

# On the Stable Paths Problem and a Restricted Variant

Kevin Donnelly and Assaf Kfoury

February 5, 2008

## Abstract

Interdomain routing on the Internet is performed using route preference policies specified independently, and arbitrarily by each Autonomous System in the network. These policies are used in the border gateway protocol (BGP) by each AS when selecting next-hop choices for routes to each destination. Conflicts between policies used by different ASs can lead to routing instabilities that, potentially, cannot be resolved no matter how long BGP is run.

The Stable Paths Problem (SPP) is an abstract graph theoretic model of the problem of selecting next-hop routes for a destination. A stable solution to the problem is a set of next-hop choices, one for each AS, that is compatible with the policies of each AS. In a stable solution each AS has selected its best next-hop given that the next-hop choices of all neighbors are fixed. BGP can be viewed as a distributed algorithm for solving SPP.

In this report we consider the stable paths problem, as well as a family of restricted variants of the stable paths problem, which we call  $F$ -stable paths problems. We show that two very simple variants of the stable paths problem are also NP-complete. In addition we show that for networks with a DAG topology, there is an efficient centralized algorithm to solve the stable paths problem, and that BGP always efficiently converges to a stable solution on such networks.

## 1 Introduction

Today's Internet is a collection of thousands of Autonomous Systems (ASs) connected in a complex graph. Each AS is administratively separate and potentially has its own policies about what routes through the graph ought to be used for traffic to each destination AS. The border gateway protocol (BGP) is a distributed algorithm used for selecting interdomain paths to each destination, and it makes use of policy preferences in making these selections.

However, routers within the network generally route based only on the destination address of a packet. As a consequence, the choice of routes that may be taken by packets originating from any AS are constrained by the routes chosen by each neighbor of that AS, which are further constrained by the choices of their neighbors, and so on. It is possible that the policies of different ASs conflict in such a way as to cause permanent instability in the network, i.e. that there is always some AS that wants to change its route selection in response to the current set of routes offered by its neighbors.

The Stable Paths Problem, introduced in [5], is an abstract graph theoretic model of Internet route selection. An instance of SPP is a graph of ASs with arbitrary policies about preferences between routes. The problem of whether an SPP instance has a solution was shown to be NP-complete in [5].

In this paper we consider a restricted, though still natural, version of the stable paths problem in which routing policies are based only on assigning weights to each link and aggregating the weights using some

function (natural choices are addition, minimum and multiplication). We show NP-completeness for several variants of this problem.

Our results show that even if policies are not selected arbitrarily but based on a small set of synthetic link metrics, the SPP is still NP-complete. In particular, even if there are only two policies used, minimum latency and maximum bottleneck bandwidth, solvability of SPP is still NP-complete.

## 2 The Stable Paths Problem

### 2.1 SPP definition

In this section we describe our formulation of the Stable Paths Problem, which is similar to the one described in [5], except that it uses a directed graph. Consider an arbitrary network of ASs represented as a directed graph  $G = (V, E)$  where each vertex  $X \in V$  represents some AS and each edge  $(X, Y) \in E$  represents a link from  $X$  to  $Y$ . For convenience we assume  $V = \{d, 1, \dots, n\}$ . The node  $d$ , called the *destination*, is the particular destination that each AS wants to send traffic to. For each node  $X \in V$ , we define  $\text{peers}(X) = \{Y \mid (X, Y)\}$ .

A path from  $X$  to  $Y$  in  $G$  is a sequence of nodes  $X_0 X_1 \dots X_k$  such that  $X_1 = X$  and  $X_k = Y$  and  $(X_i, X_{i+1}) \in E$  for each  $i \in \{0, \dots, k-1\}$ . A path of length 0 from  $X$  to  $X$  is just denoted by  $X$ . For convenience, we ignore sequentially repeated nodes in a path, so  $XYYZ = XYZ$ , and use juxtaposition to denote concatenation, so  $PQ$  denotes  $P$  followed by  $Q$ . If  $P$  is a path from  $X$  to  $Y$  and  $Q$  is a path from  $Y$  to  $Z$  then  $PQ$  is a path from  $X$  to  $Z$ . For each  $X \in V$ ,  $P_X$  is the set of permitted paths from  $X$  to the destination  $d$ . All paths in  $P_X$  must be cycle-free, and be of the form  $XPd$  for some path  $P$ , that is, they must all start from  $X$  and end at  $d$ . We also use the symbol  $\perp$  to denote the lack of a path and define  $\perp P = \perp = P \perp$ .  $P$  is the sequence containing the permitted paths for each non-destination node, i.e.  $P = \langle P_X \rangle_{X \in \{1, \dots, n\}}$ .

For each  $X \in V$  there is a cost function  $W_X$  which assigns to each path  $P \in P_X$  a non-negative integer or  $\infty$ . We require  $W_X(\perp) = \infty$  and for  $P \neq \perp$ ,  $W_X(P) < \infty$ . This weight represents the path preference policies of node  $X$ . For  $P, Q \in P_X$ , if  $W_X(P) < W_X(Q)$  then  $X$  prefers path  $P$  to path  $Q$ .  $W$  is the sequence containing the ranking function for each non-destination node, i.e.  $W = \langle W_X \rangle_{X \in \{1, \dots, n\}}$ .

An instance of the Stable Paths Problem can be thought of as a network of ASs with particular routing policies. An instance,  $N = (G, P, W)$ , is a graph together with the permitted paths and ranking functions for each non-destination node.

A *configuration*  $\mathbb{C}$  of an SPP instance  $N$  is a directed spanning tree of  $N$  rooted at  $d$ , that is it is a set of paths that satisfies

1. if  $XP \in \mathbb{C}$  then  $X \in P_X$ ,
2. if  $XP \in \mathbb{C}$  then  $P \in \mathbb{C}$ , and
3. if  $XP \in \mathbb{C}$  and  $YQ \in \mathbb{C}$  and  $XP \neq YQ$  then  $X \neq Y$ .

So  $\mathbb{C}$  contains only permitted paths, the suffix of each path in  $\mathbb{C}$  is also in  $\mathbb{C}$ , and for each node  $X \in V$ ,  $\mathbb{C}$  contains at most one path starting from  $X$ . For every node  $X \in V$ , we define

$$\mathbb{C}(X) = \begin{cases} XP & \text{if } XP \in \mathbb{C} \\ \perp & \text{o.w.} \end{cases}$$

So  $\mathbb{C}(X)$  is the unique path from  $X$  to  $d$  that is an element of  $\mathbb{C}$ , or else  $\perp$  if there is no such path. The configuration can be interpreted as a set of routing next-hop choices and  $\mathbb{C}(X) = \perp$  means that  $X$  has no route to  $d$ .

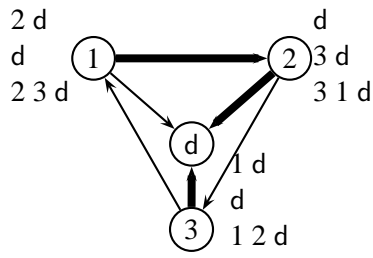


Figure 1: GOOD GADGET

The set  $\text{choices}(\mathbb{C}, X)$  represents the legal path choices for  $X$ , given the current choices for his neighbors. This is defined by

$$\text{choices}(\mathbb{C}, X) = \begin{cases} \{X \mathbb{C}(Y) \mid (X, Y) \in E\} \cap P_X & \text{if } X \neq d \\ \{d\} & \text{o.w.} \end{cases}$$

Given the next-hop choices available to a node  $X$ , the weight of the best path from  $X$  is fixed. We define  $\text{bestweight}(\mathbb{C}, X)$  by

$$\text{bestweight}(\mathbb{C}, X) = \min(\{\infty\} \cup \{W_X(P) \mid P \in \text{choices}(\mathbb{C}, X)\})$$

The configuration  $\mathbb{C}$  is *stable* if for all  $X \in V$ ,  $\text{bestweight}(\mathbb{C}, X) = W_X(\mathbb{C}(X))$ , i.e. no node prefers to change its next-hop choice, given that the choices of neighbors are fixed.

An SPP instance  $N$  is *solvable* if there exists a stable configuration for  $N$ , and *unsolvable* if none of the configurations for  $N$  are stable.

**Theorem 2.1 (SPP is NP-complete).** *The problem of determining whether an instance of the Stable Paths Problem is solvable is NP-complete.*

*Proof.* It is clear that a configuration can be checked for stability in polynomial time, so SPP is in NP. In [5] it is shown that 3-SAT can be reduced to SPP.  $\square$

## 2.2 Examples

An instance of SPP can be succinctly described by giving the graph and a list of the permitted paths for each node in preference order. Figure 1 shows a network with three nodes and a destination. The permitted paths for each node are listed next to the node. This network has a unique solution, which is shown with heavy lines. Figure 2 shows a network with the same topology and set of permitted paths, but this network has no stable solutions.

## 3 A Restriction of the Stable Paths Problem

### 3.1 $F$ -SPP definition

Given the NP-completeness of the Stable Paths Problem, it is natural to ask whether there is a more restricted formulation of the problem that has a polynomial time solution and is still useful in practice. The approach that we will take is to restrict the sorts of policies that are allowed. We will define a family of restrictions,

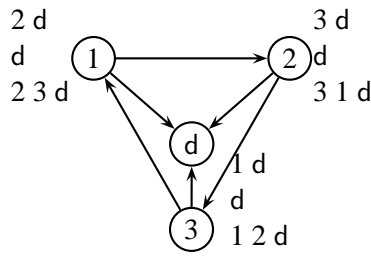


Figure 2: BAD GADGET

referred to as  $F$ -SPP, and investigate the properties of some very simple (i.e. highly restricted) problems in this family.

