ILP Formulations for Non-Simple *p*-Cycle and *p*-Trail Design in WDM Mesh Networks

Bin Wu, Kwan L. Yeung and Pin-Han Ho

***Abstract*—Conventional simple *p*-cycle (Preconfigured Protection Cycle) concept allows fast and capacity-efficient span protection in WDM mesh networks. Unlike simple *p*-cycle, non-simple *p*-cycle can traverse a node or span multiple times. The recently proposed *p*-trail (Pre-Cross-Connected Trail) concept further removes the cycle constraint by allowing arbitrary protection trails, leading to the most flexible and general design. Although non-simple *p*-cycles and *p*-trails are expected to be more capacity-efficient than simple *p*-cycles, it is still unclear how much capacity gain they can achieve compared with simple *p*-cycles. In this paper, we first point out some unique features of non-simple *p*-cycles and *p*-trails which are not fully explored in previous studies. This motivates us to formulate a new suite of ILPs (Integer Linear Programs) for optimal design of non-simple *p*-cycles and *p*-trails, based on which the achievable capacity gain over simple *p*-cycles can be investigated. Different from all the previous studies, the proposed ILPs are free of candidate cycle/trail enumeration, and can truly achieve an optimal design of non-simple *p*-cycles and *p*-trails. Our numerical results show that the capacity gain can be significant in some small-size networks with lightly-loaded traffic, but is generally trivial as the network size and the traffic load increase.**

***Index Terms*—Fast span protection, *p*-cycle (Preconfigured Protection Cycle), *p*-trail (Pre-Cross-Connected Trail), WDM (Wavelength Division Multiplexing).**

I. Introduction

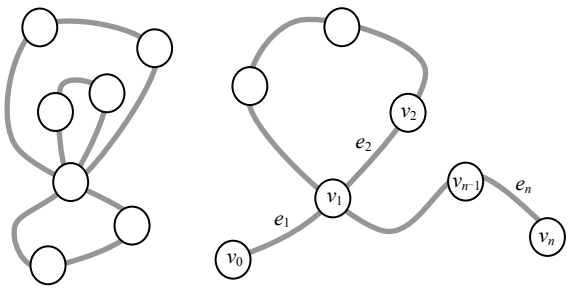
WDM (Wavelength Division Multiplexing) technology allows hundreds of wavelengths to be multiplexed onto a single fiber for parallel transmission. To minimize service downtime and data loss upon a span/link failure (such as a fiber-cut), the most crucial issue is to achieve immediate optical recovery using a fast protection scheme. As pointed out in [1], the key for achieving fast protection is the pre-cross-connection of the backup path/wavelengths. Protection schemes can be further classified into span-based and path/segment-based schemes. Span-based protection tends to be faster in optical recovery, because failure detection and traffic switching are carried out locally by the two end nodes of the failed span and thus the signaling overhead is minimized. In this paper, we focus on span-based protection with pre-cross-connected backup wavelengths.

Bin Wu and Pin-Han Ho are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada, N2L 3G1 (Tel: 1-519-888-4567 ex. 32452; Fax: 1-519-746-3077; e-mail: b7wu@uwaterloo.ca, pinhan@bbcr.uwaterloo.ca).

Kwan L. Yeung is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong (Tel: 852-2857-8493; Fax: 852-2559-8738; e-mail: kyeung@eee. hku.hk).

The classical *p*-cycle (Preconfigured Protection Cycle) concept [2] allows fast span protection with high capacity efficiency. Generally, a *p*-cycle refers to a *simple p-cycle* which traverses a node or span at most once. It is a ring-like loop-back pre-cross-connection using one backup wavelength (i.e., one unit of spare capacity) on each span it traverses. If a span is traversed by a *p*-cycle, it is called an *on-cycle span* of this *p*-cycle; otherwise it is a *straddling span* if its both end nodes are traversed by the *p*-cycle. Assume a single span failure at a time. The pre-cross-connected spare capacity reserved along a *p*-cycle can be shared to protect all its on-cycle and straddling spans. This leads to very high capacity efficiency. Meanwhile, only the two end nodes of the failed span need to carry out real time switching, and this leads to fast optical recovery. For a given network, an optimal set of simple *p*-cycles for 100% span protection can be found in two steps [2-3]. The first step enumerates all the distinct simple cycles in the network to form a candidate set. The second step adopts an ILP (Integer Linear Program) to find an optimal set of *p*-cycles from the candidate set, so as to minimize the required spare or total capacity. The *p*-cycle concept is also extended to *non-simple p-cycles* [3-4], where a non-simple *p*-cycle can traverse some nodes or spans multiple times (see Fig. 1a). So far, the same two-step approach based on non-simple cycle enumeration is the only way for non-simple *p*-cycle design. However, enumerating all non-simple cycles in a network is generally a formidable task.

Besides simple and non-simple *p*-cycles, the *p*-trail concept (Pre-Cross-Connected Trail or PXT [1]) is also introduced for fast protection. Let v_{i-1} and v_i be the two end nodes of a span e_i. A *trail* can be denoted by an alternating sequence (v_0, e_1, v_1, e_2, v_2, …, v_{n-1}, e_n, v_n) of nodes v_i and spans e_i, as shown in Fig. 1b. If a span is traversed by a trail, it is called an *on-trail span*; otherwise it is a *straddling span* if its two end nodes are



(a) Non-simple *p*-cycle (b) *p*-trail

Fig. 1. Non-simple *p*-cycle and *p*-trail (Pre-Cross-Connected Trail).

traversed by the trail. As defined in [1], a trail can traverse a node multiple times but a span at most once. To facilitate our analysis on *p*-trails, in this paper we extend the concept by allowing a *p*-trail to traverse a span once per direction. If $v_0 \neq v_n$, the trail is called an *open trail*; otherwise it is a closed trail, which is indeed a simple or non-simple cycle. A *p*-trail is implemented by pre-cross-connecting the backup wavelengths along the trail. Similar to *p*-cycles, *p*-trails can achieve fast optical recovery due to the pre-cross-connection nature. Since simple *p*-cycles can be regarded as a special case of non-simple *p*-cycles and both simple and non-simple *p*-cycles are special cases of *p*-trails, theoretically a non-simple *p*-cycle solution will never has poorer capacity efficiency than its simple *p*-cycle counterpart, and a *p*-trail solution will give the best capacity efficiency.

