ILP Formulations for $p$-Cycle Design without Candidate Cycle Enumeration

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

Abstract—The concept of $p$-cycle (Preconfigured Protection Cycle) allows fast and efficient span protection in WDM mesh networks. To design $p$-cycles for a given network, conventional algorithms need to enumerate cycles in the network to form a candidate set, and then use an ILP (Integer Linear Program) to find an optimal set of $p$-cycles from the candidate set. Because the size of the candidate set increases exponentially with the network size, candidate cycle enumeration introduces a huge number of ILP variables and slows down the optimization process. In this paper, we focus on $p$-cycle design without candidate cycle enumeration. Three ILPs for solving the problem of spare capacity placement (SCP) are first formulated. They are based on recursion, flow conservation, and cycle exclusion, respectively. We show that the number of ILP variables/constraints in our cycle exclusion approach only increases linearly with the network size. Then, based on cycle exclusion, we formulate an ILP for solving the joint capacity placement (JCP) problem. Numerical results show that our ILPs are very efficient in generating $p$-cycle solutions.

Index Terms—ILP (Integer Linear Program), protection, $p$-cycle (Preconfigured Protection Cycle), WDM (Wavelength Division Multiplexing) mesh networks.

I. INTRODUCTION

OPTICAL networks based on WDM (Wavelength Division Multiplexing) provide the backbone infrastructure for high-speed data communications. In WDM networks, hundreds of wavelengths can be multiplexed onto a single fiber for parallel data transmission. This not only fully utilizes the fiber bandwidth, but also efficiently reduces the network cost. Because of the high-speed nature of WDM networks, network survivability is of paramount importance. Upon an accidental failure such as a fiber cut, it is imperative that the network can achieve fast optical recovery in order to minimize data loss.

As optical network topology evolves from ring to mesh, the concept of $p$-cycle (Preconfigured Protection Cycle) [1] enables fast span/link protection with high capacity efficiency. The idea is to organize the spare capacity in the network into a set of preconfigured cycles to protect all the traffic on each span (i.e., 100% span protection). Unless otherwise specified, a $p$-cycle in this paper always refers to a unity-$p$-cycle, which is implemented by using one unit of spare capacity (or one wavelength) on each span it traverses. A span traversed by a $p$-cycle is called an on-cycle span of this $p$-cycle. If a span is not traversed by a $p$-cycle but its two end nodes are, then it is a straddling span of this $p$-cycle. In an undirected network with at most a single span failure at a time, a $p$-cycle can protect one unit of traffic on each on-cycle span and two units on each straddling span. Fig. 1 gives an example. Spans traversed by the dashed $p$-cycle are on-cycle spans. Other spans, i.e., (0, 1) and (3, 4), are straddling spans. If on-cycle span (2, 3) fails, one unit of traffic on (2, 3) can be protected by rerouting it to the other side of the $p$-cycle, or path 2→0~4→1~3. If straddling span (0, 1) fails, two backup paths 0→4→1 and 0→2→3→1 are available and thus two units of traffic on span (0, 1) can be protected. Note that the straddling spans do not have any spare capacity reserved, but the spare capacity reserved along the $p$-cycle can protect both the on-cycle and straddling spans. As a result, this link-based $p$-cycle protection scheme yields a high capacity efficiency comparable to SBPP (Shared Backup Path Protection) [2]. On the other hand, since the spare capacity on the $p$-cycle is preconfigured, only the two end nodes of the failed span need to fulfill real-time switching upon failure. This leads to a BLSR (Bidirectional Line Switched Ring) [3] ring-like fast recovery speed.

Because of its outstanding performance on both recovery speed and capacity efficiency, $p$-cycle has attracted extensive research interests [4-23] since it was first introduced in 1998 [1]. Recently, it was also extended to path/segment protection [24-25] at the cost of a slower recovery speed.

For a given network, $p$-cycle design [26] can be formulated as either the problem of spare capacity placement (SCP), or joint capacity placement (JCP). In SCP [1, 9], traffic load on each span is given. That means traffic demands have been properly routed according to some routing algorithm (such as...
shortest path routing). The objective of SCP is to minimize the spare capacity required for 100% span protection. In JCP [16-17], routing and spare capacity placement are jointly optimized to minimize the total capacity required.

A major issue in $p$-cycle design is its high complexity. Conventional algorithms follow a two-step approach. The first step enumerates all distinct cycles [27] in the network to form a candidate set. The second step chooses an optimal set of $p$-cycles from the candidate set based on an ILP (Integer Linear Program). Since the size of the candidate set soars exponentially as the network size increases, cycle enumeration is time-consuming. More importantly, a large candidate set incurs a huge number of variables in the ILP. This slows down the optimization process (if not making it intractable). To reduce the size of the candidate set, some heuristic cycle pre-selection algorithms [18-21] can be used, or the length of candidate cycles is limited. As a result, only a subset of all cycles is chosen for ILP optimization. Obviously, this will adversely affect the quality of the solution. More recently, a new approach was proposed [17] to generate $p$-cycle solutions using only fundamental cycles, defined as cycles without straddling links. This approach needs a preprocessing step to enumerate all fundamental cycles and construct the straddling link information. Besides, it cannot efficiently handle $p$-cycle design in non-planar networks.

In this paper, three new ILPs without candidate cycle enumeration are first formulated for solving the SCP $p$-cycle design problem. They follow three different approaches, recursion, flow conservation and cycle exclusion. We show that the number of ILP variables involved is drastically reduced. In particular, the number of variables and constraints in the cycle exclusion-based ILP only increases linearly with network size. This makes our ILPs very efficient. We further extend the cycle exclusion-based ILP to solve the problem of JCP $p$-cycle design.

