

Figure 1: An example to illustrate max-flow min-cut theorem.

图例

## 0.1 Runtime

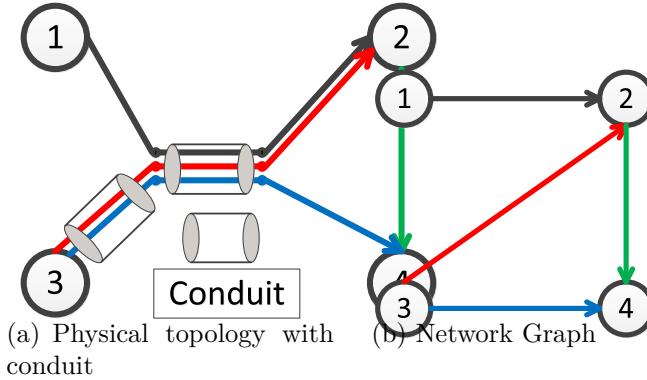


Figure 2: Example of shared risk link group(SRLG)

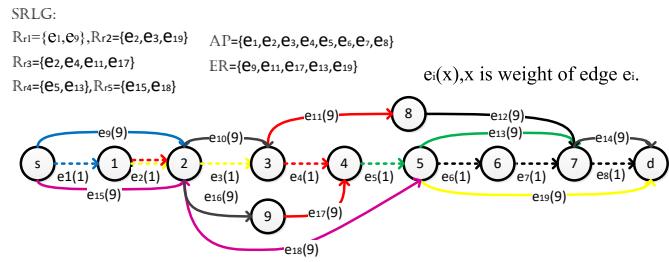


Figure 3:  $\mathbb{AP} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ ,  $\mathbb{ER} = \{e_9, e_{11}, e_{17}, e_{13}, e_{19}\}$ . SRLGs:  
 $\mathbb{R}_{r1} = \{e_1, e_9\}$ ,  $\mathbb{R}_{r2} = \{e_2, e_3, e_{19}\}$ ,  $\mathbb{R}_{r3} = \{e_2, e_4, e_{11}, e_{17}\}$ ,  $\mathbb{R}_{r4} = \{e_5, e_{13}\}$ ,  $\mathbb{R}_{r5} = \{e_{15}, e_{18}\}$

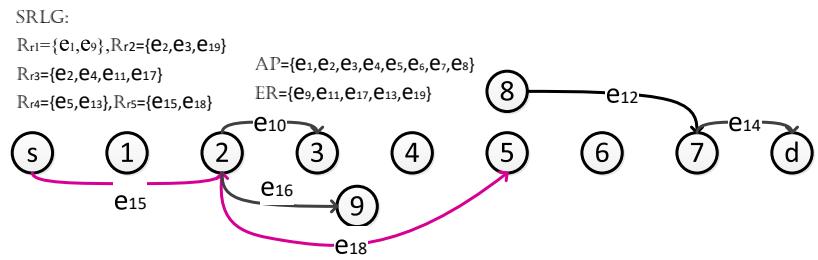


Figure 4: Disconnected graph after deleting the links in AP and ER.

$$\mathcal{P}(\emptyset, \emptyset) \left\{ \begin{array}{l} \mathcal{P}(\{e_2\}, \emptyset) \\ \boxed{\mathcal{P}(\emptyset, \{e_2\})} \end{array} \right\} \left\{ \begin{array}{l} \mathcal{P}(\{e_2, e_5\}, \emptyset) \\ \boxed{\mathcal{P}(\{e_2\}, \{e_5\})} \end{array} \right\} \left\{ \begin{array}{l} \boxed{\mathcal{P}(\{e_2, e_5, e_6\}, \emptyset)} \\ \boxed{\mathcal{P}(\{e_2, e_5\}, \{e_6\})} \end{array} \right\}$$

Figure 5: Example to illustrate divide-and-conquer solution

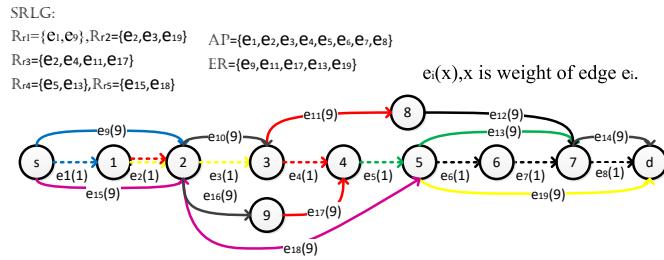


Figure 6: New Graph  $G^*$

Sample	1	2	3	4	5	6
Node	424	1825	514	514	8301	1814
Edge	1378	24837	2024	2074	40326	24816
Sample	7	8	9	10	11	12
Node	2014	2014	2014	2014	2013	20
Edge	2068	2068	2068	2068	15661	31
Sample	13	14	15	16	17	
Node	2014	2014	424	2014	2014	
Edge	2068	2068	1378	2068	2068	

Table 1: node number and edge number of all samples.