Although it is obvious that non-simple *p*-cycles and *p*-trails can achieve better capacity efficiency than simple *p*-cycles, it is still unclear how much capacity gain they can achieve. This is due to two reasons. First, we notice that previous studies have ignored some distinct features of non-simple *p*-cycles and *p*-trails. Unlike a simple *p*-cycle (which can protect one unit of working capacity on each on-cycle span and two units on each straddling span), a non-simple *p*-cycle or *p*-trail may protect more than two units of working capacity on a particular on-cycle/on-trail or straddling span. Besides, the same set of on-cycle/on-trail spans can be pre-cross-connected in different patterns to generate different non-simple *p*-cycles/*p*-trails with different protection capabilities. Without observing those distinct features, a non-simple *p*-cycle or *p*-trail solution cannot be properly obtained. Second, designing an optimal non-simple *p*-cycle or *p*-trail solution is very complex. With the conventional two-step approach based on candidate cycle enumeration, even simple *p*-cycle design is very complex, because the total number of candidate cycles in a network increases exponentially with the network size. If non-simple cycles are considered, the size of the candidate set will be dramatically boosted. This is because multiple simple cycles can be geometrically combined in a combinatorial manner to form different non-simple cycles. Besides, multiple non-simple *p*-cycles can be generated from the same set of on-cycle spans depending on the pre-cross-connection patterns. Obviously, if a similar trail enumeration approach is adopted for *p*-trail design, the total number of candidate trails in a network will be far more than that of non-simple cycles. A large candidate set leads to a huge number of ILP variables which significantly slows down the optimization process. Although the size of the candidate set can be reduced by using only a subset of all the candidates in the design [3-7], the solution quality may be impaired accordingly [8]. Recently, some ILPs without candidate cycle enumeration were formulated for simple *p*-cycle design [8-10], but they cannot be directly extended to non-simple *p*-cycle or *p*-trail design.

It is clear that both non-simple *p*-cycles and *p*-trails have not been sufficiently studied so far, and finding the corresponding optimal solutions is still a difficult task. Motivated by the above observations, our work contributes to the related studies in the following three aspects: 1) the paper identifies those distinct features of non-simple *p*-cycles and *p*-trails, such that their potential benefits can be fully explored; 2) by taking those distinct features into account, a set of optimal ILP models is formulated for non-simple *p*-cycle and *p*-trail design. We claim that our ILPs are the first efforts on optimally solving the problem without candidate cycle/trail enumeration; and 3) based on the ILPs formulated, the paper investigates the capacity gain of non-simple *p*-cycles and *p*-trails over the conventional simple *p*-cycles. To the best of our knowledge, the paper is the first study that initiates an optimal performance comparison among simple *p*-cycles, non-simple *p*-cycles and *p*-trails. We conclude that the capacity gain can be significant in small-size and lightly-loaded networks but is generally trivial as network size and traffic load increase. This makes it clear that the capacity gain due to non-simple *p*-cycles and *p*-trails is generally trivial in most practical designs.

The rest of the paper is organized as follows. Section II highlights the distinct features of non-simple *p*-cycles and *p*-trails. Section III formulates the ILPs without candidate cycle/trail enumeration. Numerical results are presented in Section IV, and we conclude the paper in Section V.

II. Distinct Features of Non-Simple *p*-Cycles and *p*-Trails

In an undirected network, a simple *p*-cycle can protect one unit of traffic on each on-cycle span and two units on each straddling span [2]. Fig. 2a shows an example of simple *p*-cycle protection. For the *p*-cycle 0−4−1−3−2−0, if on-cycle span (2, 3) fails, it provides one backup path 2−0−4−1−3; if straddling span (0, 1) fails, it provides two backup paths 0−4−1 and 0−2−3−1

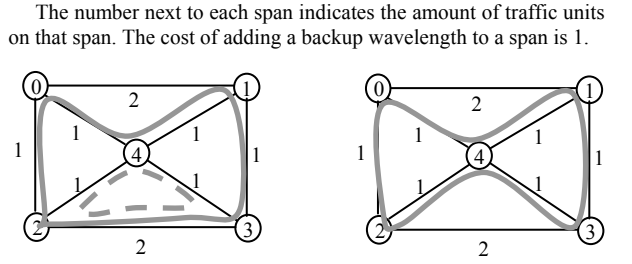


Fig. 2. Simple *p*-cycles do not fully explore mesh connectivity.

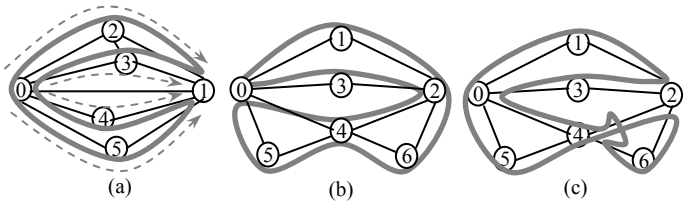


Fig. 3. A non-simple *p*-cycle may protect more than 2 units of traffic on a span, and the same set of on-cycle spans may accommodate different non-simple *p*-cycles with different protection capabilities, depending on how the backup wavelengths are pre-cross-connected.

and thus two units of traffic on (0, 1) can be protected.

In Fig. 2, the number next to each span indicates the amount of traffic units carried on that span, and the cost of adding a backup wavelength to each span is 1. In Fig. 2a, two simple *p*-cycles with a total cost of 8 are necessary for 100% span protection. If non-simple *p*-cycle is used, Fig. 2b shows that a single non-simple *p*-cycle is sufficient with a total cost of 6. This gives a 25% spare capacity saving. Unlike existing observations which attribute such a performance gain to the increase of the candidate set size, we would like to point out that this is because non-simple *p*-cycles can better explore mesh connectivity of a network than simple *p*-cycles, due to the more flexible protection structure.

Unlike a simple *p*-cycle, a non-simple *p*-cycle may protect more than 2 units of traffic on a particular on-cycle or straddling span. In Fig. 3a, although spans (0, 1) and (2, 3) are both straddling spans, the non-simple *p*-cycle can protect 4 units of traffic on (0, 1) as indicated by the four dashed arrows, in contrast to 2 units on (2, 3). Notably, a non-simple *p*-cycle is defined not only by its on-cycle spans, but also by its pre-cross-connection pattern. Consider the two on-cycle spans (0, 4) and (2, 4) in Figs. 3b-3c. The non-simple *p*-cycle pre-cross-connected in Fig. 3b can protect 3 units of traffic on (0, 4) and 1 unit on (2, 4). With the same set of on-cycle spans, the non-simple *p*-cycle pre-cross-connected in Fig. 3c can protect 1 unit of traffic on (0, 4) but 3 units on (2, 4). This example shows that, a non-simple *p*-cycle cannot be properly defined unless we can figure out how the backup wavelengths are pre-cross-connected, and different pre-cross-connection patterns lead to different non-simple *p*-cycles with different protection capabilities.

Compared with non-simple *p*-cycles, *p*-trails are even more flexible in exploring mesh connectivity by removing the cycle constraint. As indicated in [11], "an acyclic structure within a graph contains no closed path. Both trees and trails are acyclic. Trails, as in [1], may exist with loops in their route, but remain as acyclic structures, because they are not preconnected at such looping points along their route. When unraveled, the trail is always a single linear segment with no contained cycles". Based on this, a mistaken understanding on *p*-trails is that "any acyclic structure can provide at most one path for a failure that is off the structure itself and none for failures that are on the structure itself" [11]. Fig. 4 gives a counterexample where the *p*-trail is pre-cross-connected in a linear structure. It can protect 2 units of traffic on (0, 4), 1 unit on (0, 1), and so on. In fact, *p*-cycles can be regarded as a special case of *p*-trails where the two trail endpoints collocate. With our earlier analysis (see Fig. 3), it is easy to see that a *p*-trail can also protect more than 2 units of traffic on a particular span.