The rest of the paper is organized as follows. Section II introduces the cycle set definition, which is an important concept in our ILP formulations. Some related work on ILP formulation without candidate cycle enumeration is also reviewed. In Section III, four ILPs are formulated, three for solving the SCP problem and one for the JCP problem. Section IV gives numerical results and Section V concludes the paper.

II. CYCLE SET AND RELATED WORK

In an ILP formulation, cycles can be defined by requiring each node in the network to have either 2 or 0 on-cycle spans incident on it [22, 28]. With this definition, multiple disjoint cycles may be generated at a time, where two disjoint cycles do not have any common on-cycle span or node. Fig. 2 shows an example of two disjoint cycles. Without loss of generality, we call the set of cycles generated from the above definition as a cycle set $\text{CS}_j$, with index $j \in \{1, 2, \ldots, J\}$, where $J$ is the maximum number of cycle sets in a solution. We further define that if a span is an on-cycle or straddling span of any cycle in $\text{CS}_j$, it is an on-cycle or straddling span of $\text{CS}_j$. If a cycle set $\text{CS}_j$ is chosen to provide $p$-cycles, it can protect all its on-cycle and straddling spans. In Fig. 2, span $(c, d)$ is not a straddling span of $\text{CS}_j$ because it straddles two disjoint cycles. Therefore, $(c, d)$ is not protected by $\text{CS}_j$. For simplicity, we say that a node or span is on $\text{CS}_j$ if it is traversed by any cycle in $\text{CS}_j$.

To the best of our knowledge, Schupke’s ILP [22] is the only work for $p$-cycle design without cycle enumeration. It adopts a flow-based technique to ensure a single cycle in each $\text{CS}_j$. As a result, a span can be protected by the only cycle in $\text{CS}_j$ if its two end nodes are on $\text{CS}_j$. In Schupke’s ILP, a unique master node is defined on each cycle set $\text{CS}_j$, and it serves as the source of all flows, as shown in Fig. 3. Other nodes in the network are target nodes. Each target node $t$ receives a flow generated by the master node. If the flow can move along the on-cycle spans of $\text{CS}_j$ to reach $t$, then $t$ is on $\text{CS}_j$. Otherwise the flow must traverse some spans not on $\text{CS}_j$, and thus the target node is not on $\text{CS}_j$ (e.g. node $t'$ in Fig. 3).

Despite of its theoretical significance of removing candidate cycle enumeration, Schupke’s ILP needs a very long running time. This is due to four reasons: 1) the ILP must check all the master-target node pairs in the network to ensure a single cycle in $\text{CS}_j$; 2) the ILP must determine a specific master node on $\text{CS}_j$ in order to prove the optimality of the solution, though it does not matter which master node is chosen (as long as it is on $\text{CS}_j$); 3) since each $p$-cycle is not treated as a unity-$p$-cycle, the ILP must also determine the number of required copies of each cycle; 4) the number of variables and constraints in the ILP is a quadratic function of the number of nodes in the network. To reduce the running time of this ILP, a four-step heuristic is proposed in [22] for finding suboptimal solutions.
III. ILP FORMULATIONS

We formulate ILPs for solving both spare capacity placement (SCP) and joint capacity placement (JCP) problems. For easy reference, we summarize the notations in our ILPs in Fig. 4. We consider undirected networks with at most a single span failure at a time. We also assume there are enough wavelength channels and wavelength converters to support all necessary optical connections.

A. Recursion-Based ILP for SCP

We observe that it is not necessary to ensure a single cycle in each cycle set $CS_j$ (as in Schupke’s ILP [22]). When multiple disjoint cycles are present in a $CS_j$, a $p$-cycle solution can still be constructed if we can properly identify individual spans that can be protected by each $CS_j$.

In checking whether a span $(a, b)$ can be protected by a cycle set $CS_j$, we examine if there is a route on $CS_j$ connecting nodes $a$ and $b$, where the route can only consist of the on-cycle spans of $CS_j$. We refer this process as checking the connectivity between $a$ and $b$. If such a route exists, then nodes $a$ and $b$ are on the same cycle, or “$a$ connects to $b$” for short. In this case, span $(a, b)$ is either an on-cycle span or a straddling span of this cycle, and it can be protected by $CS_j$. Otherwise span $(a, b)$ cannot be protected by $CS_j$.

Motivated by classic routing algorithms such as Dijkstra’s and Floyd-Warshall algorithms [29], we design a recursive

---

**Fig. 4. Notations in the ILP formulations.**
process for connectivity checking. In Fig. 5, checking the connectivity between nodes a and b is equivalent to checking the connectivity between nodes a and v, because (v, b) is an on-cycle span. This further depends on the connectivity between nodes a and u if (u, v) is also an on-cycle span, and so on. We define node b as the starting node of the recursion. The nodes involved in the recursion are sequentially referred to in a unidirectional manner along the cycle, as indicated by the dashed arrows in Fig. 5. If a and b are on the same cycle, the recursion will be stopped when node a is referred to (defined as the first stop condition). Then, span (a, b) can be protected by CS_J. On the other hand, if span (a, b') in Fig. 5 is checked and node b' is the starting node, the recursion will be stopped when node b' is revisited (defined as the second stop condition). In this case, nodes a and b' are not connected and thus span (a, b') cannot be protected by CS_J.

