^{1}=1$ in Fig. 4f and $e_{45}^{1}=1$ in Fig. 4h (See the cycle in Fig. 4a), and we have omitted $y_{15}^{2}\big|_{(1,4)}^{1}$ in Fig. 4g because it equals to 0. Besides, we have $y_{15}^{6}\big|_{(1,4)}^{1}=0$ in Fig. 4g to avoid mutual reference between nodes 5 and 6. Finally, we can see that the value of $x_{14}\big|_{(1,4)}^{1}$ in Fig. 4c ultimately depends on its own value (i.e., $x_{14}\big|_{(1,4)}^{1}$ in Fig. 4h). We call such a logical loop as *self-reference* of span (1, 4), and it should also be avoided. The physical meaning behind is as follows. To check the connectivity of nodes 1 and 4 in Fig. 4a, we start from node 4 and retrieve the connectivity in a recursive manner along the cycle. We finally reach node 4 instead of node 1. Generally, it means that node 1 does not connect to node 4, and span (1, 4) cannot be protected by the cycle. Formula (8) in our ILP is to prevent the self-reference problem. When we check whether span (*a*, *b*) can be protected by cycle *j*, formula (8) sets $y_{ak}^{b}\big|_{(a,b)}^{j}=0$ if *k* (where *k≠a*) is a direct neighbor of node *b*. This is equivalent to removing the dashed arrow 4→5 in Fig. 4a, or set $y_{15}^{4}\big|_{(1,4)}^{1}=0$ in Fig. 4g, such that the self-reference of span (1, 4) can be avoided. Note that this is carried out only for checking the connectivity of nodes *a* and *b*. It does not affect our analysis on the connectivity of other node pairs.

In Fig. 4g, we have $y_{15}^{4}\big|_{(1,4)}^{1}=0$ to avoid self-reference of span (1, 4), and $y_{15}^{6}\big|_{(1,4)}^{1}=0$ to avoid mutual reference between nodes 5 and 6. Therefore, we get $x_{15}\big|_{(1,4)}^{1}=0$, meaning that node 1 does not connect to node 5. Recursively, we have $y_{14}^{6}\big|_{(1,4)}^{1}=0$ in Fig. 4c. If $y_{14}^{5}\big|_{(1,4)}^{1}$ in Fig. 4c is considered, we can get $y_{14}^{5}\big|_{(1,4)}^{1}=0$ in a similar way. Consequently, $x_{14}\big|_{14}^{1}=0$ in Fig. 4c, and span (1, 4) cannot be protected by cycle *j*=1.

On the other hand, if we check whether span (1, 2) can be protected by cycle *j*=1 in Fig. 4i, the connectivity of nodes 1 and 2 can be retrieved as indicated by the dashed arrows. We can finally reach node 1, and $\chi_{12}^{1}=x_{12}\big|_{(1,2)}^{1}=1$ is ensured by the root definition $x_{11}\big|_{(1,2)}^{1}=1$ (see formula (4)). Therefore, span (1, 2) can be protected by cycle *j*=1 which is also a *p*-cycle.

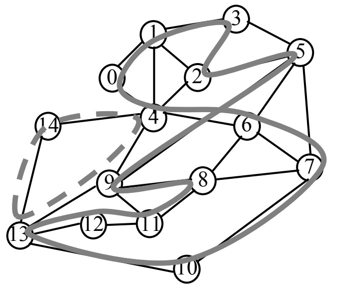
In summary, formulas (4)-(10) are used to check whether span (*a*, *b*) can be protected by cycle *j*. Specifically, formula (4) defines the root, and formulas (5) & (6) form the main body of the recursive process. Formulas (7) & (8) are used to prevent mutual reference and self-reference, respectively. Formula (9) defines $\chi_{ab}^{j}=x_{ab}\big|_{(a,b)}^{j}$. Finally, formula (10) says that, if there is no on-cycle spans of cycle *j* incident on node *a*, then *a* does not connect to any other node along cycle *j*. For the example in Fig. 4, a final solution with a single *p*-cycle can be obtained in Fig. 4i, instead of the two *p*-cycles in Fig. 4a.