Additional benefit of using non-simple *p*-cycles or *p*-trails can be seen from the example in Fig. 5. If we only allow simple *p*-cycles, five simple *p*-cycles with a total cost of 18 are required for 100% protection, as shown in Fig. 5a. With the non-simple *p*-cycle in Fig. 5b, the required spare capacity can be cut down by 33.33% with a cost of 12. In Fig. 5a, each of the spans (0, 3), (0, 9) and (3, 9) straddles two different dotted *p*-cycles and thus is not a straddling span of any dotted *p*-cycle. The three spans can only be protected by the two dashed *p*-cycles. But the single non-simple *p*-cycle in Fig. 5b can protect them without any additional cost, because they are now straddling spans of the non-simple *p*-cycle. Note that the dotted *p*-cycles in both figures traverse the same set of spans but with different pre-cross-connection patterns at nodes 1 and 6. This example shows that, if several (simple or non-simple) *p*-cycles can be merged into a single non-simple *p*-cycle by rearranging the pre-cross-connection at some nodes, the links straddling two original *p*-cycles can be protected by the merged

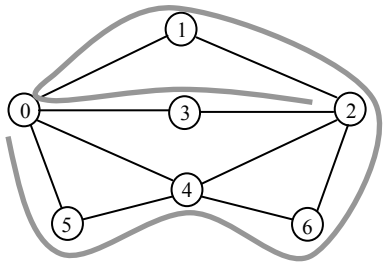
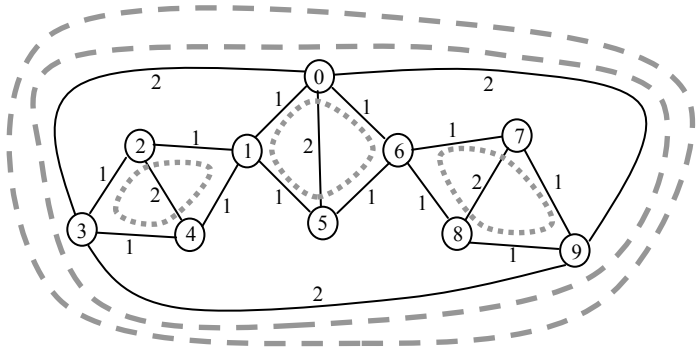
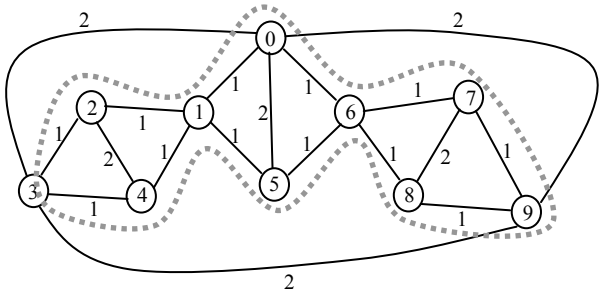


Fig. 4. The *p*-trail can protect 2 units of traffic on (0, 4), and 1 unit on (0, 1), (1, 2), (2, 3), (2, 4) and (0, 5).



(a) Optimal solution with 5 simple *p*-cycles (total cost: 18).



(b) Optimal solution with 1 non-simple *p*-cycle (total cost: 12).

Fig. 5. Advantage of non-simple *p*-cycles and *p*-trails on protection capability.

non-simple *p*-cycle.

The above observations show that non-simple *p*-cycles and *p*-trails provide higher protection flexibility than simple *p*-cycles. To fully exploit the advantages of non-simple *p*-cycles and *p*-trails, all the above features must be properly considered. To this end, existing work fails to identify those distinct features. This motivates us to formulate proper ILPs for optimal design of non-simple *p*-cycles and *p*-trails, such that we can accurately investigate their capacity gain over simple *p*-cycles. Meanwhile, we should not follow the conventional two-step approach as the candidate cycle/trail enumeration is a formidable task.

III. ILPs for Non-Simple *p*-Cycle and *p*-Trail Design

Without loss of generality, we consider an undirected network. Two scenarios are taken into account for comparison: a non-simple *p*-cycle/*p*-trail can traverse a span at most once per direction, or at most once in either direction. To facilitate our ILP formulations on both scenarios, we map each span (u, v) onto two directed vectors (u, v) and (v, u). If a vector is traversed by a *p*-cycle/*p*-trail, it is defined as an on-cycle/on-trail vector, and the corresponding span is an on-cycle/on-trail span. By our definition, if a *p*-cycle/*p*-trail can traverse a span (u, v) at most once per direction, then both vectors (u, v) and (v, u) can be an on-cycle vector of this *p*-cycle/*p*-trail. Otherwise, at most one vector (either (u, v) or (v, u)) can be an on-cycle vector.

A. General Idea

We first consider non-simple *p*-cycles. In an ILP formulation, non-simple cycles can be defined by requiring each node in the network to have an equal number (including zero) of inbound and outbound on-cycle vectors incident on it. With this definition, multiple disjoint cycles may be generated at a time, where two disjoint cycles do not traverse any common node. Fig. 6 shows an example of two disjoint cycles, with on-cycle vectors indicated by the broad-brush arrows. We call the set of disjoint cycles generated from the above definition as a *cycle set* $\boldsymbol{S_j}$ with index $j \in \{1, 2, \ldots, J\}$, where J is the maximum number of cycle sets allowed in the solution. An on-cycle or straddling span of any cycle in $\boldsymbol{S_j}$ is called an on-cycle or straddling span of $\boldsymbol{S_j}$. If a cycle set $\boldsymbol{S_j}$ is chosen to provide *p*-cycles, it can protect all its on-cycle and straddling spans. In Fig. 6, span (b, c) straddles two disjoint *p*-cycles. By our definition, it is not a straddling span of $\boldsymbol{S_j}$ and is not protected by $\boldsymbol{S_j}$. For simplicity, we say that a node or span is on $\boldsymbol{S_j}$ if it is traversed by any cycle in $\boldsymbol{S_j}$.