Based on the above mechanism, an ILP for solving the SCP problem is formulated below. Note that for the SCP problem, traffic load l_v, and cost c_vw for each span (u, v) ∈ E are given.

\[
\begin{align*}
\text{minimize} & \quad \sum_j \sum_{(u,v) \in E} c_{uv} x_{uv}^j \\
\text{s.t.} & \quad \forall u \in V, \quad \forall j; \\
& \quad \forall (a,b) \in E, \quad \forall j; \\
& \quad \forall (a,v) \in E; \\
& \quad x_{uv}^j \mid_{(a,b)} = 1, \\
& \quad x_{uv}^j \mid_{(a,b)} = \frac{1}{2} \sum_{(u,v) \in E} y_{uv}^j, \\
& \quad \forall (a,v) \in E, \quad \forall j; \\
& \quad \forall (a,v) \in E; \\
& \quad \forall (u,v) \in E: u \neq a, v \neq a; \\
& \quad \forall (u,v) \in E: u \neq v, v \neq a.
\end{align*}
\]

Objective given in (1) aims at minimizing the total cost of all cycle sets/p-cycles. Constraint (2) defines a cycle set CS_J by requiring each node in the network to have either 2 or 0 on-cycle spans incident on it. Constraint (3) ensures 100% span protection, where \(2x_{uv}^j - e_{uv}^j\) gives the number of traffic units on span (u, v) that can be protected by CS_J. 2x_{uv}^j - e_{uv}^j takes 1 if (u, v) is an on-cycle span, 2 if (u, v) is a straddling span, and 0 otherwise. Constraints (4)-(10) form the recursion for checking whether span (a, b) can be protected by CS_J. Constraint (4) specifies that node a must always “connect” to itself. This provides the first stop condition for the recursion, i.e. when node a is referred to. Constraint (5) defines “a connects to v via u”, or \(y_{uv}^j \mid_{(a,b)} = 1\) only if a connects to u (i.e., \(x_{uv}^j \mid_{(a,b)} = 1\)) and (u, v) is an on-cycle span (i.e., \(e_{uv}^j = 1\)). Constraint (6) stipulates that, a connects to v on CS_J only if there exists a neighbor u of v such that “a connects to v via u”. Constraint (7) prohibits any possible loopback to the starting node b by preventing a to connect with any node via b. This provides the second stop condition for the recursion. Constraint (8) determines whether span (a, b) can be protected by CS_J. Constraint (9) means that, if there is no on-cycle span incident on node a, then we cannot find a route on CS_J between a and any other node in the network. Finally, constraint (10) ensures that the recursion is carried out in a unidirectional manner. Assume (u, v) is an on-cycle span, and we have referred to node u for checking the connectivity between nodes a and v. Constraint (10) says that, we will not refer back to node v when we further check the connectivity between nodes a and u.

Compared to Schupke’s ILP [22], the above recursion-based ILP runs much faster. Instead of checking all the node pairs to ensure a single cycle in each CS_J, it only checks every span in the network to see if it can be protected by a CS_J. Since each p-cycle is defined as a unity-p-cycle, our ILP does not calculate the required number of copies of each p-cycle. Approximatively, the ILP formulated in (1)-(10) contains \(J(2|E|+|V|)(|E|+1)\) variables and \(3|E|+ J(|E|+|V|+2|E|×|V|+|E|\) constraints (the exact number of variables and constraints depends on whether node a or b serves as the starting node). Note that J is the
maximum number of cycle sets allowed in the solution. Its value can be set according to (37).

B. Flow Conservation-Based ILP for SCP

Unlike the recursion-based ILP above, we now adopt the notion of flow conservation in checking whether span \((a, b)\) can be protected by CS\(_j\). Without loss of generality, we assume that node \(a\) is the source and node \(b\) is the sink. We check if a flow between \(a\) and \(b\) can be carried only by the on-cycle spans of CS\(_j\). (Note that Schupke’s ILP [22] allows a flow to move along any spans in the network.) Source \(a\) can generate at most one flow but it does not receive any flow. Similarly, sink \(b\) can receive at most one flow but it does not generate or relay any flow. Except source \(a\) and sink \(b\), all other nodes in the network must obey flow conservation [29]. If both \(a\) and \(b\) are on the same cycle as in Fig. 6, then a flow between \(a\) and \(b\) exists, and span \((a, b)\) can be protected by CS\(_j\). If span \((a, b')\) in Fig. 6 is checked instead, no flow exists between \(a\) and \(b'\), and span \((a, b')\) cannot be protected by CS\(_j\).

For an arbitrary node in the network, if an on-cycle span incident on it and this on-cycle span carries a flow, we say that there is a unit-flow incident on this node. To reduce the number of variables and constraints in ILP, we can use an undirected flow to replace the directed one in Fig. 6. Then, flow conservation can be formulated by requiring each node in the network (except source \(a\) and sink \(b\)) to have either 2 or 0 unit-flows incident on it, whereas \(a\) or \(b\) can have at most 1 unit-flow incident on it. With the notations defined in Fig. 4, the flow conservation-based ILP is formulated below for solving the SCP problem.