$F$ -SPP will only allow policies which are generated by assigning weights to each edge and combining the weights with one of a limited set of aggregation functions. Depending on the aggregation functions allowed, this form of policy can be very flexible; using it we can encode preference metrics such as minimum latency, maximum bottleneck bandwidth, minimum end-to-end packet loss probability, or simpler metrics which have only a preferred next-hop for each destination.

The problem is again modeled by a directed graph,  $G = (V, E)$ , representing ASs and connections between them, and we assume  $V = \{d, 1, \dots, n\}$ .  $\text{peers}(X)$ , paths and path concatenation are defined as previously for SPP.

In the restricted problem, there are  $k \leq n$  different policies, represented by an integer in  $T = \{1, \dots, k\}$ . Each node,  $X \in V$ , is assigned a label  $l(X) \in T$ . Each policy,  $\tau \in T$ , is associated with edge weight function  $W_\tau$  and an aggregation function  $F_\tau$ . For each  $e \in E$ ,  $W_\tau(e)$  is a non-negative integer.  $F_\tau$  is a function on sequences of non-negative integers, which combines the cost of the edges in a path to get the weight of the path. In this paper we will specify aggregation functions by associative, commutative binary functions and set  $F_\tau = W_\tau$  (i.e. the weight of a path of length one is the weight of its single edge). The sequence of weight functions is denoted by  $W$  and the sequence of aggregation functions is denoted by  $F$ . For example, a policy  $\tau_{lat}$  may prefer lowest latency paths by setting

$$W_{\tau_{lat}}(X, Y) = \text{“the latency of the link from } X \text{ to } Y\text{”}$$

and  $F_{\tau_{lat}} = +$ .

In our restricted problem, any simple path from a node to the destination is permitted. That is

$$P_X = \{P \mid P \text{ is a simple path from } X \text{ to } d \text{ in } G\}$$

An instance of the  $F$ -Stable Paths Problem,  $N = (G, l, W)$ , is a directed graph representing ASs and connections between them, an assignment of policies to each AS and an assignment of edge weights for each policy. For example, an instance of the  $\langle +, +, + \rangle$ -Stable Paths Problem assigns to each node one of three policies, each of which uses addition as the aggregation function but may assign different weights to the edges.

As previously, a *configuration*  $\mathbb{C}$  for instance  $N$  is a spanning tree of  $N$  rooted at  $d$ , with all paths directed from the leaf nodes of  $\mathbb{C}$  to  $d$ . For every node  $X \in V$ , the path from  $X$  to  $d$  according to  $\mathbb{C}$  is denoted  $\mathbb{C}(X)$ . The definitions of choices and bestweight are as before:

$$\text{choices}(\mathbb{C}, X) = \begin{cases} \{X \mathbb{C}(Y) \mid (X, Y) \in E\} \cap P_X & \text{if } X \neq d \\ \{d\} & \text{o.w.} \end{cases}$$

$$\text{bestweight}(\mathbb{C}, X) = \min(\{\infty\} \cup \{W_X(P) \mid P \in \text{choices}(\mathbb{C}, X)\})$$

The configuration  $\mathbb{C}$  is *stable* if for all  $X \in V$ ,  $W_X(\mathbb{C}(X)) = \text{bestweight}(\mathbb{C}, X)$ . The interpretation of stability remains the same: every node prefers its path over all others, given that the choices of neighbors are fixed.

It is clear that any instance of the  $F$ -SPP can be translated into an instance of SPP having the same topology and set of stable configurations.

## 4 Examples

Several examples that are useful in understanding some of the complications encountered. In order to succinctly describe instances of  $F$ -SPP, we use graphs where the shapes of the node represent the policy type used at that node and each edge is labeled with its weight under each policy.

**Example 4.1.** Figure 3 shows an instance of  $\langle +, +, + \rangle$ -SPP, which is equivalent to the SPP instance shown in Figure 1, along with a stable configuration. Figure 4 shows an unsolvable instance of  $\langle +, +, + \rangle$ -SPP, which is equivalent to the SPP instance shown in Figure 2. Figure 5 shows an instance of  $\langle +, + \rangle$ -SPP with no stable configurations.

**Example 4.2.** Figure 6 is a network with 2 routing policies which has no stable configurations. This fact was verified by exhaustively checking all configurations of the network using a computer program – a total of 63 valid configurations were checked. We refer to the three nodes with edges directly to  $d$  as “intermediate nodes” and the 6 nodes with edges to the intermediate nodes as “leaf nodes.” This network, and minor variations play a crucial role in the proof of Theorem 5.2.

**Example 4.3.** Figure 7, Figure 8 and Figure 9, are three networks with a stable configuration, obtained by reducing the weights on exactly two of the three edges from the intermediate nodes to  $d$  (from  $7/7$  to  $2/2$ ). That the exhibited configuration in each of the three figures is stable is easy to check. Understanding these is helpful in following the proof of Theorem 5.2.

**Example 4.4.** The next example, shown in Figure 10, has the same topology as the previous network, but this time all three edges from intermediate nodes to the destination are reduced from  $7/7$  to  $2/2$ . This results in a network with no stable configuration.

## 5 Negative Results

The results in this section are “negative” in that they show some computational problems in the  $F$ -SPP family to be NP-complete, and therefore (most probably) beyond resolution by any efficient algorithm, current or future. Note that because  $F$ -SPP is a restriction of SPP, these results also subsume NP-completeness of SPP.

The next lemma is used in the proof of Theorem 5.2 together with the networks shown in Figures 6, 7, 8, 9, 10, and 11. The proof consists in building a network  $N(G)$  from a directed graph  $G$ , where the in-degree of every vertex is at most 3, such that  $G$  is Hamiltonian iff  $N(G)$  has a stable configuration.

**Lemma 5.1.** *Let  $G$  be a directed graph, where the in-degree of every vertex  $\leq 3$ . Whether  $G$  has a Hamiltonian circuit is an NP-complete problem.*

**Theorem 5.2.**  $\langle +, + \rangle$ -SPP is NP-complete.

**Theorem 5.3.**  $\langle +, \max \rangle$ -SPP is NP-complete.

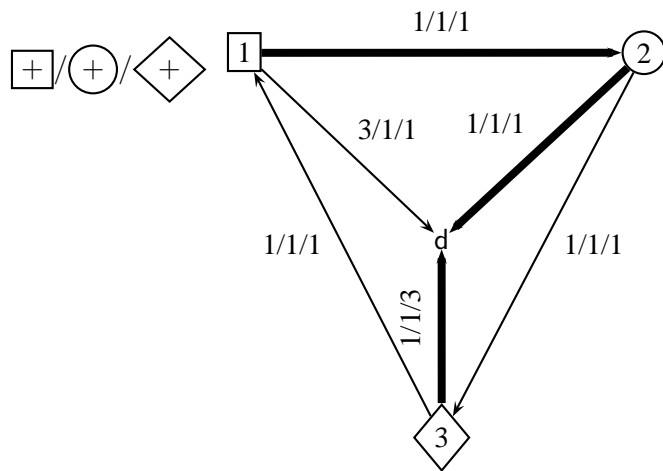


Figure 3: GOOD GADGET

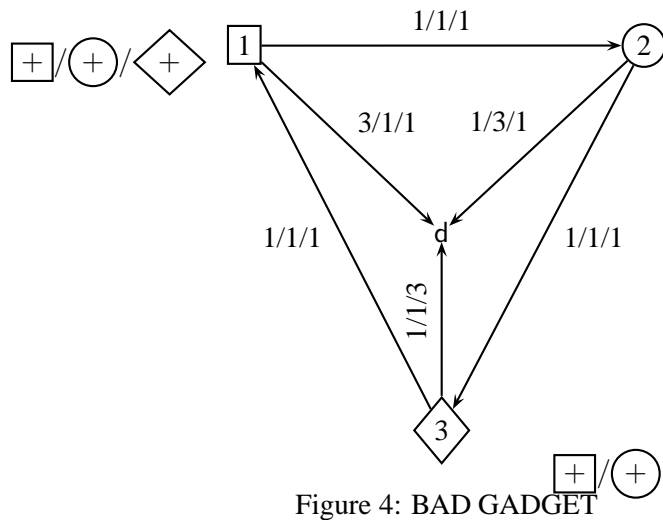


Figure 4: BAD GADGET

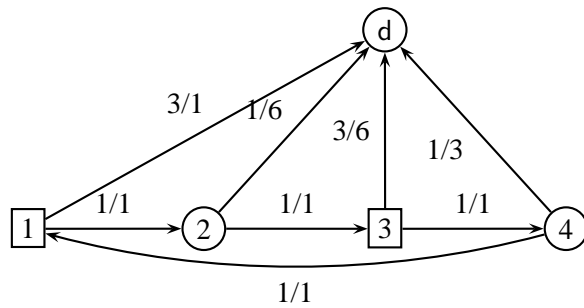


Figure 5:  $\langle +, + \rangle$ -SPP network with no stable configuration.