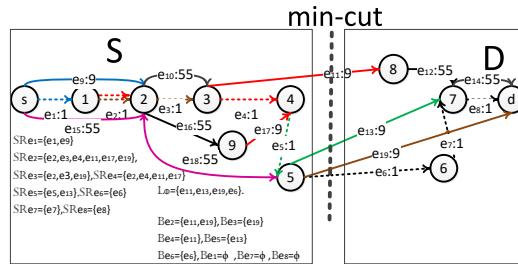


Figure 7: Min cut of Graph  $G^*$ ,  $\mathbb{SR}_{e_1} = \{e_1, e_9\}$ ,  $\mathbb{SR}_{e_2} = \{e_2, e_3, e_4, e_{11}, e_{17}, e_{19}\}$ ,  $\mathbb{SR}_{e_3} = \{e_2, e_3, e_{19}\}$ ,  $\mathbb{SR}_{e_4} = \{e_2, e_4, e_{11}, e_{17}\}$ ,  $\mathbb{SR}_{e_5} = \{e_5, e_{13}\}$ ,  $\mathbb{SR}_{e_6} = \{e_6\}$ ,  $\mathbb{SR}_{e_7} = \{e_7\}$  and  $\mathbb{SR}_{e_8} = \{e_8\}$ .  $\mathbb{L}_\Phi = \{e_{11}, e_{13}, e_{19}, e_6\}$ .  $\mathbb{B}_{e_1} = \emptyset$ ,  $\mathbb{B}_{e_2} = \{e_{11}, e_{19}\}$ ,  $\mathbb{B}_{e_3} = \{e_{19}\}$ ,  $\mathbb{B}_{e_4} = \{e_{11}\}$ ,  $\mathbb{B}_{e_5} = \{e_{13}\}$ ,  $\mathbb{B}_{e_6} = \{e_6\}$ ,  $\mathbb{B}_{e_7} = \emptyset$  and  $\mathbb{B}_{e_8} = \emptyset$ . **Dear sister, SR is wrong in this example, I will let the student to modify tomorrow**

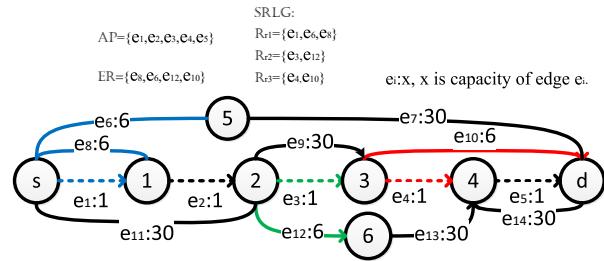


Figure 8: Sample Graph  $G_2^*$

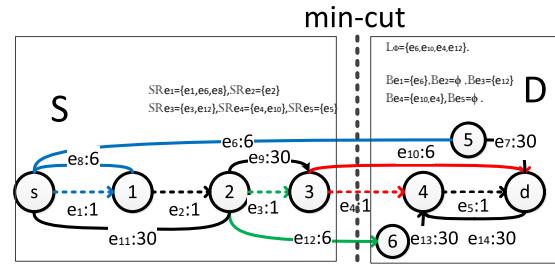


Figure 9: Min cut of Sample Graph  $G_2^*$ ,  $\mathbb{SR}_{e_1} = \{e_1, e_6, e_8\}$ ,  $\mathbb{SR}_{e_2} = \{\}$ ,  $\mathbb{SR}_{e_3} = \{e_3, e_{12}\}$ ,  $\mathbb{SR}_{e_4} = \{e_4, e_{10}\}$ ,  $\mathbb{SR}_{e_5} = \{e_5\}$ .  $\mathbb{L}_\Phi = \{e_6, e_{10}, e_4, e_{12}\}$ .  $\mathbb{B}_{e_1} = \{e_6\}$ ,  $\mathbb{B}_{e_2} = \emptyset$ ,  $\mathbb{B}_{e_3} = \{e_{12}\}$ ,  $\mathbb{B}_{e_4} = \{e_{10}, e_4\}$ ,  $\mathbb{B}_{e_5} = \emptyset$ .

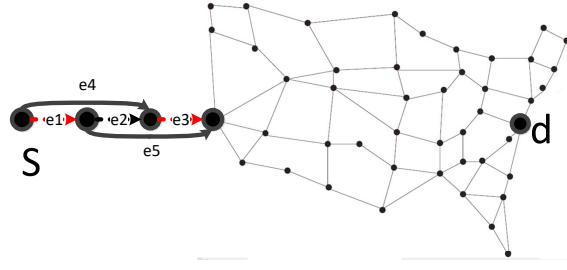


Figure 10: In this graph, suppose SRLG conflicting set is  $e_1, e_3$ , When KSP algorithm find the  $AP = (e_1, e_2, e_3, \dots)$  iteratively, but the problematic edges  $e_1, e_3$  always exist the k-th shortest path, then KSP could find the AP which is SRLG-disjoint with BP in high time consumption. However, my algorithm find problematic edges  $e_1, e_3$  and find path AP in two subproblem  $\mathcal{P}(\emptyset, \{e_1\})$  and  $\mathcal{P}(\{e_1\}, \{e_3\})$ .  $AP$  of the subproblem  $\mathcal{P}(\emptyset, \{e_1\})$  pass edges  $e_4$  and  $e_3$  to destination  $d$  with SRLG-disjoint path BP which pass edges  $e_1$  and  $e_3$  to destination  $d$ . In like manner, the subproblem  $\mathcal{P}(\{e_1\}, \{e_3\})$  could find disjoint paths pair.