To determine how many units of traffic on a span (a, b) (defined as the *target span*) can be protected by a particular $\boldsymbol{S_j}$, we carry out a flow-based analysis. In our approach, a vector (u, v) can support at most one (unit) flow from node u to node v, and all the flows must move along the on-cycle vectors of $\boldsymbol{S_j}$. Let span (a, b) in Fig. 6 be the target span. Our flow-based analysis includes two steps. In the first step, we assume a as a source and b as a sink. Source a can emanate at most one flow onto each of its outbound vectors, but it cannot receive any flow. In Fig. 6, source a emanates two flows F_1 and F_2. F_3 does not exist because otherwise it will loop back to a. For an on-cycle vector (u, v), if u is not the source, a flow on (u, v) must come from a previous hop on-cycle vector (t, u). Similarly, if node v of an on-cycle vector (u, v) is not the sink, a flow on (u, v) must be forwarded to a next hop on-cycle vector (v, w). The pre-cross-connection patterns at nodes u and v can be formulated accordingly. All flows can only be terminated at sink b. In the second step, we assume b as a source and a as a sink, and go through the same process above. As shown in Fig. 6, two (dotted) flows F_4 and F_5 are found from b to a. Taking both steps into account, the non-simple *p*-cycle in Fig. 6 can support four distinct flows F_1, F_2, F_4 and F_5 between a and b. Note that F_4 is on the target span (a, b). If (a, b) fails, the number of traffic units on it that can be protected by $\boldsymbol{S_j}$ is $4-1=3$, with three backup paths indicated by F_1, F_2 and F_5.

The above flow-based analysis can be extended to *p*-trail design by removing the cycle constraint. $\boldsymbol{S_j}$ is now called a *trail set*. Fig. 7 shows a trail set with two disjoint *p*-trails. The *p*-trail traversing node h (*head*) and node t (*tail*) is an open trail, and the other *p*-trail is a closed trail (i.e., a cycle). In fact, any node on a closed trail (e.g., node d in Fig. 7) can be regarded as both the head and the tail of the *p*-trail. After the cycle constraint is removed, *p*-trails can be properly formulated with the same two-step approach based on the flow conservation principle at each node (except the source and the sink).

Consider the target span (a, b) in Fig. 7. To determine the

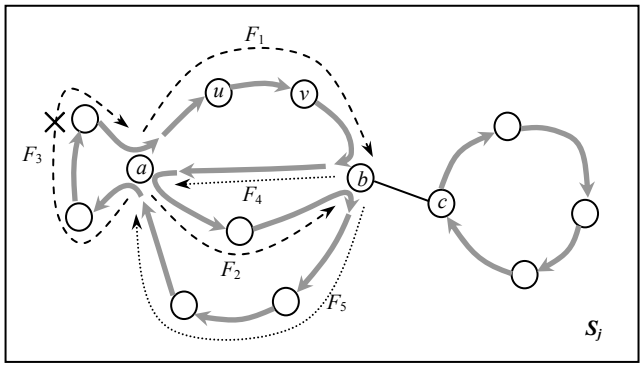


Fig. 6. Flows on a cycle set $\boldsymbol{S_j}$.

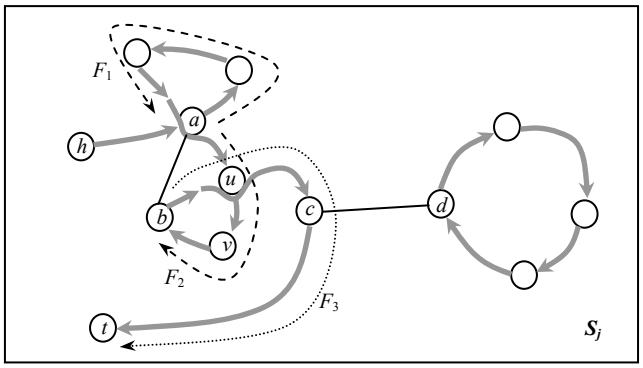


Fig. 7. Flows on a trail set $\boldsymbol{S_j}$.

number of traffic units on it that can be protected by $\boldsymbol{S}_j$, we assume a as a source and b as a sink, and vice versa. Among the three possible flows F_1, F_2 and F_3 in Fig. 7, only F_2 can exist. Otherwise, either F_1 generated by a will loop back to a, or F_3 generated by b will violate flow conservation at t and cannot be terminated at node a. As a result, $\boldsymbol{S}_j$ can protect one unit of traffic on (a, b), with the backup path indicated by F_2. If (c, d) in Fig. 7 is considered, no flow can exist between nodes c and d, and thus span (c, d) cannot be protected by $\boldsymbol{S}_j$.

In Fig. 7, vector (h, a) is not pre-cross-connected to any vector at head h, and vector (c, t) is not pre-cross-connected to any vector at tail t. It is different from the p-cycle scenario, where any on-cycle vector must be pre-cross-connected to another on-cycle vector at both end nodes. This complicates our formulation on the pre-cross-connection at the head/tail of the p-trail. To tackle this issue, we formulate the pre-cross-connection based on flows instead of vectors. Specifically, if there is a flow on an on-trail vector (u, v) and v is not the sink, then (u, v) must be pre-cross-connected to another vector (v, w) at node v, and the flow must be forwarded to (v, w). Similarly, if there is a flow on (u, v) and u is not the source, then (u, v) must be pre-cross-connected to another on-trail vector (t, u) at node u, and the flow must come from (t, u). Based on flow F_2 in Fig. 7 (with source a and sink b), we can determine that on-trail vector (a, u) is pre-cross-connected to (u, v) at node u, and (u, v) is pre-cross-connected to (v, b) at node v. Note that the pre-cross-connection at other nodes of the p-trail is not determined from F_2. In fact, the pre-cross-connection at other nodes can be determined from other flows when we consider other target spans. In the extreme case, if (h, t) is the target span and a flow exists on the p-trail from h to t, the pre-cross-connection at all nodes on this p-trail can be determined based on this flow.

B. Notation List

J: The maximum number of cycle/trail sets $\boldsymbol{S}_j$ in the solution.

j: $\boldsymbol{S}_j$ index where $j \in \{1, 2, \ldots, J\}$.

$\boldsymbol{E}$: The set of all the vectors. Each span (u, v) in the network is mapped onto two directed vectors (u, v) and (v, u).

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

c_{uv}: The cost of adding a backup wavelength to span (u, v). Either hop-count or distance related cost can be used as the cost metric and $c_{uv}=c_{vu}$.

L: Predefined hop-count/length limit of each p-cycle/p-trail.

d_{uv}: Traffic load on span (u, v) and $d_{uv}=d_{vu}$.

e_{uv}^{j}: Binary variable. It takes 1 if a vector (u, v)$\in \boldsymbol{E}$ is an on-cycle/on-trail vector in $\boldsymbol{S}_j$, and 0 otherwise.

p_{uvw}^{j}: Binary variable. It takes 1 if the backup wavelengths on two on-cycle/on-trail vectors (u, v) and (v, w) are pre-cross-connected in $\boldsymbol{S}_j$, and 0 otherwise.

$f_{uv}\big|_{ab}^{j}$: Binary variable. Assume span (a, b) is the target span with source a and sink b. It takes 1 if there is a flow on vector (u, v)$\in \boldsymbol{E}$ in $\boldsymbol{S}_j$, and 0 otherwise. Note that $f_{uv}\big|_{ba}^{j}$ assumes node b as the source and node a as the sink.