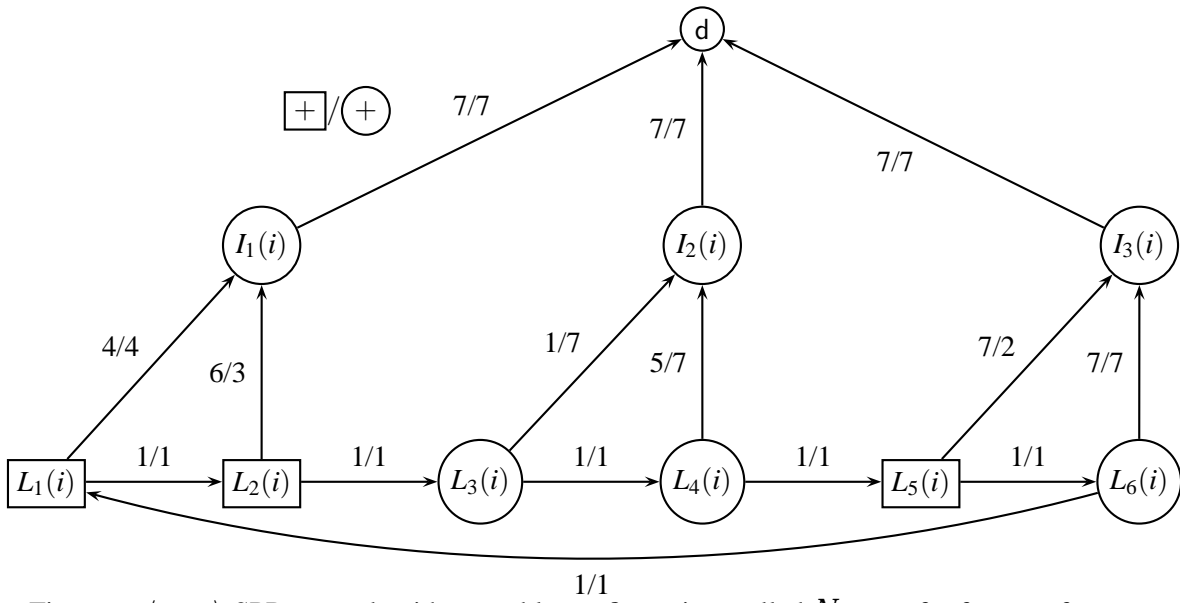


Figure 6:  $\langle +, + \rangle$ -SPP network with no stable configuration, called  $N_{\text{unstable}}$  for future reference.

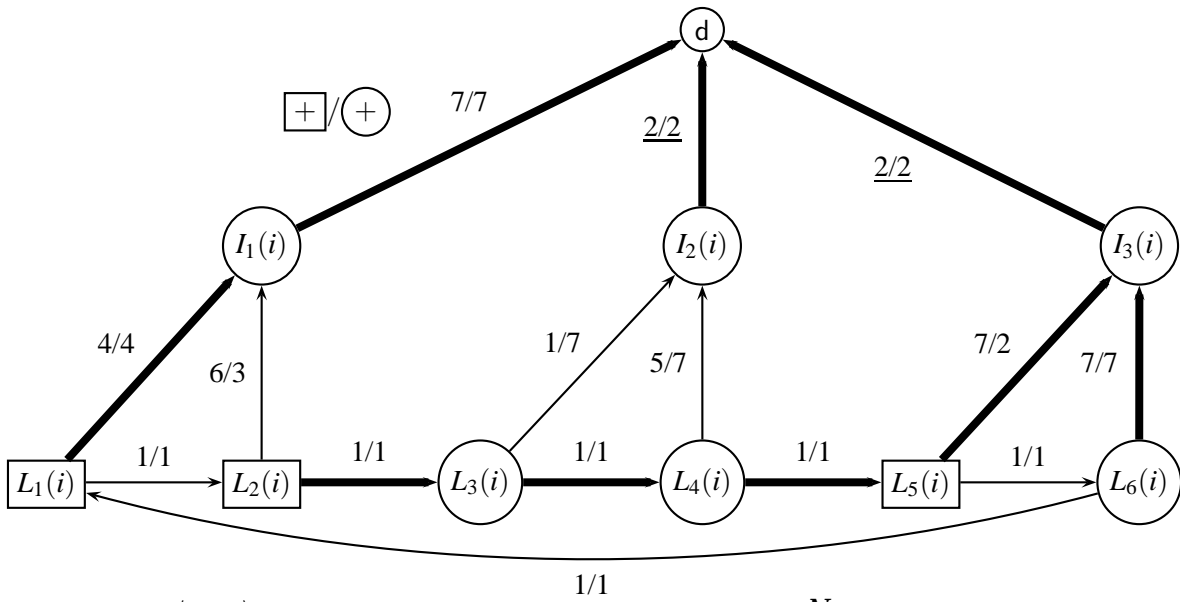


Figure 7:  $\langle +, + \rangle$ -SPP network with stable configuration, called  $N_{\text{stable}}$  for future reference.

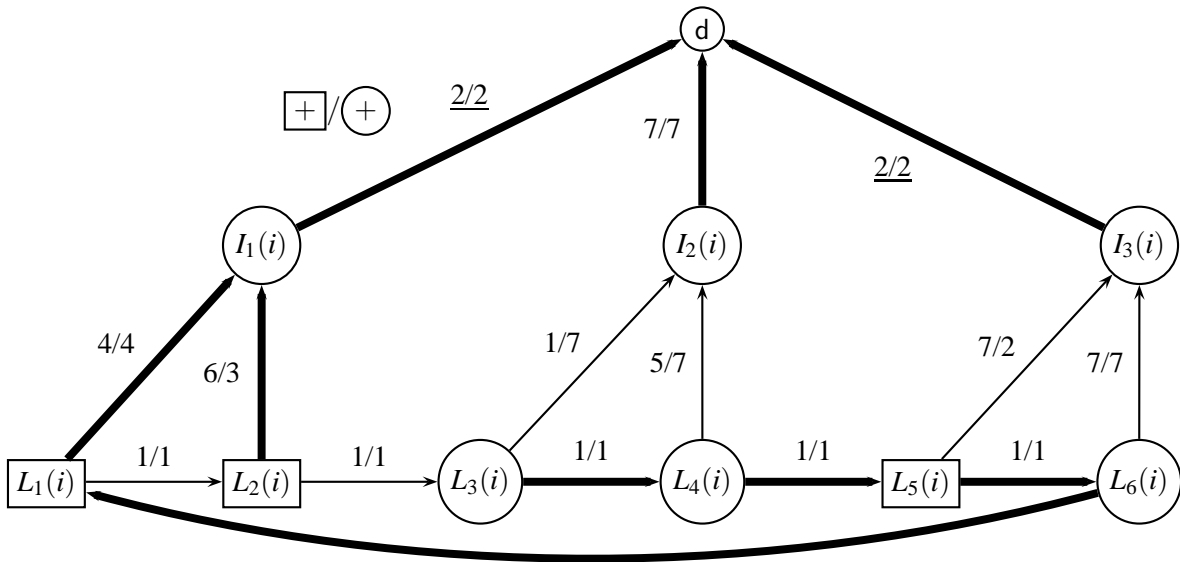


Figure 8:  $\langle +, + \rangle$ -SPP network with stable configuration, called  $N'_{\text{stable}}$  for future reference.

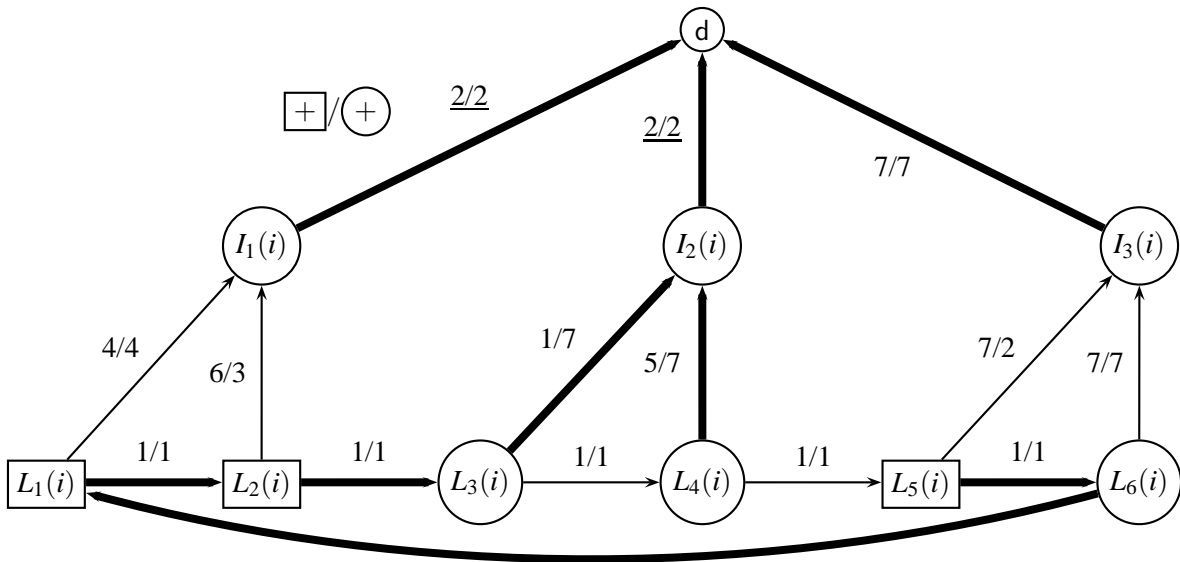


Figure 9:  $\langle +, + \rangle$ -SPP network with stable configuration, called  $N''_{\text{stable}}$  for future reference.