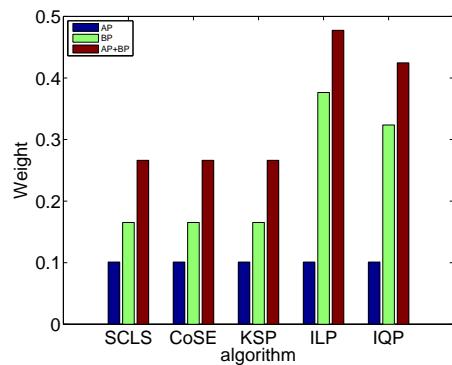


Figure 11: Normalization Weitgh Value

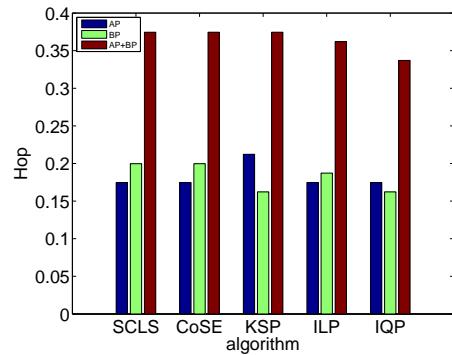


Figure 12: Normalization Hop

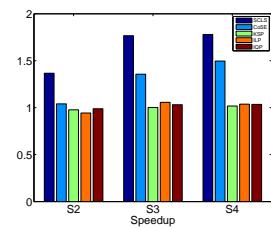


Figure 13: Speedup

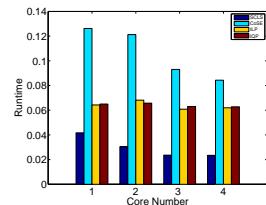


Figure 14: Runtime without KSP algorithm

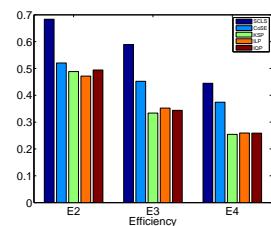


Figure 15: Efficiency

---

**Algorithm 1** Min-Min

---

**Require:**  $G$ : the network graph

$s$ : the source node

$d$ : the destination node

$\mathbb{I}$ : the inclusion link set should be included in AP

$\mathbb{O}$ : the exclusion link set should not be included in AP

**Ensure:**  $AP$ : the active path

$BP$ : the backup path

1:  $AP = \emptyset, BP = \emptyset$

2:  $AP \leftarrow \text{FIND\_AP}(G, s, d, \mathbb{I}, \mathbb{O})$

3: **if**  $AP \neq \emptyset$  **then**

4:    $BP \leftarrow \text{FIND\_SRLG\_Disjoint\_BP}(G, s, t, AP)$

5:   **if**  $BP \neq \emptyset$  **then**

6:     return path pair  $(AP, BP)$

7:   **else**

8:     find SRLG conflict link set  $\mathbb{T}$

9:      $\mathbb{T} \leftarrow \mathbb{T} - (\mathbb{I} \cup \mathbb{O})$

10:   divide and conquer for execution in parallel

$$\left\{ \begin{array}{l} (AP_1, BP_1) = \text{Min} - \text{Min}(G, s, d, \mathbb{I}, \mathbb{O} \cup \{t_1\}), \\ (AP_2, BP_2) = \text{Min} - \text{Min}(G, s, d, \mathbb{I} \cup \{t_1\}, \mathbb{O} \cup \{t_2\}), \\ (AP_3, BP_3) = \text{Min} - \text{Min}(G, s, d, \mathbb{I} \cup \{t_1, t_2\}, \mathbb{O} \cup \{t_3\}), \\ \dots \\ (AP_{|\mathbb{T}|}, BP_{|\mathbb{T}|}) = \text{Min} - \text{Min}(G, s, d, \mathbb{I} \cup \{t_1, t_2, \dots, t_{|\mathbb{T}|-1}\}, \mathbb{O} \cup \{t_{|\mathbb{T}|}\}) \end{array} \right.$$

11:  $F \leftarrow \text{FIND\_Feasible}((AP_1, BP_1), \dots, (AP_{|\mathbb{T}|}, BP_{|\mathbb{T}|}))$

12:   **if**  $F \neq \emptyset, \emptyset$  **then**

13:     return path pair  $(AP, BP)$  satisfying that  $AP = \arg \min_{AP} \{F\}$