\[
\text{minimize} \left\{ \sum_j \sum_{(u,v) \in E} e_{uv}^i \right\},
\]  

subject to:

\[
\sum_{(u,v) \in E} e_{uv}^i = 2z_u^i, \quad \forall u \in V, \forall j; \tag{12}
\]

\[
\sum_j (2x_{uv}^i - e_{uv}^i) \geq l_{uv}, \quad \forall (u,v) \in E; \tag{13}
\]

\[
x_{ab}^i \leq \frac{1}{2}(z_a^i + z_b^i); \quad \forall (a,b) \in E, \forall j; \tag{14}
\]

\[
f_{uv}^i \leq e_{uv}^i, \quad \forall (u,v) \in E, \forall j; \tag{15}
\]

\[
\sum_{(u,v) \in E} f_{uv}^i \leq 1, \quad \forall (a,b) \in E, \forall j; \tag{16}
\]

\[
\sum_{(a,u) \in E} f_{au}^j \leq 1, \quad \forall (a,b) \in E, \forall j; \tag{17}
\]

\[
f_{ab}^j = 2k_{ab}^j, \quad \forall (a,b) \in E, \forall j; \tag{18}
\]

\[
\chi_{ab}^j \leq \sum_{(a,u) \in E} f_{au}^j, \quad \forall (a,b) \in E, \forall j; \tag{19}
\]

Objective in (11) aims at minimizing the total cost of all \(p\)-cycles (or cycle sets). Constraint (12) defines cycle sets and (13) ensures 100% span protection. Constraint (14) gives a necessary (but not sufficient) condition to identify those spans protected by CS\(_j\), i.e., a span can be protected by CS\(_j\) if only its two end nodes are on CS\(_j\). This constraint confines the solution space and thus speeds up the optimization process. Constraints (15)-(19) check whether a span \((a, b)\) can be protected by CS\(_j\) based on flow conservation concept. By (15), flows can only move along the on-cycle spans of CS\(_j\). Constraints (16) & (17) require source \(a\) and sink \(b\) to have at most 1 unit-flow incident on each. Constraint (18) formulates the flow conservation property for other nodes in the network. Finally, constraint (19) indicates that, \((a, b)\) can be protected by CS\(_j\) if there is a unit-flow incident on source \(a\). In fact, \((a, b)\) can be protected by CS\(_j\) if any flow exists on CS\(_j\).

Compared to the recursion-based ILP, the above flow conservation-based ILP is simpler. To check whether a span can be protected by a CS\(_j\), the former must retrieve nodal connectivity in a recursive manner, but the latter only examines the existence of a unit-flow on CS\(_j\). In other words, the former requires “node level” details but the latter only requires “path level” details. The flow conservation-based ILP in (11)-(19) involves \(|J|E^{2+|J|E}|V|+|J|V| \) variables and \(|J|E \times (|J|E + |V| + 2)+|J|V+|E| \) constraints. Both the number of variables and constraints are smaller than that in the recursion-based ILP.

C. Cycle Exclusion-Based ILP for SCP

Schupke’s ILP [22] pays great effort in ensuring a single cycle in each cycle set CS\(_j\), and in return, it is very simple in checking whether a span can be protected by a CS\(_j\). In Schupke’s ILP, span \((a, b)\) can be protected by CS\(_j\) if and only if both \(a\) and \(b\) are on the only cycle in CS\(_j\). In contrast, our recursion and flow conservation based ILPs greatly simplify cycle formulation by allowing multiple disjoint cycles in CS\(_j\), but the downside is that we need a more complex process to check whether each span \((a, b)\) can be protected by CS\(_j\). In the following, we formulate a cycle exclusion-based ILP to take the advantages from both approaches, i.e. simple cycle formulation.
For the Fig. 7c, we say that there is a call this common head a span. An on-cycle span (u → v) and easy protection checking.

We use a directed on-cycle vector to denote each on-cycle span. An on-cycle span (u, v) of CSj is denoted by either u → v or v → u, but not both. Note that the direction of the on-cycle vector is not important, as either one sufficiently denotes the on-cycle span (u, v). If a span in the network is not associated with an on-cycle vector, then it is not an on-cycle span of CSj. For the p-cycle in Fig. 7a, Figs. 7b & 7c give two possible representations using different sets of on-cycle vectors. For each on-cycle vector u → v, we define node u as the head and node v as the tail. A value called voltage is assigned to each node in the network, and its absolute value is not important. For each on-cycle vector u → v, we only require that its tail v has a larger voltage value than its head u. If two on-cycle vectors have a common tail on CSj, e.g. node v in Fig. 7b and node u in Fig. 7c, we say that there is a reversal at this node, and the common tail is called a reversal node. If two on-cycle vectors have a common head on CSj, e.g. node r in Figs. 7b & 7c, we call this common head a root node.

In Figs. 7b-7d, the number (fraction) next to each node denotes a possible voltage value for that node. If there is a single root-reversal node pair on the cycle (as in Figs. 7b & 7c), then a feasible set of voltage values exists. If a cycle does not have any root-reversal node pair (as in Fig. 7d), there must exist a voltage value conflict at some node on the cycle. An example is shown in Fig. 7d. If we start from node r and move along the direction of the on-cycle vectors, the nodal voltage value increases monotonically until we loop back to the starting point r. Node r should have a voltage value larger than 0.04 of node u. But at the same time, it should have a voltage value smaller than 0.01 of node t. This causes a conflict.