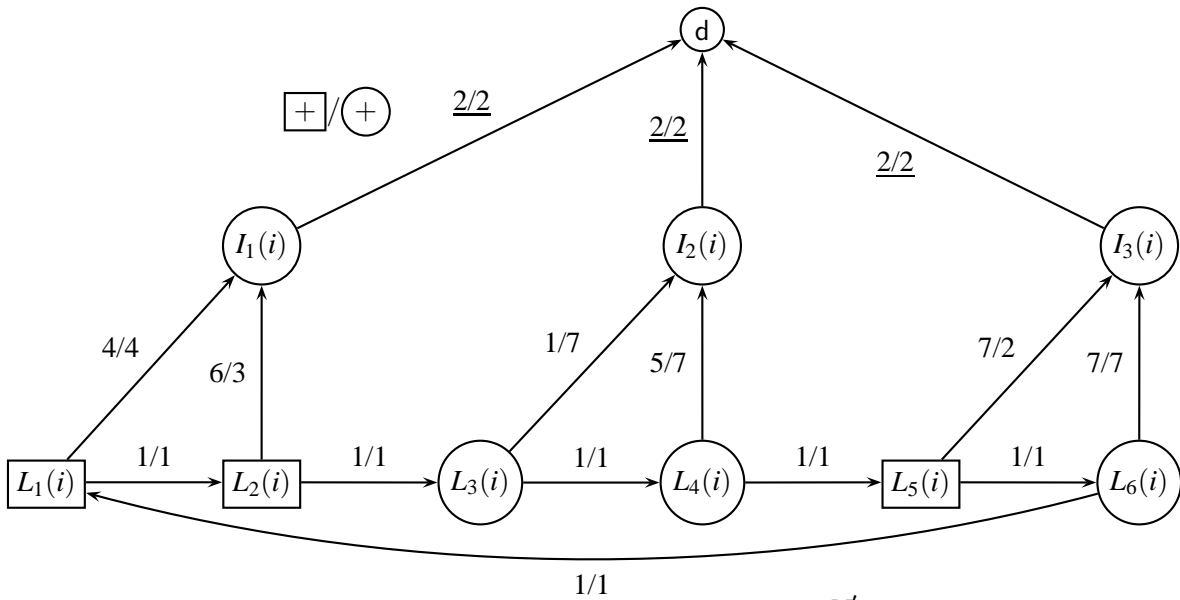


Figure 10:  $\langle +, + \rangle$ -SPP network with no stable configurations, called  $N'_{\text{unstable}}$  for future reference.

## 6 Positive Results

The results in this section describe subsets of SPP and  $F$ -SPP problems which can be efficiently solved.

### 6.1 DAG topology

For the case of SPP or  $F$ -SPP over a DAG topology, a stable solution always exists and can be found using the following algorithm:

1. Use a topological sorting algorithm to list nodes such that no node has an edge directed towards a node later in the list. This can be done in  $O(|V| + |E|)$ .
2. Starting from the front of the list, have each node select the best path currently available. Because each outgoing edge is considered only one, this is  $O(|E|)$ .

**Theorem 6.1.** *After one pass through the list, we have reached a stable configuration.*

*Proof.* By induction on the topologically sorted list of node we will show invariant that after we first choose a path for a node, no better path will ever become available.

When we consider the paths of a node, the first hop of each path must be an edge directed towards a node earlier in the list, so the induction hypothesis applies to it. By induction, each of these next-hop nodes has already chosen the best path that will ever be available to it, so it will never change its chosen path. Therefore the best path currently available to the current node is the best path the will ever be available to the current node.

At the end of one pass, each node has chosen the best path available to it, so the resulting configuration is stable.  $\square$

The algorithm described above requires complete knowledge about the topology and policies of the graph (we will refer to this as the “omnipotent algorithm”), so it is not practical for use in a real network. In real networks the BGP protocol chooses paths by repeatedly choosing the best available path given knowledge of the paths chosen by upstream neighbors, and then telling its downstream neighbors what its chosen path is. This protocol is also guaranteed to converge on a stable configuration in a DAG topology. Further more, if we assume the algorithm operates in synchronous rounds in which all nodes simultaneously pick their best available path and then inform neighbors of available paths, we can guarantee convergence in  $O(|V|)$  rounds.

**Theorem 6.2.** *For a DAG topology and assuming synchronous operation at each node, the BGP protocol will converge to a stable configuration after at most  $|V|$  iterations.*

*Proof.* We will show by induction on  $i$ , that after the  $i^{\text{th}}$  iteration, no node whose longest path to  $d$  has length  $i$  will ever change the path to  $d$  that it has selected. Assume we are on the  $i^{\text{th}}$  iteration. There is some set of nodes whose longest path has length  $i$ . If the set is empty, we are done. Otherwise, for each such node, consider the set of nodes that it has edges directed towards. The longest path for each of these nodes has length strictly less than  $i$ . By induction none of these nodes will change its path in this round or any subsequent round. Because the choice of paths in BGP does not change unless the paths advertised by neighbors change, the choice for this node will not change after this round. The longest path from any node to  $d$  is bounded by  $|V|$ , so we must have a stable configuration after  $|V|$  rounds.  $\square$

## 7 Related/Future Work

There have been many approaches toward solving the BGP convergence problem, which fall into two broad categories: static and dynamic.

Dynamic approaches address the convergence problem by modifying the BGP protocol. BGP route flap damping [6] is a dynamic approach that attempts to detect routing oscillations and slow them down to reduce their negative effects on the network. However, this is not really a solution because it does not eliminate divergence where it exists, furthermore, flap damping can also have the negative effect of slowing down progress toward convergence.

The safe path vector protocol [4] forces convergence by augmenting path update messages with a list of path update events that caused it. This path update history is then examined for cycles, which occur when an update by one AS indirectly *causes* an update in the same AS. Paths which cause such update cycles are removed from the set of permitted paths for the AS when the update cycle is detected. This approach provably leads to BGP convergence. However, it increases communication costs because each path update message must also include an update event history. It also has the significant downside that, because path update cycles are a necessary but not sufficient condition for divergence, it forces ASs to change their policies when convergence may have been possible under the original policies.

Adaptive Policy Management [7] is another dynamic solution to BGP convergence in which each AS keeps a local history of how many times it selects and then gives up a route. Using this count, ASs adapt their policies so as to attempt to cause convergence, while occasionally trying to revert to their more preferred paths, which may have become stable. This approach shares a mixture of the benefits and drawbacks of the previous two approaches.

Existing static solutions achieve BGP convergence by strongly restricting the policies that are allowed. The hierarchical routing solution described in [1] requires all ASs to consistently categorize links as customer, peer or provider and to strictly prefer customer routes over peer or provider route and peer route over provider

route and, furthermore, to only route through a peer if the next hop after the peer is to a customer of that peer. This induces a DAG topology in the graph of possible routes which guarantees convergence.

The dispute wheel analysis [3] provides static method for proving the solvability of an SPP instance. However, this analysis is sufficient, but not necessary for convergence. In fact the criterion is extremely conservative and cannot prove solvability of any SPP instance which has multiple possible solutions, which probably means it cannot be directly applied to complex, large scale networks.

One might reasonably hope to develop further static solutions to the convergence problem by restricting the types of policies or network topologies that are allowed. In this work we have ruled out some such static solutions to the BGP convergence problem by identifying a broad and natural class of routing policies for which it is NP-complete to decide convergence. In future work we hope to discover other combinations of policies and network topologies for which convergence is guaranteed or easy to decide.

## References

- [1] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *IEEE/ACM Trans. Netw.*, 9(6):681–692, 2001.
- [2] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
- [3] Timothy Griffin, F. Bruce Shepherd, and Gordon T. Wilfong. Policy disputes in path-vector protocols. In *Proceedings of the 7th Annual International Conference on Network Protocols*, pages 21–30, Toronto, Canada, November 1999.
- [4] Timothy Griffin and Gordon T. Wilfong. A safe path vector protocol. In *INFOCOM*, pages 490–499, 2000.
- [5] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002.
- [6] C. Villamizar, R. Chandra, and R. Govindan. BGP route flap damping, 1998. RFC 2439.
- [7] Selma Yilmaz and Ibrahim Matta. An Adaptive Policy Management Approach to BGP Convergence. Technical Report BUCS-TR-2005-028, CS Department, Boston University, July 7 2005.

## A Proofs

### A.1 Proof of Lemma 5.1

Let  $G$  range over the set of undirected graphs where every node has degree at most 3. Whether such a graph  $G$  has a Hamiltonian circuit is an NP-complete problem [2]. This implies whether an arbitrary directed graph where the in-degree and the out-degree of every node  $\leq 3$  has a Hamiltonian circuit is an NP-complete problem. This implies our lemma.