14:   **end if**

15:   **end if**

16: **end if**

---

## 一种求完全风险共享链路组分离路径对的算法

### 技术领域

本发明是涉及网络生存性领域，特别是涉及光网络环境下光纤物理信道出现物理损坏时，具体是一种求完全风险共享链路组分离路径对的算法。

### 背景技术

随着多媒体流和视讯会议等新应用的出现要求网络提供可靠的服务质量 (QoS) 保证，不仅要满足应用的 QoS 要求，还要在网络故障时能够持续保证业务不间断地进行。要达到这些要求，通常为 1 个连接提供两条链路/节点分离的路径，其中 1 条主用，1 条备用。当主用路径故障时，将其承载的业务流倒换到备用路径上，从而实现快速的业务恢复。此外，负载均衡也需要分离路径实现网络中业务流的均匀分布，避免网络拥塞，优化网络吞吐量。健壮性 (Robustness) 和负载均衡是可靠 QoS 路由的 2 个重要方面。光网络和 MPLS /GMPLS 技术的发展提供资源预约和显式路由能力，使得在网络中提供可靠的 QoS 保证成为可能。如何在节点之间建立链路/节点分离路径成为提供可靠 QoS 的主要问题。

光网络设备技术的进一步发展和成熟，具有更高性能的光分插复用器 (OADM, Optical Add / Drop Multiplexer) 和光交叉连接器 (OXC, Optical Cross Connector) 等正在不断涌现，如何构造新的光传送网络 (OTN, Optical Transport Network) 和利用这些设备设计，从网络设计、控制和管理的角度，研究和开发新的协议，以保证 OTN 具有更高的可扩展性 (Scalability)、生存性 (survivability) 和灵活性 (flexibility)，正日益受到人们的广泛关注。其中的一个重要问题是如何在 OTN 中自动拆除和建立光通路。目前，已有多家企业联盟和标准化组织都在制定相应的标准，IETF 正在研究采用 MPLS 的控制面协议实现光网络控制，从 1999 年底至今，有关工作才在 IETF 的众多草案文本中体现了出来，并被统一地称为 GMPLS 协议族。

所有这些工作目的都是规范光网络的控制层协议，但对其中涉及到的有关算法细节没有作细致的规定。例如光路的保护问题。由于一条光路可能聚合了大量的用户业务，因此光路的失效造成的损失难以承受，公认的观点是必须在光层提供合适的保护和恢复机制。

一种有效的提供光路保护的机制是为每个光路请求建立两条路径，分别称为工作路径 AP 和保护路径 BP，一旦工作路径失效，可以立刻将业务切换到保护路径上运行。显然，要保证这一机制有效运行，计算出的两条路径必须是“物理分离”的。物理分离根据防止的失效程度有三种含义，即节点分离、链路分离和范围分离。所谓共享风险链路组 (Shared Risk Link Groups, SRLG) 的概念是对“物理分

离”概念的进一步扩展和抽象，其定义为一组链路共享同一物理资源，比如具有相同的失效风险、经过同一灾区。网络操作者通过指定物理链路的 SRLG 来满足不同的要求。例如，所有穿过同一光缆的光纤属于同一 SRLG；类似地，所有穿过同一光纤的波长通路属于同一 SRLG。甚至，网络操作者为了能在地震、洪水等非常情况下也能确保服务，可以指定穿过同一灾区的物理链路具有同一 SRLG 标识。同时每条链路可以同时属于多个 SRLG。SRLG 分离的两条路径可以减少同时失效的可能性，提高了光路的抗毁机制。因此，提供通路保护的路由计算问题可以描述为：给定网络拓扑以及业务的源宿节点，要求找到两条以源节点为起点，以宿节点为终点的路径，且这两条路径是 SRLG 分离的路径上的所有链路与保护路径上的所有链路都不共享风险)。

传统的计算两条物理分离(注意，不是 SRLG 分离)路径的方法有两种。第一种可以称作“裁剪相继最短路”算法。这是最直观，也是最通常使用的方法：先计算一条最短路作为工作路径，然后在网络拓扑图中删除所有属于工作路径上的链路，最后在裁剪后的拓扑上计算一条最短路作为保护路径。另一种可以称为“变换相继最短路”算法，这是由 Suurballe 提出的，其基本思想仍然是两次调用最短路算法，不过在两次调用之间不是对图进行链路裁剪，而是对图进行权值变换。

下面分析一下直接采用这两种方法进行 SRLG 分离路径对的计算的情况。

先来考察“裁剪相继最短路”算法。将该算法扩展到计算 SRLG 分离路径对是容易的，只需在进行裁剪时不仅删除工作路径的所有链路，而且删除所有与这些链路具有相同 SRLG 属性的其它链路就可以了。但是，该算法有两个缺点：首先它是不完备的，即可能存在次优路径对，但采用该算法无法得到可能存在的次优路径对。其次，该算法计算出的两条路径只是保证工作路径最优，两条路径的指标之和未必最佳，这在很多场合下是不合理的。例如，光网络中最佳路由的常用计算指标之一是路径跳数最小，其物理含义是保证该业务占用的资源最少，因为增加一跳就多占了一跳链路上的资源。由于光网络中进行 1+1 独占式通路保护时，工作路径和保护路径上的资源都被占用了，不能再被其它业务占用，因此更合理的最佳路由指标应该是工作／保护路径的跳数之和最小。但采用“裁剪相继最短路”算法计算时，只能保证工作路径跳数最小，工作和保护路径跳数之和可能很大。