## III. Numerical Results and Discussion

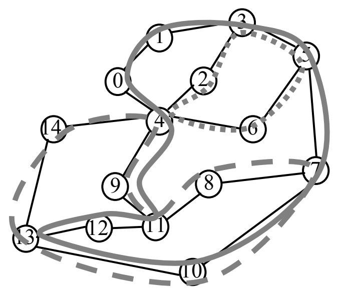
The ILP formulated in (1)-(10) is implemented using ILOG CPLEX 10.0 [22]. To speed up the ILP optimization, we emphasize on finding feasible solutions instead of optimal solutions. Specifically, we set the environment parameters of CPLEX 10.0 as in (11) below to tighten our ILP model, and allow more frequent heuristic processing.

$$\left\{\begin{array}{l}1\rightarrow \text{emphasis mip}\\ 2\rightarrow \text{mip strategy probe}\\ 3\rightarrow \text{mip strategy rins}\\ 3\rightarrow \text{mip strategy heuristicfreq}\\ 2\rightarrow \text{mip cuts all}\\ 3\rightarrow \text{mip strategy dive}\\ 3\rightarrow \text{preprocessing symmetry}\end{array}\right. \tag{11}$$

Let *N*=||***V***|| be the network size and *S*=||***E***|| be the total



(a) *N*=15, *S*=27.



(b) *N*=15, *S*=19.

Fig. 5. *p*-cycles for the two homogeneous networks.

number of spans in the network. The two *homogeneous networks* (or flat capacity networks) in Fig. 5 (taken from [23]) are first considered, where the anticipated working capacity on each span is the same. The homogeneous scenario corresponds to $p$-cycle protection for dynamic traffic [10], or for fiber-level protection in DWDM networks [23]. In Fig. 5, we assume that the working capacity on each span is one unit, and hop-count is used as the cost metric (i.e., $c_{ab}$=1 for each span ($a$, $b$)). The optimal solutions returned by the ILP are also shown in Fig. 5, where both solutions are obtained quickly for $J$=3. For homogeneous networks, the required number of $p$-cycles is usually small. Therefore, our ILP can return an optimal solution rather quick.

We then consider the pan European COST 239 network in Fig. 6. The cost of each span is defined as the Euclid distance between its two end nodes (See Fig. 6a). The number next to each span in Fig. 6b denotes the number of working capacity units (i.e., $d_{ab}$) on that span. The optimal solution obtained is shown in Fig. 6b, which consists of 2 $p$-cycles.

Examples in Figs. 7 & 8 are for capacitated networks. Fig. 7a is the traffic matrix for the capacitated pan European COST 239 network in Fig. 7b. It is obtained by dividing the traffic matrix in [24] by 10 Gb/s. The cost of each span is the same as in Fig. 6a. Fig. 7b shows the working capacity at each span after shortest path routing. The solutions at different running time are given in Fig. 7c with $J$=7. Particularly, the set of $p$-cycles obtained after running for 3 hours are listed. The solution has a gap of 2.10% to optimality. The network in Fig. 8a is taken from [23], where hop-count is used as the cost metric. Our ILP returns the optimal solution [23] (listed in Fig. 8b) in just 886.53 seconds (though a gap to optimality of 3.14% is still observed after 3 hours).