### A.2 Proof of Theorem 5.2

Let  $G$  be a directed graph where every node has in-degree  $\leq 3$ . We shall construct a network  $N(G)$  from  $G$  efficiently, i.e. in polynomial time (in fact in logarithmic space), such that  $G$  has a Hamiltonian circuit iff  $N(G)$  has a stable configuration. This will imply the theorem. The construction of  $N(G)$  is in 3 stages: (1) We first construct an intermediary graph  $G'$  from  $G$ , (2) we then construct what we call the “node substitutes”, i.e. the subgraphs that will be substituted for the nodes in  $G'$ , and (3) we finally construct  $N(G)$  by assembling and connecting all the node substitutes. We can assume that every node in  $G$  is accessible from every other node in  $G$ , otherwise we can immediately determine that  $G$  does not have a Hamiltonian circuit; this implies, in particular, that every node has at least one incoming edge and at least one outgoing edge.

#### Construction of $G'$

For convenience, let  $G = \langle \{1, 2, \dots, n\}, E \rangle$ , i.e.  $G$  has  $n \geq 1$  nodes denoted by the natural numbers from 1 to  $n$  and  $E \subseteq \{1, \dots, n\} \times \{1, \dots, n\}$ . We first define a new graph  $G'$  from  $G$  by splitting node 1 into two nodes, say 1 and  $1'$ , such that  $1'$  has only outgoing edges and 1 has only incoming edges. The rest of the graph remains unchanged. Specifically, let:

$$\begin{aligned} G' &= \langle \{1, 1', 2, \dots, n\}, E' \rangle \quad \text{where} \\ E' &= \{(j, 1) \mid (j, 1) \in E\} \cup \{(1', k) \mid (1, k) \in E\} \cup \\ &\quad E \cap \{2, \dots, n\} \times \{2, \dots, n\} \end{aligned}$$

It is easy to see that  $G$  has a Hamiltonian circuit iff  $G'$  has a Hamiltonian path that starts at node  $1'$  and ends at node 1. By hypothesis, every node in  $G$  has at most 3 incoming edges, and this property is inherited by  $G'$ .

#### Construction of The Node Substitutes

The desired network  $N(G)$  is obtained by replacing every node  $i \in \{1, 2, \dots, n\}$  – but not the new node  $1'$  – in  $G'$  by a 16-node graph of the form shown in Figure 11 and denoted  $\text{NodeSubstitute}(i)$ . We use several conventions to help understand the functioning of  $\text{NodeSubstitute}(i)$ :

- Edges inherited from  $G$  are in heavy boldface.
- Nodes and edges shown in dashed lines are not part of  $\text{NodeSubstitute}(i)$  but rather of a different node substitute. There may be more edges into or out of the dashed nodes in  $N(G)$  that are not shown in the figure.
- The subgraph with edges in medium boldface is a copy of  $N_{\text{unstable}}$ , the inherently unstable network in Figure 6.

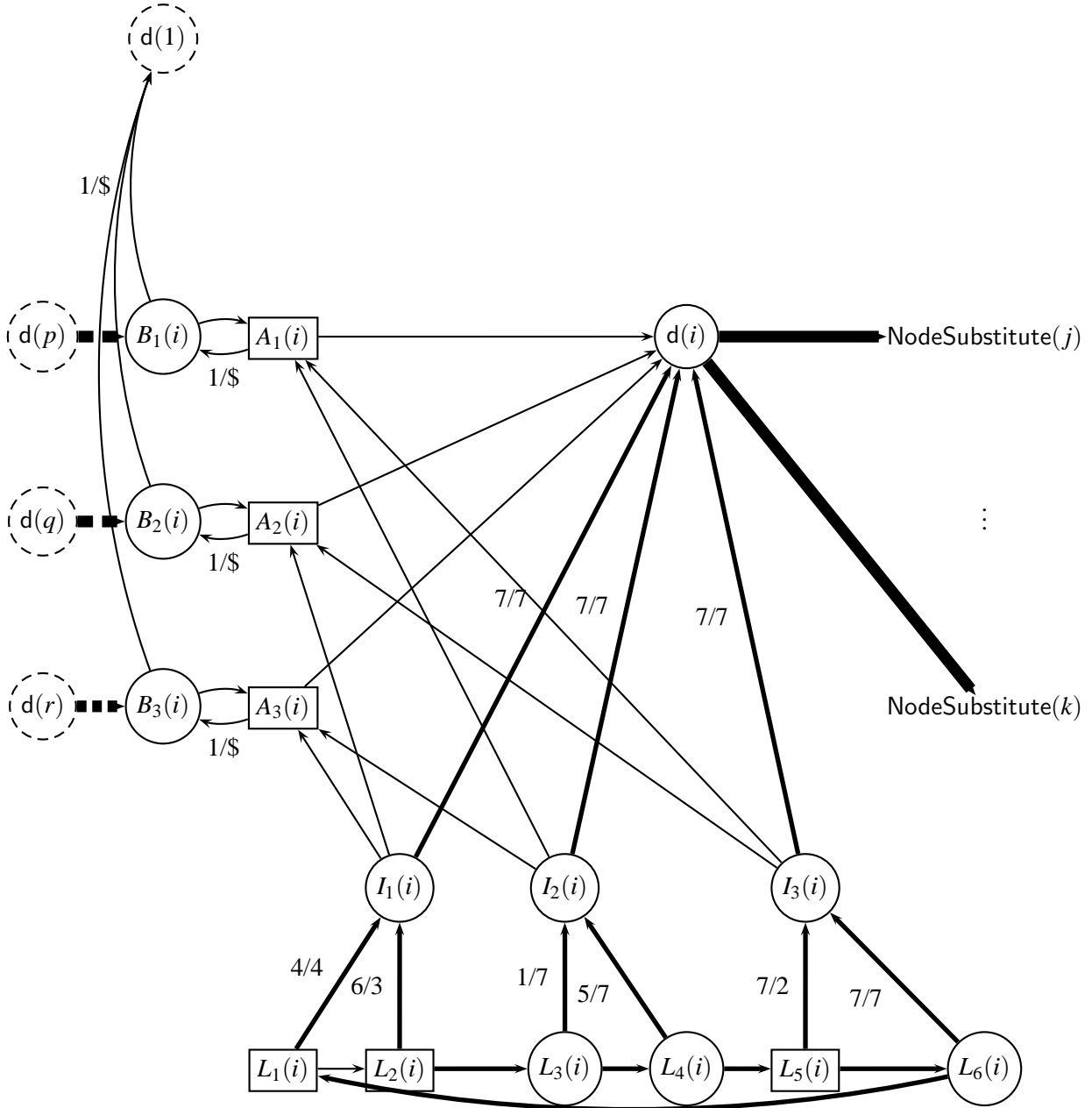


Figure 11:  $\text{NodeSubstitute}(i)$  replaces node  $i$  in the construction of network  $N(G)$  from graph  $G$ .