“变换相继最短路”算法可以计算出两条路径，且两条路径的指标之和最佳；并且该算法是完备的。但是遗憾的是，“变换相继最短路”算法只适用于链路分离／节点分离的情况(原算法只适用于链路分离，但只需在进行变换时稍作修正就可适用于节点分离情况)，不适用于更抽象、更完整的 SRLG 分离要求。实际上，正是由于该算法是变换相继最短路，若指标和最佳的两条路中有具有相同 SRLG 属性的链路，则该算法无法完备求解，因为此时可能存在两条次优路满足要求。

综上所述，现有的算法尚无法完备解决 SRLG 分离路径对的计算问题。本文首次提出了解决 SRLG 分离且低费用的路径对的查找问题的。这不仅有利于降低算法的复杂度，而且由于缩小了搜索空间，使得算法的效率可以满足实际的工程要

求。

## 1 发明内容

本发明要解决的技术问题是，针对现有技术存在的缺陷，提出一种求风险共享链路组完全分离路径对的算法。当在实际网络拓扑中求一对风险分离的路径对时，当在求得第一条路径的基础上求不到第二条风险分离路径，此时通过第一条路径的已知路径信息。并以此为基础，给出了创新性的构图方法来获得风险链路组冲突链路集，并以风险链路组冲突链路集提出分而治之的并行算法。本发明提出的求风险共享链路组完全分离路径对的算法，在现有的求点完全分离或者链路完全分离路径对的领域都可以适应，如软件定义网络中对虚拟网络路径的可生存性保护、光网络链路故障情况的路径保护、集中式路由器情景下对特定路径的路由保护等应用领域。本发明尤其适用于任何类型的风险链路组的情形，具有十分广泛的应用前景。

本发明的解决方案是：一种求风险共享链路组完全分离路径对的方法，该方法为：

1. 链路完全分离路径对算法的扩展：当一条链路属于一个或者多个风险共享链路组时，链路完全分离路径对算法无法解决非星型的风险共享链路组分离问题，当出现这种情况时，风险共享链路组完全分离路径对的算法可以通过风险链路组冲突链路集合来将原问题分而治之成各个子问题，并行计算每个子问题，最后通过子问题来获得原问题最优解。

2. 风险共享链路组完全分离路径对的算法步骤：

第一步：利用迪杰斯特拉算法在原图拓扑  $G$  中计算出第一条路径  $AP$ ；

第二步：在原图拓扑  $G$  中去除求得的第一条路径  $AP$  的边和其边共享风险的边的，得到删减图  $G^o$ ；

第三步：删减图  $G^o$  通过迪杰斯特拉算法找第二条路径  $BP$ 。如果找到  $BP$  返回 True；

第四步：如果没找到  $BP$ ，通过第一条路径  $AP$  在原图拓扑  $G$  中构造流量图  $G^*$ ，在流量图  $G^*$  上求风险链路组冲突链路集合。利用风险链路组冲突链路集合来将原问题分而治之成互不相容的子问题，并行执行每个子问题求得每个子问题的最优解，来求得原问题的最优解。

本发明还包括一种风险链路组冲突链路集合的方法，若当求得第一条路径  $AP$  而不存在分离的路径  $BP$  时，求以这条  $AP$  为基础的风险链路组冲突链路集合的流程：

1. 原图拓扑  $G$  上构建流量图  $G^*$ , 通过设置  $AP$  路径上的边, 与  $AP$  路径上的边共风险的边以及其他边的边容量, 来得到流量图  $G^*$ 。
2. 利用最大流最小割算法对流量图  $G^*$  求得源节点  $s$  与目标节点  $d$  的最小割集。
3. 最小规模  $AP$  路径上的边集合阻塞覆盖最小割集上的边, 得到风险链路组冲突链路集合。

以下对本发明做出进一步说明。

为了通过割集合来求得风险链路组冲突链路集合, 我们构造如下构造流量图  $G^*$ .

1.  $G^*$  跟原图  $G$  是有相同的点和边的拓扑关系。
2.  $G^*$  每条边的权值与原图  $G$  的相对应边的权值相等。
3. 我们设置如下规则来设定每条边的容量。

$$c_{e_i} = \begin{cases} 1 & e_i \in AP \\ |\text{AP}| + 1 & e_i \in ER \\ |\text{AP}| + (|\text{AP}| + 1) \times |\text{ER}| + 1 & \end{cases} \quad (1.1)$$

**引理 1.1.** 在流量图  $G^*$  中从  $s$  到  $d$  的任何路径必定经过至少一条在  $AP$  或者  $ER$  上的边。

**证明 1.2.** 我们通过反证法来证明这个引理。假设为路径  $AP$ , 在图  $G^*$  中存在另外一条从  $s$  到  $d$  的路径, 这条路径与  $AP$  不共享任何风险, 即这条路径不经过任何链路都不在  $AP$  或者  $ER$  上. 我们能显而易知这条路径是  $AP$  路径的风险共享链路组分离路径  $BP$ , 这与我们的前提矛盾, 我们假设  $AP$  路径是没有风险共享链路组分离路径  $BP$  的。