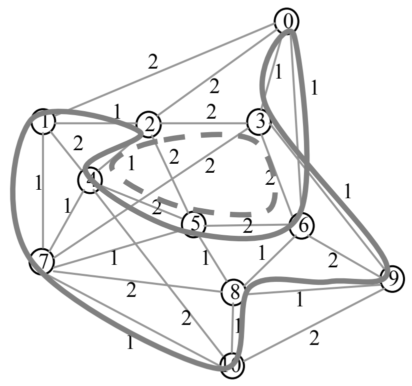
One issue in our ILP is how to determine the number of allowed cycles $J$. Theoretically, we can set $J$ large enough and the ILP can return a solution with less number of cycles. But, the running time of our ILP is very sensitive to $J$, because the number of variables and constraints will increase rapidly with $J$. Note that each $p$-cycle can protect two units of working capacity on a straddling span, and the ILP tends to generate $p$-cycles that straddle the most-heavily-loaded span (in order to reduce the number of required $p$-cycles). So intuitively, we can set $J$ to half of the working capacity units at the most-heavily-loaded span, or slightly larger than that. On the other hand, if $J$ is too small to give a feasible solution, CPLEX can identify the infeasibility very fast (usually in less than 1 second). If this happens, we can slightly increase $J$ before rerunning the ILP.

From the examples in Figs. 5-8, we can see that our ILP returns optimal/suboptimal solutions in reasonable amount of running time. In contrast, the candidate cycle enumeration in

| Span | Cost | Span | Cost |
|---|---|---|---|
| 0–1 | 1310 | 4–5 | 220 |
| 0–2 | 760 | 4–7 | 300 |
| 0–3 | 390 | 4–10 | 930 |
| 0–6 | 740 | 5–6 | 730 |
| 1–2 | 550 | 5–7 | 400 |
| 1–4 | 390 | 5–8 | 350 |
| 1–7 | 450 | 6–8 | 565 |
| 2–3 | 660 | 6–9 | 320 |
| 2–4 | 210 | 7–8 | 600 |
| 2–5 | 390 | 7–10 | 820 |
| 3–6 | 340 | 8–9 | 730 |
| 3–7 | 1090 | 8–10 | 320 |
| 3–9 | 660 | 9–10 | 820 |

0: Copenhagen
1: London
2: Amsterdam
3: Berlin
4: Brussels
5: Luxembourg
6: Prague
7: Paris
8: Zürich
9: Vienna
10: Milan

(a) Span cost in kilometers.



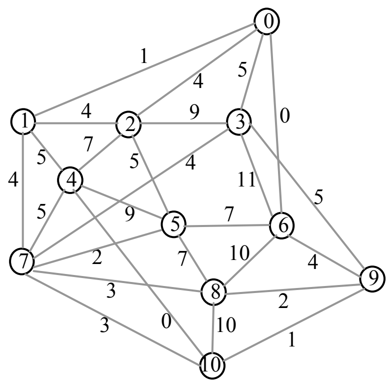
$J$ = 3
Number of cycles = 2
Total cost = 7980 km
Solution time = 616.91 sec

(b) Topology ($N$=11, $S$=26) and $p$-cycles.

Fig. 6. A simple example based on pan European COST 239 network.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ & 0 & 2 & 2 & 1 & 1 & 1 & 3 & 1 & 1 & 1 \\ & & 0 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ & & & 0 & 2 & 1 & 1 & 4 & 3 & 3 & 3 \\ & & & & 0 & 1 & 1 & 2 & 1 & 1 & 1 \\ & & & & & 0 & 1 & 1 & 1 & 1 & 1 \\ & & & & & & 0 & 1 & 1 & 1 & 1 \\ & & & & & & & 0 & 2 & 1 & 2 \\ & & & & & & & & 0 & 1 & 2 \\ & & & & & & & & & 0 & 1 \\ & & & & & & & & & & 0 \end{bmatrix}$$

(a) Traffic matrix.