Based on the above observation, we can see that ensuring a single cycle in each CSj is equivalent to ensuring a unique root-reversal node pair in each CSj. When there are multiple cycles in CSj (as in Fig. 8), only the one with the root-reversal node pair remains, and all other cycles will be excluded to avoid voltage value conflicts. We call this process cycle exclusion. After cycle exclusion, a span can be protected by the only cycle in CSj if and only if its two end nodes are on CSj. With the notations defined in Fig. 4, the cycle exclusion-based ILP is formulated below for solving the SCP p-cycle design problem. Note that we still call it an ILP for simplicity, although a fractional relaxation on voltage values is used to speed up the optimization.

\[
\text{minimize } \sum_j \sum_{(u,v) \in E} c_{uv} (\theta^j_{uv} + \theta^j_{vu}) .
\]

s. t.

\[
\theta^j_{uv} + \theta^j_{vu} \leq 1 ,
\]

\[
\sum_{(u,v) \in E} (\theta^j_{uv} + \theta^j_{vu}) = 2x^j_u ,
\]

\[
\sum_j (2\chi^j_{uv} - \theta^j_{uv} - \theta^j_{vu}) \geq l^j_{uv} ,
\]

\[
\sum_{u \in V} l^j_u \leq 1 ,
\]

\[
\sum_{(u,v) \in E} \theta^j_{uv} \leq 1 + x^j_u ,
\]

\[
p^j_v - p^j_u \geq \alpha \times \theta^j_{uv} - (1 - \theta^j_{uv}) ,
\]

\[
\forall (u,v),(v,u) \in E , \forall j ;
\]

\[
\forall u \in V , \forall j ;
\]

\[
p^j_v - p^j_u \geq \alpha \times \theta^j_{uv} - (1 - \theta^j_{uv}) ,
\]

\[
\forall u \in V , \forall j ;
\]
\[
    x_{uv}^j = \frac{1}{2} (z_u^j + z_v^j),
\]

\[
    \forall (u,v) \in E, \forall j \quad (27)
\]

The total cost of all \( p \)-cycles is minimized by (20). Constraint (21) associates each span \( (u,v) \in E \) to at most one on-cycle vector in each \( CS_j \), either \( u \rightarrow v \), \( v \rightarrow u \), or none. Constraint (22) defines cycle sets \( CS \) using on-cycle vectors. Each node in the network has either 2 or 0 on-cycle vectors incident on it (regardless of the direction of the on-cycle vectors). Constraint (23) ensures 100% span protection. Constraint (24) allows at most one root node in each \( CS_j \). Constraint (25) says that, only the root node can serve as a common head for two on-cycle vectors, and any other node in the network can be a head for at most one on-cycle vector. This ensures a single root-reversal node pair in \( CS \). For each on-cycle vector, (26) specifies that its tail must have a larger voltage value than its head. Constraint (27) indicates that, a span can be protected by the only cycle in \( CS \) if its two end nodes are on \( CS \).

From (26), we have \( p_i^j - p_i^j \geq -1 \) and \( p_i^j - p_i^j \geq 1 \) if \( (u,v) \) is not an on-cycle span of \( CS_j \), or \( p_i^j - p_i^j \leq 1 \). Note that the network contains \(|V|\) nodes and their voltage values may be arranged into a monotonically increasing order with a step of (at least) \( \alpha \) as formulated in (26). To ensure that each node has a proper voltage value, we set

\[
    \frac{1}{|V|} \geq \alpha > 0 \quad (28)
\]

Note that \( \alpha \) is a predefined constant and as long as (28) is satisfied, its value is also not important. On the other hand, voltage values can also be defined as integers if constraint (26) is replaced by (29) below, where \( \beta \) is an arbitrary constant not smaller than \(|V|\).

\[
    p_i^j - p_i^j \geq \theta_{uv}^j - \beta (1 - \theta_{uv}^j),
    \forall (u,v), (v,u) \in E, \forall j \quad (29)
\]

This turns the model into a true ILP.

Compared to the recursion and flow conservation based ILPs, which require either “node level” or “path level” details, the cycle exclusion-based ILP only involves “cycle level” details. Since this ILP ensures a single cycle per \( CS_j \), it is very efficient in checking whether a span can be protected by a \( CS \). In contrast, both the recursion and flow conservation based ILPs need a dedicated set of variables and constraints for checking each span. The cycle exclusion-based ILP formulated in (20)-(27) involves \( 3|E||V| \) variables and \( 4E|V| + 2|V| + |E| + J \) constraints, which only increase linearly with network size if \( J \) is given.

D. Cycle Exclusion-Based ILP for JCP

In the joint capacity placement (JCP) problem, traffic demands are expressed as the bandwidth requirements for all source-destination node pairs (i.e., traffic matrix \( D \)). Since working paths are jointly optimized with spare capacity placement, working capacity \( l_{uv} \) on each span \( (u,v) \) becomes a design parameter. For simplicity, we focus on an undirected network, and only consider the upper triangle of a symmetric traffic matrix \( D \). To facilitate our formulation on the routing process, we first treat each demand \( d_{uv} \) as a directed demand from source \( s \) to destination \( t \). Then, the required working capacity \( l_{uv} \) on each span \( (u,v) \) is obtained by adding up the working capacity on \( (u,v) \) in both directions.

Our ILP for solving the JCP problem is based on the cycle exclusion approach formulated in (20)-(27), due to its simplicity as compared to both recursion and flow conservation based approaches. In particular, objective in (20) is replaced by (30) below.

\[
    \min \left\{ \sum_{j} \sum_{(u,v) \in E} c_{uv} (\theta_{uv}^j + \theta_{vu}^j) + \sum_{d_{uv} \in D} \sum_{d_{uv} \in D} c_{uv} (d_{uv}^w + d_{uv}^m) \right\} \quad (30)
\]

In addition to constraints (21)-(27), (31)-(34) are required for routing optimization.

\[
    l_{uv} = \sum_{d_{uv} \in D} (d_{uv}^w + d_{uv}^m),
    \forall (u,v) \in E \quad (31)
\]

\[
    \sum_{(u,v) \in E} d_{uv}^w = d_{uv},
    \forall (u,v) \in E \quad (32)
\]

\[
    \sum_{(u,v) \in E} d_{uv}^w = d_{uv},
    \forall (u,v) \in E \quad (33)
\]

\[
    \forall d_{uv} \in D, \forall k \in V: k \neq s, k \neq t \quad (34)
\]

Constraint (31) calculates \( l_{uv} \) by adding up the working capacity on \( (u,v) \) for all demands \( d_{uv} \). For any demand \( d_{uv} \), (32)-(33) specify that the number of traffic units generated at source \( s \) and arrived at destination \( t \) must equal to \( d_{uv} \). Constraint (34) ensures flow conservation for each demand \( d_{uv} \). Except source \( s \) and destination \( t \), all other nodes in the network must have equal number of inbound and outbound flow/traffic units for each demand \( d_{uv} \).