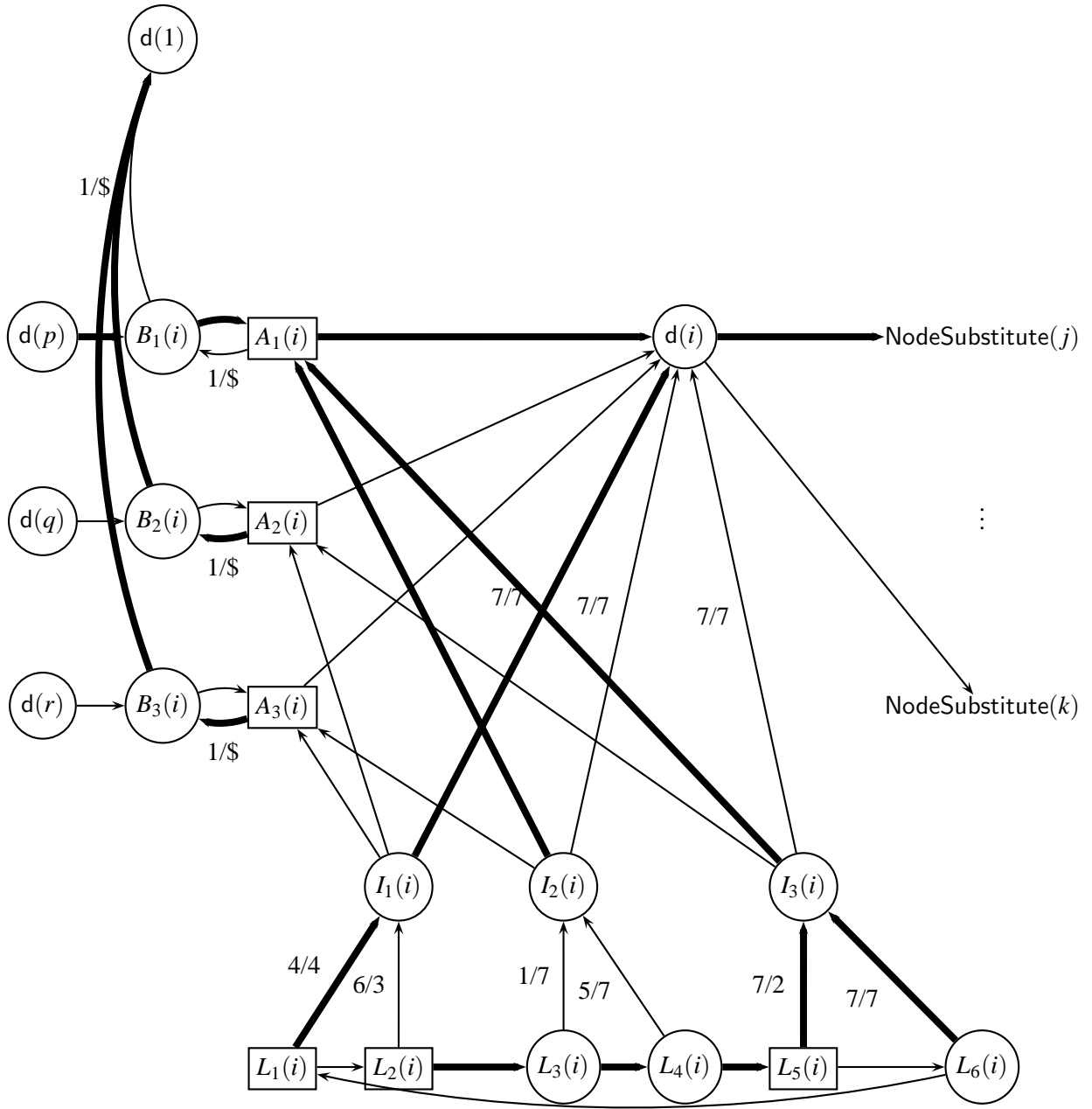


Figure 12:  $\text{NodeSubstitute}(i)$  replaces node  $i$  in the construction of network  $N(G)$  from graph  $G$ . The boldface edges correspond to a possible stable configuration of  $N(G)$ .

- The weight denoted “\$” is a “very large number”, e.g. an integer strictly larger than the sum of all the other weights in  $N(G)$  other than “\$” itself.
- For simplicity we omit weight labels of edges whose weight assignment is “1/1”.

The node omitted from this substitution,  $l'$ , simply becomes node  $d(l')$  in  $N(G)$ . Figure 11 corresponds to the case when node  $i$  has exactly three incoming edges – namely,  $\{(p,i), (q,i), (r,i)\}$  – and one or more outgoing edges – namely,  $\{(i,j), \dots, (i,k)\}$ .

If the node  $i \in \{1, 2, \dots, n\}$  has only one or two incoming edges in  $G$  – say,  $\{(p,i)\}$  or  $\{(p,i), (q,i)\}$ , resp. – we simply omit the other subgraphs in the construction – namely, the subgraph consisting of the nodes  $\{d(q), B_2(i), A_2(i), d(r), B_3(i), A_3(i)\}$  or  $\{d(r), B_3(i), A_3(i)\}$ , resp., together with all the edges they touch. These cases are not shown in Figure 11.

Consider the subgraph  $\text{NodeSubstitute}(i)$  of  $N(G)$ , as shown in Figure 11. It shows the case when node  $i$  has exactly three incoming edges, with the following correspondence:

- Edge  $(p,i)$  in  $G'$  is mapped to edge  $(d(p), B_1(i))$  in  $N(G)$ .
- Edge  $(q,i)$  in  $G'$  is mapped to edge  $(d(q), B_2(i))$  in  $N(G)$ .
- Edge  $(r,i)$  in  $G'$  is mapped to edge  $(d(r), B_3(i))$  in  $N(G)$ .

Each of the edges outgoing from node  $i$ , say  $(i,j)$ , in  $G$  is mapped to an edge “from  $d(i)$  to  $\text{NodeSubstitute}(j)$ ” in  $N(G)$ , as shown in Figure 11. This means  $(i,j)$  is mapped to an edge “from  $d(i)$  to one of the 3 nodes in  $\{B_1(j), B_2(j), B_3(j)\}$ ” – it does not matter which of the 3 – which are the 3 entry points of  $\text{NodeSubstitute}(j)$ . A few additional clarifications about the construction of  $\text{NodeSubstitute}(i)$ :

1. For every  $i \in \{2, 3, \dots, n\}$ , node  $d(i)$  has exactly 4 (or 5 or 6, resp.) incoming edges, if node  $i$  in  $G$  has 1 (or 2 or 3, resp.) incoming edges; and  $d(i)$  has exactly as many outgoing edges as  $i$  has outgoing edges in  $G$ . In order to exit  $\text{NodeSubstitute}(i)$ , a path must use one of the following:
  - One of the outgoing edges of  $d(i)$ . Call these the *forward* exits of  $\text{NodeSubstitute}(i)$ .
  - One of the 3 edges directly linked to destination  $d(1)$ . Call these the *upward* exits of  $\text{NodeSubstitute}(i)$ , namely,  $(B_1(i), d(1))$  or  $(B_2(i), d(1))$  or  $(B_3(i), d(1))$ .

A path that enters and exits  $\text{NodeSubstitute}(i)$ , and touches  $d(i)$ , must contain one of the following 3 paths as a subpath:

- $d(p) B_1(i) A_1(i) d(i)$
- $d(q) B_2(i) A_2(i) d(i)$
- $d(r) B_3(i) A_3(i) d(i)$

Observe that the total weight of these 3 paths is the same, namely,  $3/3$  – that is, 3 according to each metric.

2. Node  $d(1)$  has between 4 and 6 incoming edges, part of the definition of  $\text{NodeSubstitute}(1)$ , in addition to as many incoming edges as there are upward exits in

$\text{NodeSubstitute}(1), \dots, \text{NodeSubstitute}(n),$

There are *no* outgoing edges from  $d(1)$ .

3. Node  $d(1')$  has *no* incoming edges and as many outgoing edges as node  $1'$  has in  $G'$  (equivalently, as node 1 has in  $G$ ).

Every  $\text{NodeSubstitute}(i)$  includes as a subgraph a copy of  $N_{\text{unstable}}$ , the inherently unstable network of Figure 6, as shown in Figure 11.

**Lemma A.1.** *No stable configuration,  $\mathbb{C}$ , may contain both  $(I_x(i), A_y(i))$  and  $(A_y(i), B_y(i))$  (where  $x, y \in \{1, 2, 3\}$ ).*

*Proof.* By contradiction. The edge  $(B_y(i), d(1))$  *must* be present in  $\mathbb{C}$ , or else  $B_y(i)$  would have no path to  $d(1)$ . The cost of the path  $I_x(i) A_y(i) B_y(i) d(1)$  is  $2\$ + 1$ . Now consider the path  $\mathbb{C}(d(i))$ . It is easy to see that the weight of this path must be  $< 2\$$  just by observing that the only way for a path to have weight  $\geq 2\$$  is by using both  $(A_z(j), B_z(j))$  and  $(B_z(j), d(1))$  (for some  $z \in \{1, 2, 3\}, j \in V$ ), and it is not possible for  $\mathbb{C}(d(i))$  to use any  $(A_z(j), B_z(j))$ . But this means  $I_x(i)$  would prefer the path  $I_x(i) \mathbb{C}(d(i))$ , contradicting the stability of  $\mathbb{C}$ .  $\square$

**Corollary A.2.** *Any stable configuration,  $\mathbb{C}$ , must contain exactly one of  $\{(A_1(i), d(i)), (A_2(i), d(i)), (A_3, d(i))\}$ .*

*Proof.* If any two of these edges are in the configuration (w.l.o.g.  $A_1(i)$  and  $A_2(i)$ ), then the subgraph consisting of

$$S = \{L_x(i) \mid x \in \{1 \dots 6\}\} \cup \{I_x(i) \mid x \in \{1, 2, 3\}\} \cup \{d(i)\}$$

is equivalent to the inherently unstable graph shown in Figure 10. If none of the three edges is in  $\mathbb{C}$ , then  $S$  is equivalent to the inherently unstable one shown in Figure 6. When exactly one of the three edges is in  $\mathbb{C}$ , then  $S$  is equivalent to one of the stable networks shown in Figures 7, 8, and 9.  $\square$

### Construction of $N(G)$

$N(G)$  is constructed by assembling together  $\text{NodeSubstitute}(1), \text{NodeSubstitute}(2), \dots, \text{NodeSubstitute}(n)$ , adding the node  $d(1')$ , and adding all the edges as specified in points 1, 2 and 3 above.