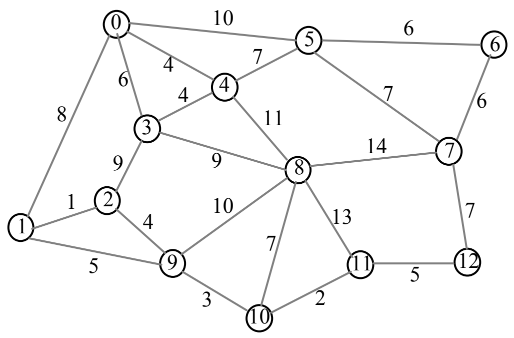


(b) Working capacity on each span.

| Time in hours | Total cost | Gap to optimality | $p$-cycles |
|---|---|---|---|
| 1 | 32960 | 7.72% | |
| 2 | 32130 | 3.23% | |
| 3 | 31790 | 2.10% | 0–3–2–4–5–8–10–9–6–0<br>1–2–3–9–6–8–5–4–10–7–1<br>0–2–4–10–8–5–6–9–3–0<br>0–3–2–4–5–8–10–9–6–0<br>4–5–8–10–4<br>0–1–2–4–5–8–7–10–9–6–3–0<br>0–3–9–10–7–1–4–2–5–8–6–0 |

(c) ILP solutions with $J$=7.

Fig. 7. Case study for the capacitated pan European COST 239 network.



(a) Working capacity on each span.

| Time in hours | Total cost | Gap to optimality | *p*-cycles |
|---|---|---|---|
| 1 | 85 | 3.53% | |
| 2 | 85 | 3.53% | |
| 3 | 85 | 3.14% | 0–4–8–3–2–1–9–10–11–12–7–6–5–0 |
| | | | 0–1–2–9–8–11–12–7–6–5–4–3–0 |
| | | | 0–3–2–1–9–8–10–11–12–7–6–5–4–0 |
| | | | 4–5–7–12–11–10–9–8–4 |
| | | | 0–1–9–2–3–8–10–11–12–7–6–5–4–0 |
| | | | 0–1–2–9–10–11–12–7–6–5–4–8–3–0 |
| | | | 0–1–2–9–8–10–11–12–7–6–5–4–3–0 |

(b) ILP solutions with $J$=7.

Fig. 8. A capacitated network taken from [23].

conventional *p*-cycle construction algorithms is much more time-consuming. However, the complexity of our ILP is very sensitive to the value of *J*. For heavily loaded networks, two approaches can be taken to address this issue. First, we can reduce the number of *p*-cycles by scaling down the traffic with a larger bandwidth unit (e.g. combine two or more wavelengths into a single bandwidth unit). Second, we can follow the divide-and-conquer approach to break the set of demands into subsets, and solve each subset separately using our ILP.

## IV. Conclusion

We proposed a new *p*-cycle construction algorithm which only relies on ILP (Integer Linear Program). Compared to the conventional algorithms, it removes the time-consuming process for candidate cycle enumeration and pre-selection. When the required number of *p*-cycles is not too large, our ILP can generate optimal/suboptimal solutions in reasonable amount of running time.

## References

[1] W. D. Grover and D. Stamatelakis, "Cycle-oriented distributed preconfiguration: ring-like speed with mesh-like capacity for self-planning network restoration," *IEEE ICC '98*, vol. 1, pp. 537-543, Jun. 1998.

[2] Huan Liu and F. A. Tobagi, "Traffic grooming in WDM/SONET BLSR rings with multiple line speeds," *IEEE GLOBECOM '05*, vol. 4, pp. 2096-2101, Dec. 2005.

[3] D. Stamatelakis and W. D. Grover, "Theoretical underpinnings for the efficiency of restorable networks using preconfigured cycles ("*p*-cycles")," *IEEE Trans. on Comm.*, vol. 48, no. 8, pp. 1262-1265, Aug. 2000.

[4] D. Stamatelakis and W. D. Grover, "IP layer restoration and network planning based on virtual protection cycles," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 10, pp. 1938-1949, Oct. 2000.

[5] W. D. Grover, "Mesh-Based Survivable Networks," Upper Saddle River, NJ: Prentice-Hall, Aug., 2003.