In addition to the \( 3|E||V| \) variables and \( 4|E| + 2|V| + |E| + J \) constraints in (21)-(27), (30)-(34) introduce additional \( 2|D| \times |E| \) variables and \( |E| + |D| \times |V| \) constraints, where \( |D| \) is the total number of non-zero demands in the traffic matrix \( D \).

E. Discussion

Our ILPs can also be easily extended to \( p \)-cycle design with a hop-count or circumference limit \( L \). In either recursion or flow conservation based ILP, this can be easily achieved with the additional constraint (35).

\[
    \sum_{(u,v) \in E} c_{uv} e^j_{uv} \leq L \quad (35)
\]

For the two cycle exclusion based ILPs (for SCP and JCP, respectively), the additional constraint (36) is needed instead.

\[
    \sum_{(u,v) \in E} c_{uv} (\theta_{uv}^j + \theta_{vu}^j) \leq L \quad (36)
\]
The number of variables and constraints in different ILPs.

<table>
<thead>
<tr>
<th>ILPs</th>
<th>Number of variables</th>
<th>Number of constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional ILP with cycle enumeration [1]</td>
<td>$</td>
<td>C</td>
</tr>
<tr>
<td>Schupke’s ILP [22]</td>
<td>$J(</td>
<td>V</td>
</tr>
<tr>
<td>Recursion-based ILP for SCP (1)-(10)</td>
<td>$≈J(2</td>
<td>E</td>
</tr>
<tr>
<td>Cycle exclusion-based ILP for SCP (20)-(27)</td>
<td>$3J</td>
<td>E</td>
</tr>
<tr>
<td>Cycle exclusion-based ILP for JCP (30), (21)-(27) and (31)-(34)</td>
<td>$3J</td>
<td>E</td>
</tr>
</tbody>
</table>

Note that in the four ILPs we formulated, $J$ (the maximum number of cycle sets allowed in the solution) should be predetermined. In general, $J$ should be set sufficiently large whereas the final solution may contain less cycle sets. However, a large $J$ generally slows down the optimization process, because for a given network the number of variables and constraints increases linearly with $J$. Define a segment as a path consisting of at least two spans and with a degree of 2 at any intermediate node. Let $S$ be the set of spans on all segments and $\delta$ be a small positive integer. For solving the SCP $p$-cycle design problem, we can set $J$ according to (37).

$$J = \max \left\{ \frac{L}{2}; (u, v) \in E - S, \ l_{uv}; (u, v) \in S \right\} + \delta \quad (37)$$

This is because $p$-cycles tend to straddle the most-heavily-loaded span so as to protect two units of traffic on it. On the other hand, $p$-cycles cannot straddle a span on a segment, and thus only one unit of traffic on such a span can be protected by a $p$-cycle traversing it. For JCP $p$-cycle design, (37) cannot be applied because $l_{uv}$ on each span $(u, v)$ is yet to be determined. As a substitute, we can first try shortest path routing based on the given traffic matrix to calculate an estimated $l_{uv}$ for each span $(u, v)$, and then use (37) to calculate the required value of $J$.

For heavily loaded networks, the value of $J$ in (37) could be quite large. We can scale down the traffic by combining multiple wavelengths into a single coarser bandwidth unit. This allows us to scale down the value of $J$. On the other hand, we can also follow a divide-and-conquer approach to partition a large network into smaller networks, and then use our ILPs to design $p$-cycles for each. These techniques may slightly decrease the quality of the solution, but will dramatically cut down the ILP running time.

TABLE I compares the number of variables and constraints involved in different ILPs. In the conventional ILP with candidate cycle enumeration [1], the number of variables equals to the total number of cycles in the network, which increases exponentially with network size. In Schupke’s ILP [22] and our recursion and flow conservation based ILPs, both the number of variables and constraints are quadratic functions of the number of nodes $|V|$ or spans $|E|$ in the network. Notably, those numbers only increase linearly with $|V|$ and $|E|$ in our cycle exclusion-based ILP for SCP (assume $J$ is given).

IV. NUMERICAL RESULTS

We use CPLEX 10.0 [30] to solve the ILPs on a standard
Pentium IV 2.2 GHz computer, with CPLEX parameters below to speedup the optimization.

- 1→ emphasis mip
- 2→ mip strategy probe
- 3→ mip strategy rings
- 3→ mip strategy heuristicfreq (38)
- 2→ mip cuts all
- 3→ mip strategy dive
- 3→ preprocessing symmetry

Unless otherwise specified, solutions obtained are SCP solutions without cycle length limit. In all examples, we set $\alpha=0.01$ for the cycle exclusion-based ILPs. For brevity, in the figures we denote the four ILPs formulated in this paper by RC (recursion-based ILP for SCP), FC (flow conservation-based ILP for SCP), CE (cycle exclusion-based ILP for SCP) and JCP (cycle exclusion-based ILP for JCP), respectively.