We need to designate the kind of all the nodes in  $\{d(1), d(1'), d(2), \dots, d(n)\}$ , which we take to be “round”. The weight assignments according to the 2 metrics are those shown in Figure 11. This completes the construction of the network  $N(G)$ . It remains to show that:  *$G$  has a Hamiltonian cycle iff  $N(G)$  has a stable configuration.*

### Proof of: If $G$ has a Hamiltonian cycle then $N(G)$ has a stable configuration

Suppose  $G$  has a Hamiltonian cycle. Then, by the construction of  $G'$ , we also have that  $G'$  has a Hamiltonian path  $P$  from node  $1'$  to node 1, from which we show how to construct a stable configuration  $\mathbb{C}$  of  $N(G)$ . Path  $P$  can be specified by:

$$\begin{aligned} P &= \ell_1 \ell_2 \dots \ell_n \ell_{n+1} \quad \text{where} \\ \ell_1 &= 1', \ell_{n+1} = 1, \{\ell_2, \dots, \ell_n\} = \{2, 3, \dots, n\} \quad \text{and} \\ (\ell_{m-1}, \ell_m) &\in E' \quad \text{for every } 1 < m \leq n+1. \end{aligned}$$

Consider an arbitrary edge  $(\ell, \ell') \in \{(\ell_1, \ell_2), (\ell_2, \ell_3), \dots, (\ell_n, \ell_{n+1})\}$ . The desired configuration  $\mathbb{C}$  will include an appropriately defined spanning forest, call it  $\mathbb{C}_{\ell'}$ , that covers the nodes of  $\text{NodeSubstitute}(\ell')$ . Since

$(\ell, \ell')$  is an edge of  $G'$ , there must be an edge from  $d(\ell)$  to  $\text{NodeSubstitute}(\ell')$ . With no loss of generality suppose this edge enters  $\text{NodeSubstitute}(\ell')$  at node  $B_1(\ell')$ . (The case when the edge from  $d(\ell)$  enters  $\text{NodeSubstitute}(\ell')$  at node  $B_2(\ell')$  or node  $B_3(\ell')$  is treated similarly.) The case when  $(\ell, \ell') = (p, i)$  is illustrated in Figure 12. The spanning forest  $\mathbb{C}_i$ , with  $i \in \{\ell_2, \dots, \ell_{n+1}\}$ , consists of 7 paths which cover all the nodes of  $\text{NodeSubstitute}(i)$ :

$d(p) B_1(i) A_1(i) d(i)$	corresponds to edge $(p, i)$ of $P$ ,
$L_1(i) I_1(i) d(i)$	covers the nodes $L_1(i)$ , and $I_1(i)$ ,
$I_2(i) A_1(i) d(i)$	covers the node $I_2(i)$ ,
$I_3(i) A_1(i) d(i)$	covers the node $I_3(i)$ ,
$L_2(i) L_3(i) L_4(i) L_5(i) I_3(i) A_1 d(i)$	covers the nodes $L_2(i)$ , $L_3(i)$ , $L_4(i)$ , $L_5(i)$ and $I_3(i)$ ,
$L_6(i) I_3(i) A_1 d(i)$	covers the node $L_6(i)$ ,
$A_2(i) B_2(i) d(1)$	connects “unused” nodes $A_2(i)$ and $B_2(i)$ upward to $d(1)$ ,
$A_3(i) B_3(i) d(1)$	connects “unused” nodes $A_3(i)$ and $B_3(i)$ upward to $d(1)$ .

$\mathbb{C}_i$  is shown in Figure 12. The desired configuration  $\mathbb{C}$  is obtained by collecting together  $\mathbb{C}_{\ell_2}, \mathbb{C}_{\ell_3}, \dots, \mathbb{C}_{\ell_{n+1}}$ . It is straightforward to check that  $\mathbb{C}$  is stable, using the weights shown in Figure 11.

**Proof of: If  $N(G)$  has a stable configuration then  $G$  has a Hamiltonian cycle**

Suppose  $N(G)$  has a stable configuration  $\mathbb{C}$ , from which we shall construct a Hamiltonian path  $P$  from  $1'$  to  $1$  in  $G'$ . The latter will imply that  $G$  has a Hamiltonian cycle, the desired conclusion.

Because  $\mathbb{C}$  is a configuration, i.e. a spanning tree with all paths directed from the leaf nodes to the root node  $d(1)$ , we must have:

1. For every  $i \in \{1', 2, 3, \dots, n\}$ ,  $\mathbb{C}$  contains exactly one of the edges outgoing from  $d(i)$ , because in a spanning tree, every node other than the root has exactly one outgoing edge. This edge connects  $d(i)$  to one of the 3 entry nodes of  $\text{NodeSubstitute}(j)$  for some  $j \in \{1, 2, \dots, n\}$ .
2. For every  $i \in \{1, 2, \dots, n\}$ , because  $\mathbb{C}$  is also stable,  $\mathbb{C}$  must contain exactly one of the 3 following edges:  $(A_1(i), d(i))$ ,  $(A_2(i), d(i))$ , or  $(A_3(i), d(i))$  – if it contains 0, 2 or all 3, of these edges,  $\mathbb{C}$  is unstable, from the observations at the end of the construction of  $\text{NodeSubstitute}(i)$ .

Consider a path ending at  $d(1)$  whose last edge is in  $\{(A_1(i), d(1)), (A_2(i), d(1)), (A_3(i), d(1))\}$ . Such a path may not contain any upward edges (edges  $(B_x(i), d(1))$  for some  $x \in \{1, 2, 3\}$ ) or backward edges (edges  $(A_x(i), B_x(i))$  for some  $x \in \{1, 2, 3\}$ ) and therefore the path must have weight  $< \$$ . Whenever such a path contains an edge  $(A_x(i), d(i))$ , it must also contain  $(B_x(i), A_x(i))$  because  $W_{\circ}(B_x(i), d(1)) = \$$ . Now consider a path is maximal for all paths which contain  $(A_x(i), d(i))$  for every  $d(i) \neq d(1')$  in the path. By the second observation above, there is exactly one such path. This path cannot be cyclic, since  $\mathbb{C}$  is a spanning tree. Whenever the path contains an edge starting from  $d(i)$  is must also contain a path from  $B_x(i)$  to  $d(i)$  (for some  $x \in \{1, 2, 3\}$ ), therefore the path must start from  $d(1')$ , so it is  $\mathbb{C}(d(1'))$ . If this path contains  $d(i)$  for every  $i \in V$ , then it is easy to extract from it a Hamiltonian path from  $1'$  to  $1$  in  $G'$  and therefore a Hamiltonian cycle in  $G$ .

In fact this path must contain  $d(i)$  for every  $i \in V$ . Suppose the contrary, and we will get a contradiction. Consider therefore some node  $d(k)$  not occurring in the path  $\mathbb{C}(d(1'))$ . Node  $d(k)$  is the forward

exit of  $\text{NodeSubstitute}(k)$ . Because  $\mathbb{C}$  is stable, it must be that exactly one of the following 3 edges of  $\text{NodeSubstitute}(k)$  is part of  $\mathbb{C}$ :

$$(A_1(k), d(k)), (A_2(k), d(k)), (A_3(k), d(k))$$

With no loss of generality, assume it is the first of these 3 edges, i.e.  $(A_1(k), d(k))$  is in  $\mathbb{C}$  while both  $(A_2(k), d(k))$  and  $(A_3(k), d(k))$  are not. This in turn implies that the edge  $(B_1(k), A_1(k))$  is in  $\mathbb{C}$  too, otherwise  $(B_1(k), d(1))$  would instead be in  $\mathbb{C}$  and  $A_1(k)$  would want to reach  $d(1)$  via the path “ $A_1(k) B_1(k) d(1)$ ”, contrary to assumption. Thus  $\mathbb{C}(B_1(k))$  is of the form:

$$\mathbb{C}(B_1(k)) = B_1(k) A_1(k) d(k) P$$

for a path  $P$  connecting  $d(k)$  to  $d(1)$ . Consider 2 possible cases for  $P$ , each causing a contradiction:

- (a)  $P$  consists of forward edges only. This is not possible, because the only forward edge to  $d(1)$  was already accounted for in  $\mathbb{C}(d(1))$ .
- (c)  $P$  includes an upward edge. Every upward edge enters  $d(1)$ , which implies  $P = Q X d(1)$  for some path  $Q$  (a possibly empty sequence of nodes) and node  $X$  in the set:

$$\{B_1(i) \mid 1 \leq i \leq n\} \cup \{B_2(i) \mid 1 \leq i \leq n\} \cup \{B_3(i) \mid 1 \leq i \leq n\}$$

In all of these cases, we have

$$\begin{aligned} \text{weight}_o(\mathbb{C}(B_1(k))) &= 1 + 1 + \text{weight}_o(d(k) Q X) + \$ \\ &> \$ \\ &= w_o(B_1(k), d(1)) \end{aligned}$$

Contradicting the stability of  $\mathbb{C}$ .

Hence  $\mathbb{C}(d(1'))$  mentions every node in  $\{d(1'), d(1), d(2), \dots, d(n)\}$ . This concludes the proof.

### A.3 Proof of Theorem 5.3

The proof of the NP-completeness of  $\langle \max, + \rangle$ -SPP is a minor variation of that proof for  $\langle +, + \rangle$ -SPP. First of all, we need to construct the appropriate gadget which is unstable unless exactly two of three equally weighted, circle originated edges are shortened. This is indeed possible, as we shall shortly see.

In the  $\text{NodeSubstitute}(i)$  gadget used for the  $\langle +, + \rangle$ -SPP proof, most of the argument relied only on arguments about the weight of paths according to the round metric. The one property that we need for the square metric is that (for each  $x \in \{1, 2, 3\}$ )  $A_x(i) B_x(i) d(1)$  is preferred by square to any path starting with  $A_x(i) d(i)$ . In order to preserve this property when the square nodes use  $\max$  as an aggregation function, we simply increase the weight of  $A_x(i) d(i)$  to  $2/2$ .

Because we increase the weight of this edge, we need to find a  $\max/+$  plus gadget where stability occurs when two out of three edges are reduced to  $3/3$ . Figures 13, 14, 15, 16, and 17 show such a gadget.

The full  $\text{NodeSubstitute}(i)$  gadget for  $\langle \max, + \rangle$ -SPP is shown in Figure 18. It is straightforward to check that the proof given for  $\langle +, + \rangle$ -SPP works for  $\langle \max, + \rangle$ -SPP given these modifications.