[6] A. Kodian, W. D. Grover, J. Slevinsky and D. Moore, "Ring-mining to *p*-cycles as a target architecture: riding demand growth into network efficiency," in *Proc. 19th Annu. Nat. Fiber Optics Engineers Conf. (NFOEC)*, pp. 1543-1552, Sep., 2003.

[7] G. Shen and W. D. Grover, "Design of protected working capacity envelopes based on *p*-cycles: an alternative framework for survivable automated lightpath provisioning," in *Performance Evaluation and Planning Methods for the Next Generation Internet*, A. Girard, B. Sansò, and F. Vazquez-Abad, Eds. Norwell, MA: Kluwer, 2005.

[8] G. Shen and W. D. Grover, "Performance of protected working capacity envelopes based on *p*-cycles: fast, simple, and scalable dynamic service provisioning of survivable services," in *Proc. Asia-Pacific Optical and Wireless Communication Conf. (APOC)*, vol. 5626, pp. 519-533, Nov. 2004.

[9] D. A. Schupke, C. G. Gruber and A. Autenrieth, "Optimal configuration of *p*-cycles in WDM network," *IEEE ICC '02*, vol. 5, pp 2761-2765, May 2002.

[10] Wensheng He, Jing Fang and A. K. Somani, "A *p*-cycle based survivable design for dynamic traffic in WDM networks," *IEEE GLOBECOM '05*, vol. 4, pp. 1869-1873, Dec. 2005.

[11] Hong Huang and J. A. Copeland, "A series of Hamiltonian cycle-based solutions to provide simple and scalable mesh optical network resilience," *IEEE Communications Magazine*, vol. 40, no. 11, pp. 46-51, Nov. 2002.

[12] D. A. Schupke, "Multiple failure survivability in WDM networks with *p*-cycles," *Circuits and Systems, ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 3, pp. 866-869, May 2003.

[13] D. A. Schupke, "The tradeoff between the number of deployed *p*-cycles and the survivability to dual fiber duct failures," *IEEE ICC '03*, vol. 2, pp. 1428-1432, May 2003.

[14] A. Kodian, A. Sack and W. D. Grover, "*p*-cycle network design with hop limits and circumference limits," *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pp. 244-253, 2004.

[15] Gangxiang Shen and W. D. Grover, "Extending the *p*-cycle concept to path segment protection for span and node failure recovery," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 8, pp. 1306-1319, Oct. 2003.

[16] A. Kodian and W. D. Grover, "Failure-independent path-protecting *p*-cycles: efficient and simple fully preconnected optical-path protection," *Journal of Lightwave Technology*, vol. 23, no. 10, pp. 3241-3259, Oct. 2005.

[17] B. J. Donald, "Finding all the elementary circuits of a directed graph," *SIAM J. Comput.*, vol. 4, no. 1, pp. 77-84, Mar. 1975.

[18] W. D. Grover and J. Doucette, "Advances in optical network design with *p*-cycles: joint optimization and pre-selection of candidate *p*-cycles," in *Proc. IEEE LEOS Summer Topicals*, pp. 49-50, Jul. 2002.

[19] J. Doucette, D. He, W. D. Grover and O. Yang, "Algorithmic approaches for efficient enumeration of candidate *p*-cycles and capacitated *p*-cycle network design," *Design of Reliable Communication Networks, 2003. (DRCN 2003). Proceedings. Fourth International Workshop on*, pp. 212-220, Oct. 2003.

[20] Hanxi Zhang and O. Yang, "Finding protection cycles in DWDM networks," *IEEE ICC '02*, vol. 5, pp. 2756-2760, May 2002.

[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice-Hall, 1993.

[22] www.ilog.com.

[23] A. Sack and W. D. Grover, "Hamiltonian *p*-cycles for fiber-level protection in homogeneous and semi-homogeneous optical networks," *IEEE Network*, vol. 18, no. 2, pp. 49-56, Apr. 2004.

[24] P. Batchelor et al., "Ultra high capacity optical transmission networks," Final report of action COST 239, 1999.