**引理 1.3.** 在流量图  $G^*$  中的最大流量最可能为  $|\text{AP}| + (|\text{AP}| + 1) \times |\text{ER}|$ 。

**证明 1.4.** 假设流量图  $G^*$  的最大流的值为  $|f| = k$ .  $f$  能在流量图  $G^*$  中被均分为  $k$  个从  $s$  到  $d$  的 1 单元的流。根据引理 1.1, 这些 1 单元的流必定经过一条边是在链路集  $AP$  或者  $ER$  中。然而  $AP$  或者  $ER$  边的容量为 1 或者  $|\text{AP}| + 1$ 。根据边  $AP$  和  $ER$  的容量设置原则, 在  $AP$  上的边最多承载 1 单元的流量, 同时在  $ER$  上的边最多承载  $|\text{AP}| + 1$  单元的流。因此, 将最大流最多只有  $|\text{AP}| + (|\text{AP}| + 1) \times |\text{ER}|$  单元的流量。

**引理 1.5.** 在流量图  $G^*$  中的最小割  $\Phi$  割边集  $L_\Phi$  的全部边属于  $AP$  或者  $ER$ 。

**证明 1.6.** 根据最小割最大流定理，最小割  $\Phi$  的容量为  $c(\Phi)$  应该等于最大流量值，而根据引理 1.3 最大流量值为  $|\text{AP}| + |\text{ER}| \times (|\text{AP}| + 1)$ 。根据等式 1.1 的容量设计原则，不是 AP 或者 ER 上的边的容量为  $|\text{AP}| + (|\text{AP}| + 1) \times |\text{ER}| + 1$

在一个具有风险共享链路组的网络中，如果一条边是 AP 的边或者是与 AP 共享风险的边，则这条边不会是 AP 路径风向共享链路组分离的路径 BP 上的边。我们称这些边被路径 AP 阻塞。

**定理 1.7.** 如果原图  $G$  的一个流路径阻塞了在最小割边集  $\mathbb{L}_\Phi$  的所有边，然后就没有流从  $s$  到  $d$  能通过这个图的割。

**证明 1.8.** 如果原图  $G$  的一个流路径阻塞了在最小割边集  $\mathbb{L}_\Phi$  的所有边，然后就没有流能使用割边集  $\mathbb{L}_\Phi$  的边，因此没有流能通过这个割集  $\Phi$  从  $s$  流到  $d$ 。

定理 1.7 告诉我们找到风险链路组冲突边集合的可能性。即当一条 AP 路径遇到一个 trap 问题时，我们能找到最小 AP 的子集来阻塞所有最小割边集  $\mathbb{L}_\Phi$  的所有边，这个最小子集边组成风险链路组冲突边集合。当我们任何一条路径包含风险链路组冲突边集合的所有边，则没有多余的流能通过这个割集  $\Phi$ ，因此不存在 SRLG 分离的路径 BP。

尽管所有在路径 AP 上的边形成一个 SRLG 冲突边集合，我们感兴趣的在于获得尽量小规模的集合，因为这个 SRLG 冲突边的大小决定子问题的数量。根据定理 1.7，求最小 SRLG 冲突边集合问题能被描述成找到在最小的规模 AP 的子集来覆盖所有的最小割集边  $\mathbb{L}_\Phi$ 。

对每条边  $e_i$ ，让  $\text{SRR}_{e_i}$  表示与边  $e_i$  共风险边的集合，很明显， $\text{SRR}_{e_i}$  包含  $e_i$  这条边本身和所有与它共风险的边。对每条在 AP 上的边  $e_i$ ，我们定义 cut-block-link 集合  $\mathbb{B}_{e_i} = \text{SRR}_{e_i} \cap \mathbb{L}_\Phi$ ，最小割集边  $\mathbb{L}_\Phi$  的边子集能被  $e_i$  阻塞。因此，求最小的风险链路组冲突边集合问题能被形式化为集合覆盖问题：给定 AP（路径 AP 的路径边的边集合），最小割边集  $\mathbb{L}_\Phi$  和 cut-block-link 集合族  $\mathbb{B}_{e_1}, \mathbb{B}_{e_2}, \dots, \mathbb{B}_{e_{|\text{AP}|}}$ ，我们想求一个  $\mathbb{B}_{e_i}$  最小规模其的集合族其并是  $\mathbb{L}_\Phi$ 。最小的  $\mathbb{T} \subseteq \{e_i, e_i \in \text{AP}\}$  使得  $\cup_{e_i \in \mathbb{T}} \mathbb{B}_{e_i} = \mathbb{L}_\Phi$ 。

集合覆盖问题是 NP-难问题并且它的计算复杂度取决于元素的规模 ( $n$ )。在我们求最小风险链路组冲突边集合的问题中， $n = \mathbb{L}_\Phi$ ，即最小割集边  $\mathbb{L}_\Phi$  的边的数目。我们应用贪婪算法求得最小风险链路组冲突边集合。根据 SRLG 的图形类型，有两种 SRLG 类型的：星型与非星型。不同于已经存在的其他研究方法，我们处理星型 SRLG 和非星型的 SRLG。