The three homogeneous networks [23] in Fig. 9 are first considered, with the same traffic load $l_{uv}=1$ and span cost $c_{uv}=1$ at every span $(u, v) \in E$. The homogeneous scenario corresponds to $p$-cycle protection for dynamic traffic [10], or fiber-level protection in DWDM networks [23]. Fig. 9 shows the optimal solutions returned by our recursion, flow conservation and cycle exclusion based ILPs. The results from Schupe’s ILP [22] are not shown because the ILP running time is too long (e.g., 428.25 seconds for Fig. 9a with $J=3$, and 16851.23 seconds for Fig. 9b with $J=2$). For homogeneous networks, the required number of $p$-cycles is generally not very large. As a result, all of our three ILPs for SCP can generate optimal solutions in a very short amount of time.

Fig. 10 gives two examples of capacitated networks [23], with span cost $c_{uv}=1$ at each span $(u, v) \in E$. The number next to each span in the topologies gives the number of traffic units $l_{uv}$ on that span (same for other examples). Since the ILPs need relatively long time to prove the optimality of the solution, we only take the solutions within a 5% gap to optimality (by setting 0.05—mip tolerance mipgap in CPLEX). Note that the total cost 85 in Fig. 10a is actually the optimal result according to [23]. Though all of our three ILPs for SCP generate a solution in a reasonable amount of time, we can see that the time required by the cycle exclusion-based ILP is the smallest.

A case study based on the pan-European COST 239 network (with 11 nodes and 26 spans) is shown in Fig. 11, with distance-related span costs defined in Fig. 11a. The traffic matrix in Fig. 11b is obtained by dividing the traffic matrix in [31] by 10Gbps. The demands are then routed according to shortest path routing, and Fig. 11c shows the traffic load on each span after routing. Based on Fig. 11c, the SCP solutions obtained from our recursion, flow conservation and cycle exclusion based ILPs with $J=7$ are summarized in Fig. 11d. We can see that the cycle exclusion-based ILP dramatically cuts down the running time when compared to the other two. In particular, Fig. 11e lists the set of $p$-cycles generated by the cycle exclusion-based ILP, which is obtained in only 34.68 seconds. Among the 7 $p$-cycles in Fig. 11e, the one marked with an asterisk has the largest circumference of 5940 km. Fig. 11f gives another SCP solution with a $p$-cycle circumference limit of 5800 km. It is obtained by including constraint (36) in the cycle exclusion-based ILP.

### Table 1

<table>
<thead>
<tr>
<th>$J=7$</th>
<th>RC</th>
<th>FC</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in seconds</td>
<td>844.63</td>
<td>4293.03</td>
<td>334.59</td>
</tr>
<tr>
<td>Total cost</td>
<td>85</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>Gap to optimality</td>
<td>4.31%</td>
<td>4.31%</td>
<td>4.90%</td>
</tr>
</tbody>
</table>

### Table 2

<table>
<thead>
<tr>
<th>$J=7$</th>
<th>RC</th>
<th>FC</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time in seconds</td>
<td>341.21</td>
<td>139.86</td>
<td>4.29</td>
</tr>
<tr>
<td>Total cost</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Gap to optimality</td>
<td>4.17%</td>
<td>4.17%</td>
<td>4.17%</td>
</tr>
</tbody>
</table>

(a) A capacitated network taken from [23] ($|V|=13, |E|=23$).

(b) SmallNet with $|V|=10$ and $|E|=22$.

Fig. 10. SCP solutions for two capacitated networks with $c_{uv}=1$ at each span $(u, v) \in E$. 

### Additional Figures

- Fig. 9: Homogeneous networks with different $J$ values.
- Fig. 11: Case study on the pan-European COST 239 network.

### Additional Equations

- $J=7$ cycle exclusion-based ILP solution.
- $p$-cycle protection for dynamic traffic.
JCP is also considered in Fig. 11 with the same traffic matrix in Fig. 11b. The solution obtained (without p-cycle length limit) contains not only a set of p-cycles as listed in Fig. 11g, but also a routing table as shown in Fig. 11h. In Fig. 11g, we can see that there are only 4 p-cycles in the solution. This explains why the JCP solution is obtained faster (only 18.26 seconds) than its SCP counterpart (34.68 seconds with 7 p-cycles). In Fig. 11h, we can observe the differences between JCP routing and
shortest path routing adopted in the SCP scenario. Specifically, the demands in the dashed rectangles are assigned with two working paths, whereas in shortest path routing we have required each demand to follow a single working path. Besides,
in Fig. 11h many demands do not follow the shortest path. For example, the working path for demand \( d_{06} \) is \( 0 \rightarrow 6 \) with a length of 740 km, but the shortest path between nodes 0 and 6 is \( 0 \rightarrow 3 \rightarrow 6 \) with a total length of 730 km. We also calculate the traffic load on each span based on the routing table in Fig. 11h, and the result is shown in Fig. 11i. Compared with Fig. 11c, we can see that the traffic loads on the spans are better balanced in Fig. 11i. This explains why the JCP solution only requires 4 \( p \)-cycles (Fig. 11g), in contrast to 7 \( p \)-cycles in the SCP solution (Fig. 11e). Generally, JCP routing can achieve a better load balancing than shortest path routing. As a result, less number of \( p \)-cycles is required in the JCP solution. Recall that in JCP, we have proposed to try shortest path routing to estimate \( l_w \) before calculating the required value of \( J \) using (37). Our above analysis validates this approach because the required value of \( J \) in JCP is generally not larger than that in SCP. Based on Figs. 11a, 11c and 11e, it is easy to calculate that the total capacity required in the SCP solution (without \( p \)-cycle length limit) is 93760 (spare capacity 32760 plus working capacity 61000). We can see that the JCP solution with a total capacity of 83245 in Fig. 11g achieves a capacity saving of 11.21% over the SCP solution.