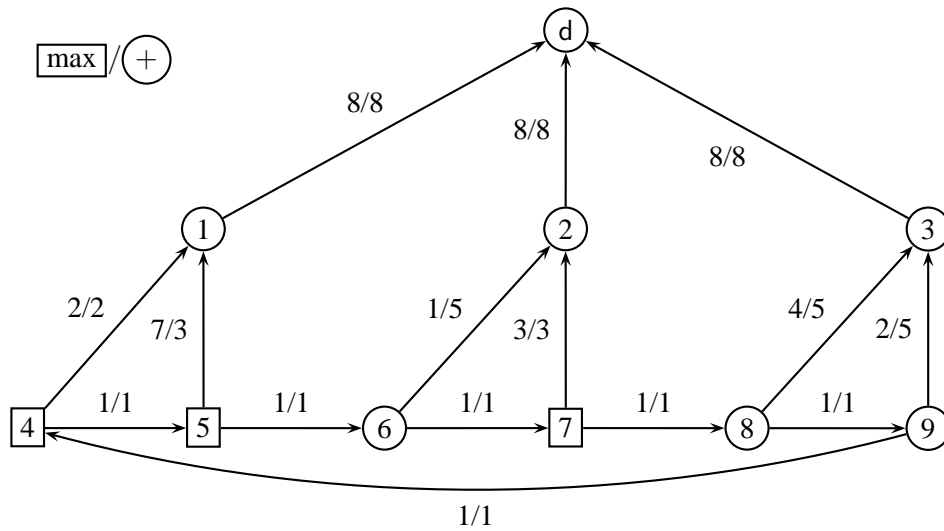


Figure 13:  $\langle \max, + \rangle$ -SPP network with no stable configuration.

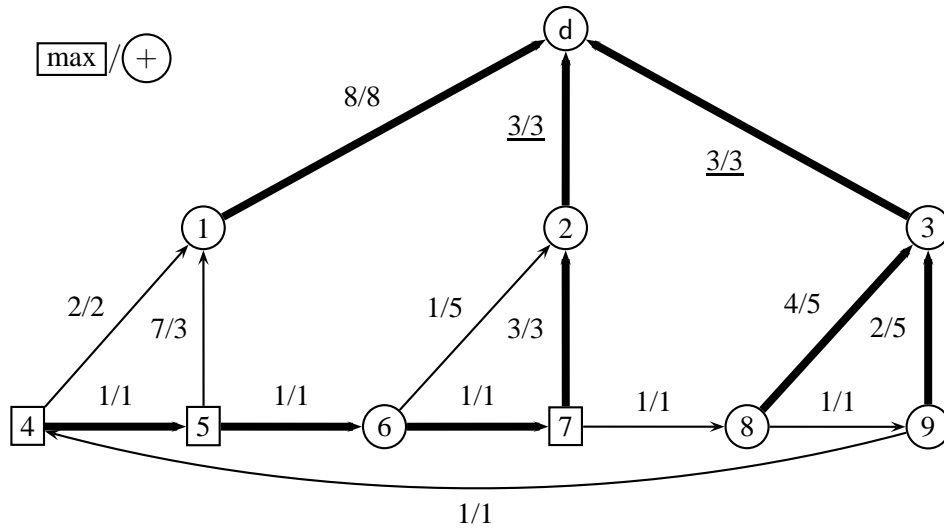


Figure 14:  $\langle \max, + \rangle$ -SPP network with stable configuration.



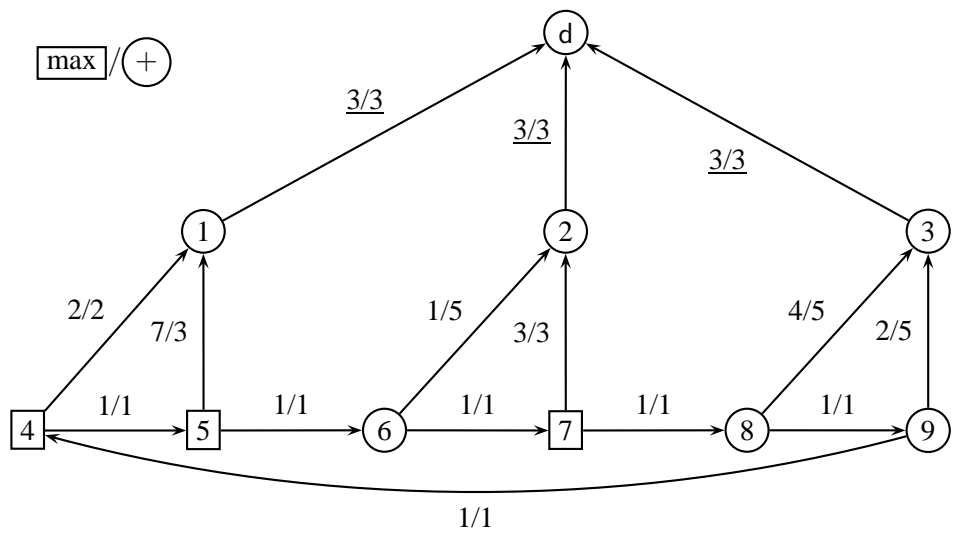


Figure 17:  $\langle \text{max}, + \rangle$ -SPP network with no stable configuration.

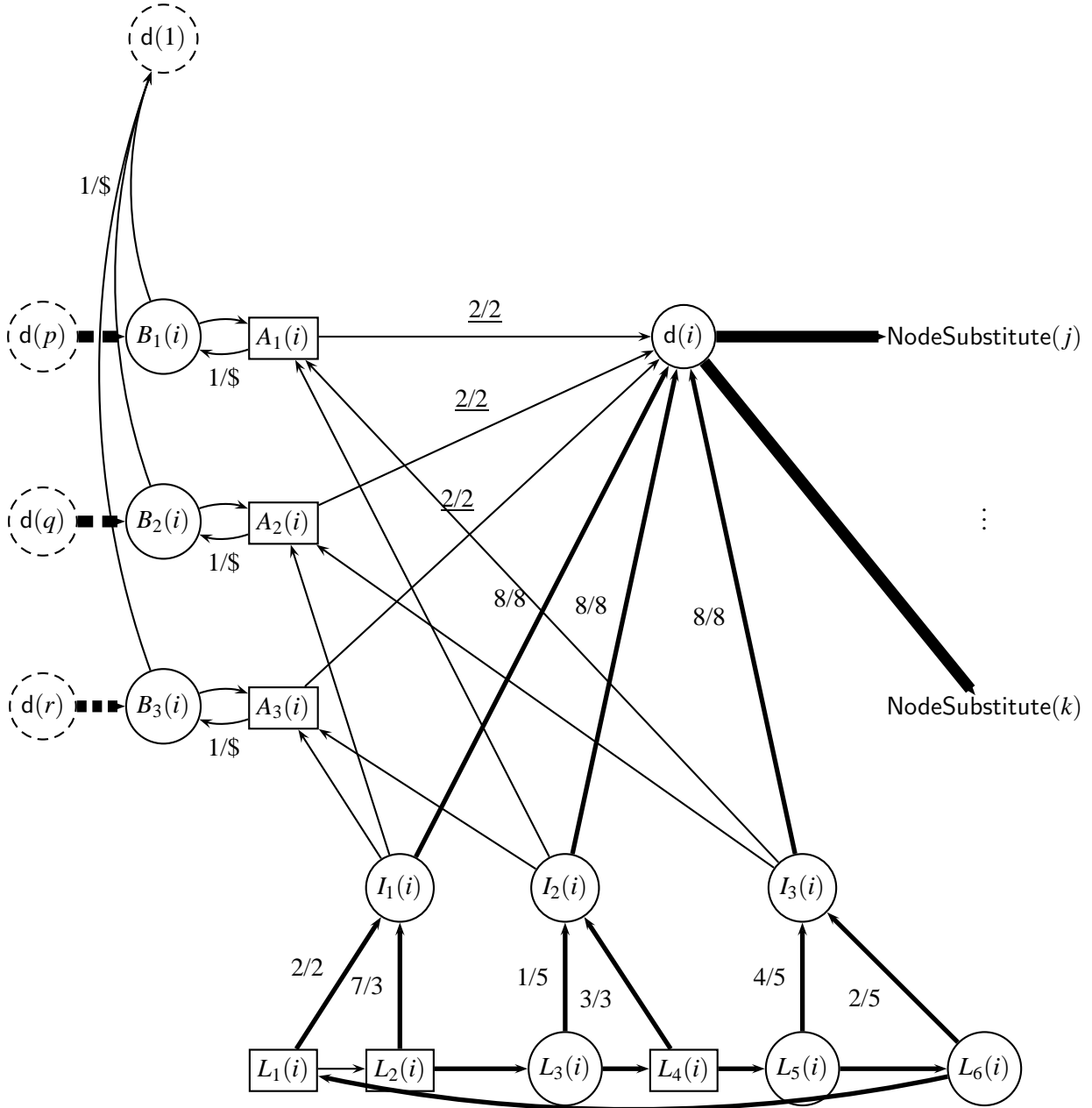


Figure 18:  $\text{NodeSubstitute}(i)$  replaces node  $i$  in the construction of network  $N(G)$  from graph  $G$ .