当我们获得最小风险链路组冲突边集合时，我们能设计一个分而治之的算法来拆分原 Min-Min SRLG-disjoint routing 问题成多个子问题，这些子问题能并行处理以至于加速整个 SRLG 分离路径对的求解过程。为了使这个问题更加好分离，我们首先定义两个互斥的边集合  $\mathbb{I}$  和  $\mathbb{O}$ ， $\mathbb{I}$  是被称作必经集合和  $\mathbb{O}$  是被称作分离集合，定义  $\mathcal{P}(\mathbb{I}, \mathbb{O})$  为 Min-Min SRLG-disjoint routing 问题的子问题，在这个

$\mathcal{P}(\mathbb{I}, \mathbb{O})$  问题里路径  $AP$  是所有必须过  $\mathbb{I}$  中所有边和必须不过  $\mathbb{O}$  中所有  $AP$  路径集里最短的那一条路径。

让  $\mathbb{I} = \emptyset$  和  $\mathbb{O} = \emptyset$  原 Min-Min SRLG-disjoint routing 问题能被表示为  $\mathcal{P}(\emptyset, \emptyset)$ 。给定风险链路组冲突边集合  $\mathbb{T}$ ,  $\mathbb{T}$  有  $|\mathbb{T}|$  条边  $e_1, e_2, \dots, e_{|\mathbb{T}|}$ , 这个原问题能被按照以下步骤分离成各个子问题。

Step 1,  $\mathcal{P}(\emptyset, \emptyset)$  能够被分离成两个子问题  $\mathcal{P}(\emptyset, \{e_1\})$  和  $\mathcal{P}(\{e_1\}, \emptyset)$ 。

Step 2, 同理,  $\mathcal{P}(\{e_1\}, \emptyset)$  能够继续被分离成两个子问题  $\mathcal{P}(\{e_1, e_2\}, \emptyset)$  和  $\mathcal{P}(\{e_1\}, \{e_2\})$ .

这个拆分过程能持续到 Step  $|\mathbb{T}|$ , 我们有问题  $\mathcal{P}(\{e_1, e_2, \dots, e_{|\mathbb{T}|-1}\}, \emptyset)$  能够被拆分成两个子问题  $\mathcal{P}(\{e_1, e_2, \dots, e_{|\mathbb{T}|-1}, e_{|\mathbb{T}|}\}, \emptyset)$  和  $\mathcal{P}(\{e_1, e_2, \dots, e_{|\mathbb{T}|-1}\}, e_{|\mathbb{T}|})$ . 即我们已知子问题  $\mathcal{P}(\{e_1, e_2, \dots, e_{|\mathbb{T}|-1}, e_{|\mathbb{T}|}\}, \emptyset)$  的  $\mathbb{I} = \{e_1, e_2, \dots, e_{|\mathbb{T}|-1}, e_{|\mathbb{T}|}\} = \mathbb{T}$  和  $\mathbb{O} = \emptyset$  而且它是无解的。我们将找到除  $\mathcal{P}(\{e_1, e_2, \dots, e_{|\mathbb{T}|}\}, \emptyset)$  其它每一个子问题的最优解, 然后取这些解中最优的一个作为原问题的最优解。如果所有的子问题都没有解则我们保证原问题也将没有解。

就时间复杂度而言, 这些子问题将比原问题花费更小的时间求得解, 因为每个子问题都至少有一条边 (来自  $\mathbb{T}$ ) 被移除出原图, 这样将减少  $AP$  路径路径复杂度, 这也保证了不同的  $AP$  将被求出来继续求是否存在 SRLG 分离的路径  $BP$ 。

当遭遇 trap 问题时, 我们的方法拆分原问题和测试每一个子问题为了找到最优解。跟已经存在的算法比较, 我们的算法能够通过前面计算的结果和信息来找到其他替代的路径  $AP$ , 这样我们的算法能大大的减少时间花销。而在给定的路径  $AP$  上为了得到最小 SRLG 冲突边集合我们求最小子集覆盖来获得。

下面从五个方面来描述 CSLS 算法与其他四类算法的区别:

1. 路径权值: 路径上所有边的权重和。
2. 路径跳数: 这条路径的跳数总数。
3. 运行时间: 找到一对 SRLG 分离路径所花费的平均时间。
4. 算法加速比: 给定两个不同算法的运行时间, 表示为  $T_1$  和  $T_2$ , 这个算法  $alg_2$  相对于算法  $alg_1$  的算法加速比为  $S_{1-2} = T_1/T_2$ .
5. 核加速比: 一个并行程序的核数加速比是被定义为  $S_P = \frac{T_1}{T_p}$ ,  $P$  是指处理器的核数,  $T_1$  和  $T_p$  表示各自运行在 1 核和  $p$  核上的运行时间。
6. 效率: 定义为  $E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$  并且取值区间是在  $(0, 1]$  内。

图11是五种算法 AP 路径 BP 路径和分离路径对的归一化路径权值, 显而易见, 全部实现的算法如 SCLS,CoSE,KSP,ILP 和 IQCP 的 AP 路径能获得相同的权值, 但是他们的 BP 路径权值的不同导致路径对权值和的不同。因为这五种算法解决相同的 Min-Min SRLG 分离路由问题, 尽管他们都去的不同的 SRLG 分离路径