In the pan-European COST 239 network, if we use 2.5Gbps instead of 10Gbps to divide the traffic matrix in [31] and then route the demands according to shortest path routing, we can get the traffic load on each span as shown in Fig. 12a. Compared to Fig. 11c, the number of traffic units on each span is larger due to the finer wavelength granularity. So, we set \( J \) to a larger value of 15 according to (37). Fig. 12b compares the running time of the recursion, flow conservation and cycle exclusion based ILPs, and Fig. 12c shows the \( p \)-cycles generated by the cycle exclusion-based ILP. Again, we can see that the cycle exclusion-based ILP runs much faster than the other two. Note that the recursion-based ILP is terminated after 5 hours (by setting 18000→timelim in CPLEX). Comparing Fig. 12b to Fig. 11d, we can see that the running time increases with \( J \) for each ILP. To check the efficiency of our cycle exclusion-based ILP for large \( J \) values, we consider the network in Fig. 13 with \( J=50 \). We can see that a good SCP solution can still be returned in a reasonable running time.

We also compare the cycle exclusion-based ILP to the conventional ILP [1] with candidate cycle enumeration. Cycles are enumerated using the algorithm in [27]. For small size networks, cycles can be enumerated in a short time, and the size of the candidate set is not too large. For example, the pan-European COST 239 network in Fig. 11c contains 3531 distinct cycles, which can be found in less than 2 seconds. As network size increases, cycle enumeration needs a long time and the size of the candidate set soars exponentially. For the randomly generated network in Fig. 14 (with 30 nodes and 62 spans), we find 13343782 cycles in 16420 seconds (4.56 hours). Though cycle enumeration is still manageable, it is infeasible for the conventional ILP [1] to handle so many cycles/variables. In contrast, our cycle exclusion-based ILP (with \( J=7 \) and \( \alpha=0.01 \)) can return an SCP solution with a gap to optimality of 4.83% in 31520.35 seconds (8.76 hours).

It should be noted that we do not compare our ILPs to the conventional ILP [1] with candidate cycle pre-selection [18-21], or other heuristic algorithms (e.g. CIDA [19]), due to multiple reasons as follows. First, we focus on an optimal ILP model, whereas those approaches are heuristic-based; Second, the solution quality of those approaches is not guaranteed, whereas our ILPs always ensure a solution with an explicit gap to the true optimality. In fact, the examples in [19, 21] show that the solution quality obtained from those approaches is generally 10%-20% worse than the optimal solution, or a close-to-optimal solution can be achieved if 20%-40% of all cycles are pre-selected (obviously, this still introduces a huge number of ILP variables for a large-size network as the one in Fig. 14); Third, the above performance evaluation [19, 21] is obtained based on some small-size networks. As network size increases, the solution quality of the heuristic-based approaches intends to be even worse. For example, Grow algorithm [19] pre-selects \( O(|E|^2 \times |V|) \) cycles, but the total number of cycles in the network increases exponentially with network size (i.e., grows faster than \( O(|E|^2 \times |V|) \)). As a result, when network size increases, the ratio of the number of pre-selected cycles to the total number of all cycles decreases rapidly, leading to a poorer solution quality (than 10%-20% gap-to-optimality); Finally, even with Grow algorithm [19] for cycle pre-selection, for the network in Fig. 14, the number of pre-selected cycles still reach a magnitude of \( |E|^2 \times |V|=62^2 \times 30=115320 \). This is only 0.86% of all 13343782 cycles, but still introduces 115320 variables to ILP. In contrast, our cycle exclusion-based ILP for SCP (with \( J=7 \)) only involves \( 3(|E|+|V|)=1932 \) variables and \( 4|E|+2|V|+|E|+J=2225 \) constraints. Though the number of constraints in our ILP is larger than \(|E|=62\) in the conventional ILP with cycle pre-selection, the overall problem size of our ILP is still much smaller.

V. CONCLUSION

We focused on \( p \)-cycle design without candidate cycle enumeration and pre-selection. Three ILPs (Integer Linear Programs) were first formulated for solving the spare capacity placement (SCP) problem, based on recursion, flow conservation and cycle exclusion, respectively. We showed that the cycle exclusion-based ILP is the most efficient one. The number of ILP variables and constraints involved in the cycle exclusion-based ILP only increases linearly with network size. This is a great advantage compared to the conventional ILP with candidate cycle enumeration, where the number of ILP variables increases exponentially with network size. We also formulated another ILP for solving the joint capacity placement (JCP) problem by extending our cycle exclusion approach. Numerical results showed that our ILPs are very efficient for \( p \)-cycle design in various networks.

Note that we only focus on simple \( p \)-cycle design, and non-simple \( p \)-cycle [28, 32] is out of the scope of this paper. A simple \( p \)-cycle traverses any node or span at most once, whereas a non-simple \( p \)-cycle can traverse a node or a span multiple times. Compared with simple \( p \)-cycles, non-simple
$p$-cycles have some new features which lead to a different design methodology. An ILP formulation for non-simple $p$-cycle design without candidate cycle enumeration can be found in [28].

REFERENCES


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. He is currently 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.