C. ILP Formulation for Non-Simple p-Cycle Design

For a network with given traffic load d_{uv} and cost c_{uv} on each span (u, v), a p-cycle solution for 100% protection against any single span failure can be generated by the ILP below. It allows non-simple p-cycles.

$$\text{minimize}\left\{\sum_{j}\sum_{(u,v)\in \boldsymbol{E}} c_{uv} e_{uv}^{j}\right\} \tag{1}$$

$$f_{uv}\big|_{ab}^{j} \le e_{uv}^{j}, \quad \forall (a,b),(u,v) \in \boldsymbol{E},\ \forall j; \tag{2}$$

$$\sum_{(u,a)\in \boldsymbol{E}} f_{ua}\big|_{ab}^{j} = 0, \quad \forall (a,b) \in \boldsymbol{E},\ \forall j; \tag{3}$$

$$f_{uv}\big|_{ab}^{j} - f_{vw}\big|_{ab}^{j} \le 1 - p_{uvw}^{j}, \quad \forall (a,b),(u,v),(v,w) \in \boldsymbol{E}: v \ne a, v \ne b,\ \forall j; \tag{4}$$

$$\sum_{j}\left(\sum_{(a,v)\in \boldsymbol{E}} f_{av}\big|_{ab}^{j} + \sum_{(b,v)\in \boldsymbol{E}} f_{bv}\big|_{ba}^{j} - e_{ab}^{j} - e_{ba}^{j}\right) \ge d_{ab}, \quad \forall (a,b) \in \boldsymbol{E}; \tag{5}$$

$$\sum_{(v,w)\in \boldsymbol{E}} p_{uvw}^{j} \le 1, \quad \forall (u,v) \in \boldsymbol{E},\ \forall j; \tag{6}$$

$$\sum_{(t,u)\in \boldsymbol{E}} p_{tuv}^{j} \le 1, \quad \forall (u,v) \in \boldsymbol{E},\ \forall j; \tag{7}$$

$$p_{uvw}^{j} \le \frac{1}{2}\left(e_{uv}^{j} + e_{vw}^{j}\right), \quad \forall (u,v),(v,w) \in \boldsymbol{E},\ \forall j; \tag{8}$$

$$\sum_{(t,u)\in \boldsymbol{E}} e_{tu}^{j} = \sum_{(u,v)\in \boldsymbol{E}} e_{uv}^{j}, \quad \forall u \in \boldsymbol{V},\ \forall j; \tag{9}$$

$$e_{uv}^{j} \le \sum_{(v,w)\in \boldsymbol{E}} p_{uvw}^{j}, \quad \forall (u,v) \in \boldsymbol{E},\ \forall j; \tag{10}$$

Objective (1) aims at minimizing the total cost of all p-cycles/cycle sets. Constraint (2) requires all flows to move on the on-cycle vectors. For a target span (a, b) in $\boldsymbol{S}_j$ with source a and sink b, constraint (3) prevents source a from receiving any flow. Constraint (4) means that, if vector (u, v) is pre-cross-connected to another vector (v, w) and node v is not the source a or the sink b, then a flow on (u, v) must be forwarded to (v, w). This constraint also allows the flows to be generated at source a and terminated at sink b. Based on our two-step flow-based analysis, constraint (5) ensures 100% span protection. Constraint (6) specifies that a vector (u, v) can be pre-cross-connected to at most one vector (v, w) at node v. Similarly, constraint (7) specifies that a vector (u, v) can be pre-cross-connected to at most one vector (t, u) at node u. Constraint (8) indicates that vectors (u, v) and (v, w) can be pre-cross-connected only if both of them are on-cycle vectors. Constraint (9) defines $\boldsymbol{S}_j$ by requiring each node in the network

to have an equal number of inbound and outbound on-cycle vectors incident on it. Finally, constraint (10) says that any on-cycle vector (u, v) must be pre-cross-connected to another on-cycle vector (v, w) at node v.

The ILP formulated in (1)-(10) allows a p-cycle to traverse a span at most once per direction. This is because both e^j_{uv} and e^j_{vu} of a span (u, v) may take 1 at the same time, and it is possible that u=w for the pre-cross-connection variable p^j_{uvw}. If we only allow a p-cycle to traverse any span at most once in either direction, we can add constraint (11) below.

$$e^j_{uv} + e^j_{vu} \le 1, \quad \forall (u,v) \in \boldsymbol{E}, \ \forall j; \tag{11}$$

Assume J is predefined large enough. The following constraint (12) can impose a hop-count or length limit L on each p-cycle.

$$\sum_{(u,v)\in E} c_{uv} e^j_{uv} \le L, \quad \forall j; \tag{12}$$

D. ILP Formulation for p-Trail Design

In p-trail design, we remove the cycle constraint (9). At the same time, constraint (10) is replaced by the following two constraints (13) & (14). In other words, our ILP for p-trail design includes (1)-(8) and (13)-(14).

$$f_{uv}\Big|^j_{ab} \le \sum_{(v,w)\in \boldsymbol{E}} p^j_{uvw}, \quad \forall (a,b),(u,v) \in \boldsymbol{E}: v \ne b, \ \forall j; \tag{13}$$

$$f_{uv}\Big|^j_{ab} \le \sum_{(t,u)\in \boldsymbol{E}} p^j_{tuv}, \quad \forall (a,b),(u,v) \in \boldsymbol{E}: u \ne a, \ \forall j; \tag{14}$$

Of course, constraint (11) can be included to require that a span is traversed at most once in either direction, and constraint (12) can be used to limit the hop-count or length of each p-trail.

Note that a p-trail may have separate head and tail where the p-trail terminates instead of being pre-cross-connected to another vector. To avoid distinguishing head and tail from other nodes on the p-trail, we formulate the pre-cross-connection based on flows as in (13)-(14). Constraint (13) says that, if there is a flow on (u, v)$\in \boldsymbol{E}$ and v is not the sink b, then (u, v) must be pre-cross-connected to another vector (v, w) at node v. Similarly, constraint (14) says that, if there is a flow on (u, v)$\in \boldsymbol{E}$ and u is not the source a, then (u, v) must be pre-cross-connected to another vector (t, u) at node u.

E. Discussion

In our ILPs, generally we need to set J large enough and the solution may contain less cycle or trail sets. If J is too small, the ILP may not be able to find a feasible solution, or the optimality of the solution cannot be ensured. But a larger J means more ILP variables which slow down the optimization process. Fig. 8 shows an example of how the ILP running time can be affected by the value of J, where we can also see that the ILP running time for non-simple p-cycle/p-trail design is greatly larger than that for simple p-cycle design. Note that the simple p-cycle solutions in this paper are obtained using the ILP in [8] (instead of the one in [2]). Besides, a proper value of J for non-simple p-cycle and p-trail design can be predefined in the same way as in [8]. It is only slightly smaller than that for simple p-cycle design [8]. Although a non-simple p-cycle/p-trail may protect more working capacity units on some spans, such spans are only a small part of all the spans protected, whereas for most spans the non-simple p-cycle or p-trail provides the same protection capability as a simple p-cycle.