对，但是这些算法都达到求最小路径权值的路径 AP。尽管两个 ILP 基础的算法，ILP 和 IQCP 主要是找到最小的路径权值的 AP 和与 AP 路径 SRLG 分离的 BP 路径，所以 ILP 和 IQCP 的 BP 路径与其他两个算法不同。

图12是五种算法 AP 路径 BP 路径和分离路径对的归一化路径跳数。因为所有的算法都在求 SRLG 分离路径里较小路径尽量小而不是求最小跳数，即使他们 AP 路径有不同跳数但是有相同的 AP 权值。尽管在图11所有算法中 AP 路径的权值是小于 BP 路径的权值，但是在图12AP 路径的跳数可能不总是小于 BP 路径的跳数。

图??展示了不同算法在不同核数的情况下的运行时间。正是因为 KSP,ILP 和 IQP 不是并行算法，这些算法在不同核数的情况下运行时间几乎相等。我们算法 SCLE 和 CoSE 的运行时间随着处理的核数增多而降低，因为这两个算法能拆分原问题为多个子问题以至于并行执行和充分利用多核 CPU 的并行性来加快路径搜索的速度。尽管 CoSE 是并行算法，它的计算时间是大于 ILP 和 IQCP。一些可能的原因 1) 因为在 CoSE 算法中找到冲突 SRLG 集合的查找过程是不够效率的，2) 因为一个 SRLG 通常包含多条边，这个分离问题是基于冲突 SRLG 集合的以至于引入大量需要求解的子问题，这将导致大量的计算代价。不同于 CoSE，我们的算法 SCLE 是根据图论的最小割定理来找到当一条 AP 路径遇到 trap 问题时的冲突边集合，和从图??显示我们算法运行更少的时间。这图描述了我们冲突边集合查找算法的高效性，和我们根据 SRLG 冲突边集合的分而治之的算法和智能的 AP 查找过程极大的减少计算代价。KSP 是另一种处理 trap 问题的有效算法。而且，在不同算法的执行过程中，KSP 算法的运行时间是最大的。KSP 算法的主要问题是当候选 AP 路径不存在相应的 SRLG 分离路径时，这个下一个候选的 AP 路径的选择是仅仅根据路径长度而选择的。在我们研究的这 17 个拓扑结构中，当为了找到分离路径而一大堆的路径要经过测试，因此这是需要大量的计算时间的。在图??中显示一个例子来说明为什么 KSP 算法是如此不高效的。在这个图里，假设 SRLG 冲突边集合是  $e_1, e_2$ ，和  $e_1, e_2, e_3, e_4$  的链路权值是远远大于其他链路的权值。首先从  $s$  到  $d$  的前  $K$  条最短路径包含路径段  $e_1, e_2$ （标识为虚线），这将让首先的 AP 路径遇到 trap 问题。为了避免 trap 问题， $K$  值必须设定成一个较大值，这将让 KSP 算法带来很大的计算复杂度。当路径 AP 遇到 trap 问题时，我们能快速的识别出  $\{e_1, e_2\}$  是 SRLG 冲突边集合和拆分原问题成两个子问题  $\mathcal{P}(\emptyset, \{e_1\})$  和  $\mathcal{P}(\{e_1\}, \{e_2\})$  以至于能在多核 CPU 里并行快速的找到 SRLG 分离路径对。

图??是五种算法不同核数下与 KSP 算法时间比较的归一化算法加速比。为了计算加速比度量，我们使用 KSP 作为基准算法和设置  $alg1 = KSP$ . 类似于图??的结果，我们的算法 SCLE 能获得明显的较大加速比。

图??是五种算法不同核数下的核加速比，为了比较两个并行算法 SCLE 和 CoSE，这个核数加速比随着核数的增加而增加。尽管增加的速度变得越来越小随着核数的增多和更高的代价来协调进程。KSP,ILP 和 IQCP 三种算法的核加速比在不同核数的情况下几乎等于 1 因为他们都不是并行算法，在所有的算法中，我

们算法 SCLS 的核加速比是最大的，这描述了基于 SRLG 冲突边分离的分而治之的算法能带来巨大的并行效应。

## 附图说明

- 图6 最大流最小割定理实例；
- 图2 风险共享链路组实例；
- 图3 求得第一条路径的例图实例；
- 图4 去除第一条路径后的残余图实例；
- 图5 原问题分而治之的原理图；
- 图6 构造求解最小割例图的实例；
- 图7 割集图实例；
- 图?? 算法步骤图；
- 图11 五种算法归一化路径权值比较柱形图；
- 图12 五种算法归一化路径跳数比较柱形图；
- 图10 KSP 算法在 trap 情况下的示意图；
- 图?? 五种算法在多核情况下运行时间比较柱形图；
- 图?? 五种算法在多核情况下算法加速度比比较柱形图；
- 图13 五种算法在多核情况下核数加速度比比较柱形图；
- 图15 五种算法在多核情况下运行效率比较柱形图；

## 具体实施方式