Solution	Link traversal times	Total cost	Running time
Simple p-cycle	Once	13	1.73 sec
Non-simple p-cycle	Once	12	619.72 sec
	Once per direction	12	1972.08 sec
p-trail	Once	11	1117.28 sec
	Once per direction	11	2532.64 sec

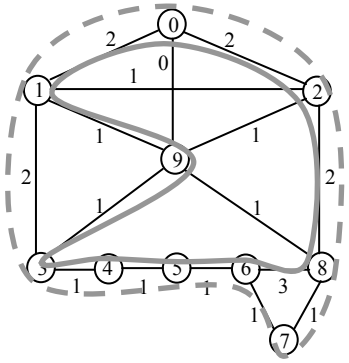
Fig. 8. ILP running time for the network in Fig. 10 with J=5. As compared with Fig. 10d (J=2), a larger value of J leads to a much longer ILP running time. Meanwhile, optimal solutions are ensured as long as J is larger than the necessary number of p-cycles/p-trails.

IV. Numerical Results and Discussions

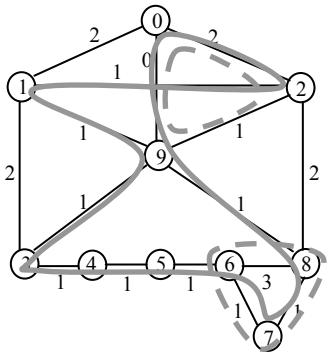
We use hop-count as the span cost metric for all the examples (i.e., c_{uv}=1 for each span (u, v)). The ILPs are implemented using ILOG CPLEX 11.0 on a server with 3GHz Intel Xeon CPU 5160. To carry out accurate comparison among simple, non-simple p-cycles and p-trails, all the ILP problems are solved to generate optimal solutions.

We first consider the network in Fig. 9, where a number next to each span indicates the amount of traffic units on that span (same for other examples). Though we use directed vectors to facilitate our ILP formulations, the direction of the vectors is not important in an undirected network and is thus ignored. Fig. 9a gives a simple p-cycle solution obtained using the cycle exclusion based ILP in [8]. Figs. 9b & 9c give two non-simple p-cycle solutions, and Figs. 9d & 9e provide two p-trail solutions. In particular, we allow a non-simple p-cycle/p-trail to traverse a span at most once (in either direction) in Figs. 9b & 9d, and at most twice (once per direction) in Figs. 9c & 9e. Fig. 9f summarizes the results. We can see that the non-simple p-cycle solutions in Figs. 9b & 9c are more capacity-efficient than the simple p-cycle solution in Fig. 9a. By allowing a span to be traversed twice, a solution better than Fig. 9b is obtained in Fig. 9c. Similar observations can be obtained from Figs. 9d & 9e, where the p-trail solutions achieve better capacity efficiency than the non-simple p-cycle solutions in Figs. 9b & 9c. Compared with the simple p-cycle solution in Fig. 9a, the p-trail solution in Fig. 9e saves 22.22% spare capacity.

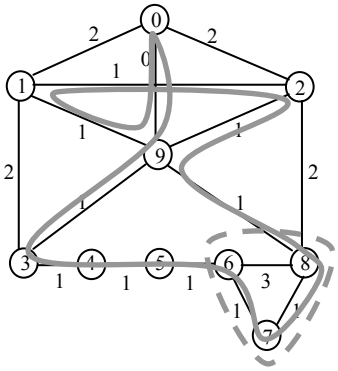
In Figs. 9c & 9e, span (0, 9) is traversed twice. We notice that the traffic load on (0, 9) is 0. Otherwise the solid p-cycle in Fig. 9c or the solid p-trail in Fig. 9e will not be able to protect the traffic on span (0, 9) (but the solid p-cycle/p-trail in Figs. 9b & 9d will). In our experiments, we also set positive integers as



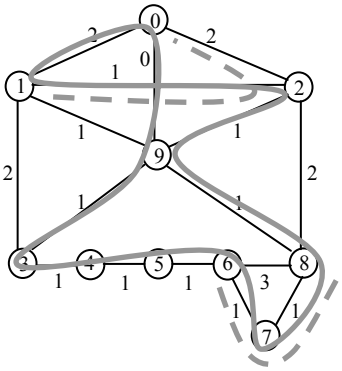
(a) Simple *p*-cycle solution.



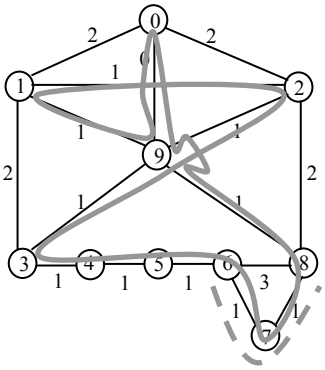
(b) Non-simple *p*-cycle solution with each span traversed at most once by each *p*-cycle.



(c) Non-simple *p*-cycle solution with each span traversed at most once per direction by each *p*-cycle.



(d) *p*-trail solution with each span traversed at most once by each *p*-trail.



(e) *p*-trail solution with each span traversed at most once per direction by each *p*-trail.

Fig.	Total cost	Running time
(a)	18	1.11 sec
(b)	17	246.95 sec
(c)	15	1004.19 sec
(d)	15	472.59 sec
(e)	14	941.20 sec

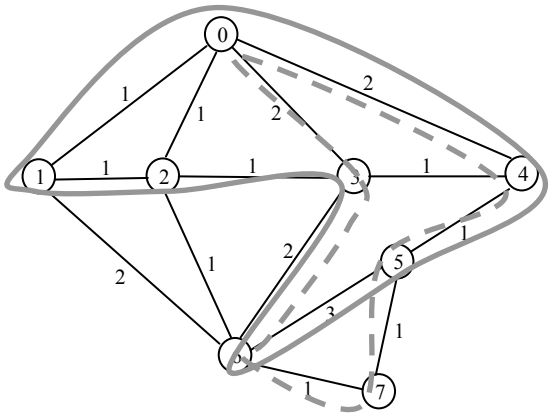
(f) Comparison (*J*=2).

Fig.9. ILP solutions for a simple network based on different fast protection schemes (*J*=2).

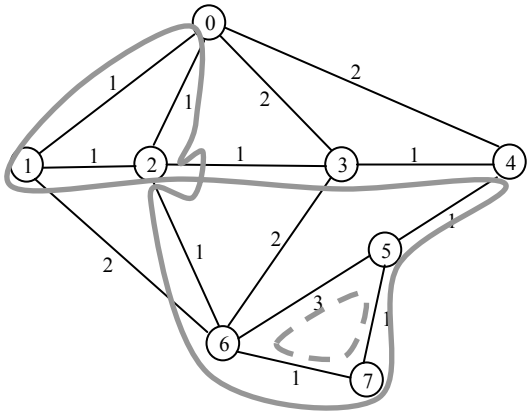
the traffic load on span (0, 9). It turns out that the capacity gain due to non-simple *p*-cycles and *p*-trails will be diminished in this case. In fact, allowing a *p*-cycle/*p*-trail to traverse a span multiple times can lead to higher design flexibility, but it also leads to several drawbacks: 1) multiple units of backup wavelengths are required on the span, which will be disrupted at the same time if the span fails. As a result, the *p*-cycle/*p*-trail will be broken into multiple pre-cross-connected segments which cannot protect any traffic on the failed span; 2) additional wavelength converters may be required to support multiple traversal times on a particular span, which will increase both the network cost and the management efforts; and 3) allowing multiple traversal times on a span will dramatically increase the ILP running time, as shown in Fig. 9f (and also other examples presented later). In fact, for simplicity of the performance comparison, we have ignored the cost of additional wavelength converters in the ILPs. Otherwise the ILPs will be more complex with much longer running time.

Fig. 10 shows another example. With the solid simple *p*-cycle in Fig. 10a, one unit of traffic on spans (0, 4), (3, 6), (6, 7), (5, 7) and two units of traffic on span (5, 6) are still unprotected. As a result, the dashed simple *p*-cycle is required for 100% span protection. In contrast, the solid non-simple *p*-cycle in Fig. 10b can better explore mesh connectivity of the network, and it only leaves one unit of unprotected traffic on span (5, 6). Under the cycle constraint, an additional dashed *p*-cycle in Fig. 10b is required for 100% span protection. The *p*-trail solution in Fig. 10c only needs a (dashed) backup path by removing the cycle constraint. Fig. 10d summarizes our experiments, where the capacity efficiency is not improved at all by allowing a *p*-cycle/*p*-trail to traverse a span twice (once per direction). On the other hand, Fig. 10d reinforces our earlier observation that the ILP running time will be increased if we allow multiple traversal times on the spans. Note that our ILPs do not allow a span (u, v) to be traversed twice in the same direction. Otherwise the two vectors on span (u, v) cannot be distinguished by e^j_{uv} and e^j_{vu}. The same issue exists if a span is allowed to be traversed more than twice. Though our approach can be extended to such more complex scenarios, they are not considered in this paper due to the ILP complexity and the marginal spare capacity saving.

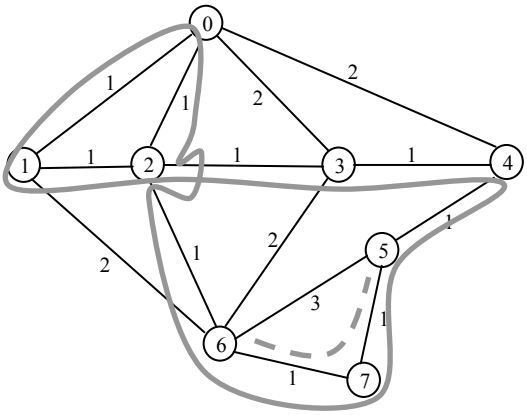
So far we have shown multiple examples where non-simple *p*-cycles and *p*-trails can lead to significant spare capacity saving. In those examples, both the network size and the traffic load on each span are small. An important concern is whether non-simple *p*-cycles and *p*-trails can keep their superior performance as network size and traffic load increase. To address this issue, we randomly generate various networks with different topologies and traffic loads, and then use our ILPs to find the corresponding optimal solutions. Due to the



(a) Simple *p*-cycle solution.



(b) Non-simple *p*-cycle solution.



(c) *p*-trail solution.

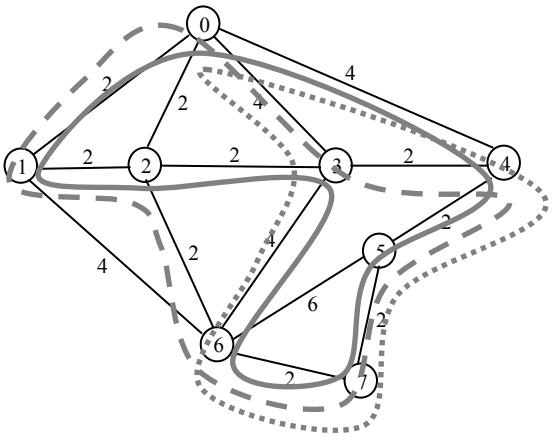
Solution	Link traversal times	Total cost	Running time
Simple *p*-cycle	Once	13	0.17 sec
Non-simple *p*-cycle	Once	12	81.33 sec
	Once per direction	12	177.42 sec
p-trail	Once	11	128.73 sec
	Once per direction	11	202.56 sec

(d) Comparison (J=2).

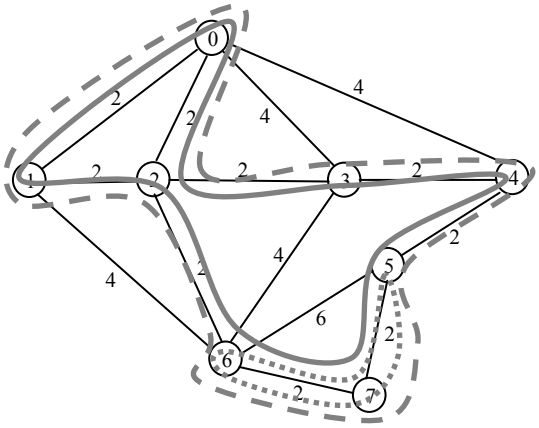
Fig. 10. An example where allowing multiple traversal times on the spans does not improve the solution quality.

complexity of non-simple *p*-cycle/*p*-trail design and the limitation of the currently available computation capability, optimal solutions can be obtained in reasonable running time only for those networks with small or medium network size (less than 30 nodes) and traffic load ($J \leq 10$). In our experiments, we find that the capacity gain due to non-simple *p*-cycles and *p*-trails becomes marginal (or no gain at all for most tests) as network size and traffic load increase, even though we have fully exploited the potential benefits of non-simple *p*-cycles and *p*-trails by considering those distinct features presented in Section II. Fig. 11 gives an example where the network topology is the same as that in Fig. 10 but the traffic load on each span is doubled. Unlike Fig. 10 where a significant spare capacity saving due to non-simple *p*-cycles/*p*-trails can be observed, Fig. 11c indicates that all the solutions require the same amount of spare capacity (i.e., total cost) of 22 for 100% span protection, no matter whether the solution is based on simple or non-simple *p*-cycles, or *p*-trails. It also does not matter whether we allow a span to be traversed by a *p*-cycle/*p*-trail at most once (see Fig. 11a), or at most once per direction (see Fig. 11b). Note that the examples reported are only typical ones for illustrating our key points, whereas various other examples with similar observations can be reproduced by solving the ILPs formulated in this paper. Based on our experiments, non-simple *p*-cycles and *p*-trails can lead to significant spare capacity saving for some networks with small network size and light traffic load, but generally the gain over the corresponding simple *p*-cycle solutions becomes trivial as network size and traffic load increase.

A careful study based on the above observations reveals that each of the protection structures (simple *p*-cycle, non-simple *p*-cycle and *p*-trail) has its own pros and cons. In particular, a simple *p*-cycle has a much simpler structure of optical pre-cross-connection than a non-simple *p*-cycle or *p*-trail, and thus inherently intends to require less amount of spare capacity. But it is subject to the cycle constraint, and cannot explore mesh connectivity as flexibly as a non-simple *p*-cycle or *p*-trail does. On the other hand, a non-simple *p*-cycle or *p*-trail may protect more than two units of working capacity on some spans. This is generally due to its more complex optical pre-cross-connection, which inherently intends to consume more spare capacity. In addition, those spans with more traffic units protected are only a small part of all the spans protected by the non-simple *p*-cycle/*p*-trail, and thus non-simple



(a) Optimal solution with each span traversed at most once by a *p*-cycle/*p*-trail.



(b) Optimal solution with each span traversed at most once per direction by a *p*-cycle/*p*-trail.

Solution	Link traversal times	Total cost	Running time
Simple *p*-cycle	Once	22	0.63 sec
Non-simple *p*-cycle	Once	22	579.56 sec
	Once per direction	22	5075.02 sec
p-trail	Once	22	4031.22 sec
	Once per direction	22	41951.63 sec

(c) Summary of the experiment with J=5.

Fig. 11. Doubling the traffic load on each span in Fig. 9 makes all the *p*-cycle/*p*-trail solutions with the same total cost.

p-cycle/*p*-trail can gain only little from this distinct feature. Although non-simple *p*-cycles and *p*-trails can better explore mesh connectivity of a network and thus may lead to some spare capacity saving, this effect generally becomes marginal in a practical network with a moderate average nodal degree. Instead, if the network size is small (as in Fig. 2) or the network nodes are sparsely connected (as in Fig. 5), then non-simple *p*-cycles and *p*-trails may achieve a quite significant capacity gain, because the flexibility of using simple *p*-cycles is greatly limited in this case. Besides, as the traffic loads increase, more *p*-cycles or *p*-trails are required for 100% span protection. For a simple *p*-cycle solution, a larger number of required *p*-cycles implies more flexibility in a combinatorial sense to minimize the required spare capacity. This explains why the capacity gain due to non-simple *p*-cycles and *p*-trails becomes marginal as the traffic load increases.

V. Conclusion

The paper investigated non-simple *p*-cycles and *p*-trails by identifying their distinct features which have never been explored in previous studies. We formulated a set of optimal ILP models for non-simple *p*-cycle and *p*-trail design. Our ILP models are characterized by removing candidate cycle/trail enumeration and taking all the identified features of non-simple *p*-cycles and *p*-trails into account, where a true optimal design can be achieved. Numerical results showed that in some small-size and lightly-loaded networks, non-simple *p*-cycles and *p*-trails can yield significant spare capacity saving over simple *p*-cycles, but the capacity gain is generally trivial as network size and traffic load increase. By analyzing optimal ILP solutions obtained in various networks, we also provided in-depth insights to explain our findings.

References

[1] T. Y. Chow, F. Chudak and A. M. Ffrench, "Fast optical Layer mesh protection using pre-cross-connected trails," *IEEE/ACM Transactions on Networking*, vol. 12, no. 3, pp. 539-548, Jun. 2004.

[2] 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.

[3] 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.

[4] C. G. Gruber, "Resilient networks with non-simple *p*-cycles," *IEEE ICT '03*, vol. 2, pp. 1027-1032, Feb. 2003.

[5] 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.

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

[7] C. Liu and L. Ruan, "Finding good candidate cycles for efficient *p*-cycle network design," *IEEE ICCCN '04*, pp. 321-326, 2004.

[8] B. Wu, K. L. Yeung and P.-H. Ho, "ILP formulations for *p*-cycle design without candidate cycle enumeration," *IEEE/ACM Trans. Netw.*, to appear. http://www.eee.hku.hk/research/doc/tr/TR2008001_IFDCC.pdf.

[9] D. A. Schupke, "An ILP for optimal *p*-cycle selection without cycle enumeration," *8th Conference on Optical Network Design and Modelling ONDM '04*, 2004.

[10] H. N. Nguyen, D. Habibi, V. Q. Phung, S. Lachowicz, K. Lo and B. Kang, "Joint optimization in capacity design of networks with *p*-cycle using the fundamental cycle set," *IEEE GLOBECOM '06*, Nov. 2006.

[11] 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.

Bin Wu received the B.Eng. degree from Zhe Jiang University, Hangzhou, China, in 1993, M.Eng. degree from University of Electronic Science and Technology of China, Chengdu, China, in 1996, and PhD degree from the University of Hong Kong, Hong Kong, in 2007. During 1997-2001, he served as the department manager of TI-Huawei DSP co-lab in Huawei Tech. Co. Ltd, Shenzhen, China. Currently he is a postdoctoral research fellow at the University of Waterloo, Waterloo, Canada.

Kwan L. Yeung was born in 1969. He received his B.Eng. and Ph.D. degrees in Information Engineering from The Chinese University of Hong Kong in 1992 and 1995, respectively. He joined the Department of Electrical and Electronic Engineering, The University of Hong Kong in July 2000, where he is currently an Associate Professor, and the Information Engineering Program Co-Director. Before that, he has spent five years in the Department of Electronic Engineering, City University of Hong Kong as an Assistant Professor. During the summer of 1993, Dr. Yeung served with the Performance Analysis Department, AT&T Bell Laboratories (now Bell Labs, Lucent Technologies), Holmdel, USA, as a Member of Technical Staff. Dr. Yeung's research interests include next-generation Internet, packet switch/router design, all-optical networks and wireless data networks. He has obtained two patents and published over 120 papers in international journals and conferences since 1993.

Pin-Han Ho received his B.Sc. and M.Sc. degrees from the Electrical and Computer Engineering, Department of National Taiwan University in 1993 and 1995, respectively. He started his Ph.D. studies in 2000 at Queen's University, Kingston, Ontario, Canada, focusing on optical communications systems, survivable networking, and QoS routing problems. He finished his Ph.D. in 2002, and joined the Electrical and Computer Engineering Department at the University of Waterloo as an assistant professor in the same year. He is the author/co-author of more than 100 refereed technical papers and book chapters, and the co-author of a book on optical networking and survivability. He is the recipient of the Distinguished Research Excellence Award in the ECE Department at the University of Waterloo, the Early Researcher Award in 2005, the Best Paper Award at SPECTS '02 and the ICC '05 Optical Networking Symposium, and the Outstanding Paper Award in HPSR '02.