NOVA LINCS Technical Report
September 2015

The GENERALIZED MIN-CUT Problem

Konstantinos Kloudas[‡1], Margarida Mamede[†],
Nuno Preguiça[†], Rodrigo Rodrigues[‡1]

[†]NOVA LINCS, DI, FCT, Universidade NOVA de Lisboa
[‡]INESC-ID / IST, University of Lisbon

[1]Work done while the author was at NOVA LINCS

The GENERALIZED MIN-CUT Problem

Konstantinos Kloudas[†2], Margarida Mamede[1], Nuno Preguiça[1] and Rodrigo Rodrigues[†2]

[1]NOVA LINCS, DI, FCT, Universidade NOVA de Lisboa
[2]INESC-ID / IST, University of Lisbon

1 Background

This section presents the background necessary to understand the rest of the paper. We start by presenting the basic notions from network theory, which are needed for the presentation of the classic MIN-CUT problem, and the EDMONDS-KARP algorithm that solves it. We focus on the MIN-CUT problem as this is the problem the closest to ours, and our algorithm is inspired by the EDMONDS-KARP one. For a more formal definition of the above notions, the reader can refer to [1].

1.1 Flow Networks and Flows

A **flow network** is a graph $G(V,E)$ seen as a network of water pipes, each with a specific **capacity**, $c(i,j) \geq 0$. The capacity stands for the maximum amount of water that can be sent through pipe $e(i,j) \in E$. Pipes are considered *bidirectional*, *i.e.,* water can flow both ways, with one direction *canceling* the other. The later corresponds to the edges in G being, either *undirected*, or *directed* with two edges $e(i,j), e(j,i) \in E$ of equal capacity, and such that if k units of flow are sent from i to j, while $m < k$ units are sent from j to i, then the flow along $e(i,j)$, with direction from i to j, has value $f(i,j) = k - m$. In addition, a flow network has two special nodes, the **source** s, where water is being pumped from, and the **sink** t, where water is being pumped to.

As in real water pipes, the total water flowing along an edge $e(i,j) \in E$ cannot exceed its capacity, $c(i,j)$, and the total flow entering a node has to be equal to the one exiting it. The only exceptions to the later are nodes s and t. More formally, we extend the edge capacity function to every $(i,j) \in V \times V$, by defining $c(i,j) = 0$ whenever $e(i,j) \notin E$, and adopt the definition of flow in [2]:

Definition 1 (Flow) *A* flow *in $G(V,E)$ is a real-valued function, $f : V \times V \to \mathbb{R}$, with the additional constraints that:*

- *Capacity Constraint:* $-c(i,j) \leq f(i,j) \leq c(i,j)$, $\forall i,j \in V$;
- *Symmetry Constraint:* $f(i,j) = -f(j,i)$, $\forall i,j \in V$;
- *Flow Conservation:* $\sum_{j \in V} f(i,j) = 0$, $\forall i \in V - \{s,t\}$.

The value *of the flow is:* $|f| = \sum_{j \in V} f(s,j)$.

[†]Work done while the author was at NOVA LINCS

In the **MAX-FLOW problem**, we are given a flow network $G(V,E)$ with source s and sink t and we want to find a flow whose value is maximum, that is, where the maximum amount of flow is sent from s to t. The MAX-FLOW problem is solved by the EDMONDS-KARP algorithm, presented below. As shown by Theorem 2, finding the maximum flow from s to t is equivalent to finding the edges with the minimum total capacity that, when removed, they separate s from t. The latter corresponds to the classic MIN-CUT problem.

1.2 The Edmonds-Karp Algorithm

Before diving into the EDMONDS-KARP algorithm for solving the MAX-FLOW and the MIN-CUT problems, we present some necessary notions, namely: *residual networks* and *augmenting paths*.

Intuitively, the **residual network** $G_f(V,E_f)$ of a network $G(V,E)$ with flow f is the graph itself with the capacities and edges transformed to represent how much more flow can be sent on each edge. Due to the capacity constraint, given a network G and a flow f, an edge $e(i,j)$ of G cannot receive more than

$$c_f(i,j) = c(i,j) - f(i,j)$$

additional flow, where $c_f(i,j)$ is the **residual capacity** of $e(i,j) \in E$. In addition, as said earlier, pipes are bidirectional and an algorithm that tries to maximize the *total* flow from s to t may need to decrease the flow on a particular edge. The latter is called *flow cancelation*. To integrate this possibility, for each edge $e(i,j) \in E$, when k units of flow are sent from i to j, the value of $f(i,j)$ increases by k and the value of the flow passing through the reverse edge, $f(j,i)$, decreases by an equal amount. Flow cancelation is illustrated in the example in Figure 1.

Given the above, the definition of residual network $G_f(V,E_f)$ follows.

Definition 2 (Residual Network) *Given a flow network $G(V,E)$ and a flow f, the* residual network G_f *is:*

$$G_f = (V,E_f), \text{ where } E_f = \{e(i,j) \in E \mid c_f(i,j) > 0\}.$$

In G_f, an augmenting path is a simple path p from s to t. That is, p is a path in G that has spare capacity to receive additional flow. A path is *simple* if it contains each node at most once. The above definition implies that paths with cycles, *e.g.,* $v \to w \to v$, are not simple, thus not augmenting paths.

Definition 3 (Augmenting Path (AP)) *Given a flow network $G(V,E)$ with source s and sink t, and a flow f, an* augmenting path *p is a simple path from s to t in the residual network G_f. The* residual capacity *of p is given by:*

$$c_f(p) = min\{c_f(i,j) \mid e(i,j) \in p\}.$$

It follows from the definition of residual network that the residual capacity of an augmenting path is always positive.

With the above notions in mind, Algorithm 1 presents the pseudocode for the EDMONDS-KARP algorithm that solves the MAX-FLOW problem. The algorithm starts by initializing the flow (assuming that there are two directed edges, $e(i,j)$ and $e(j,i)$, in E for every pipe in G). Then, at each iteration, a new augmenting path p is discovered

Algorithm 1: The EDMONDS-KARP Algorithm.

1 $f(i,j) \leftarrow 0$ for every edge $e(i,j) \in E$;
2 **while** $(p \leftarrow \texttt{findPath}(G_f, s, t)) \neq \emptyset$ **do**
3 $\quad c_f(p) \leftarrow \min\{c_f(i,j) \mid e(i,j) \in p\}$;
4 $\quad$ **for** $e(i,j) \in p$ **do**
5 $\quad\quad f(i,j) \leftarrow f(i,j) + c_f(p)$;
6 $\quad\quad f(j,i) \leftarrow f(j,i) - c_f(p)$;
7 **return** f;

in the residual network G_f (line 2), its residual capacity is computed (line 3), and the path is saturated by updating the flows of the edges of the path accordingly (lines 5,6). Here we have to note that flows are updated for both the original edge, and its reverse. The detail that differentiates the EDMONDS-KARP algorithm from pre-existing ones is that the exploration for new augmenting paths is done in a *breadth-first* way, *i.e.,* a shortest augmenting path is saturated first. This is done for efficiency, not correctness. In fact, the EDMONDS-KARP algorithm runs in $O(VE^2)$ time. An example execution of the algorithm step-by-step is shown in Figure 1.

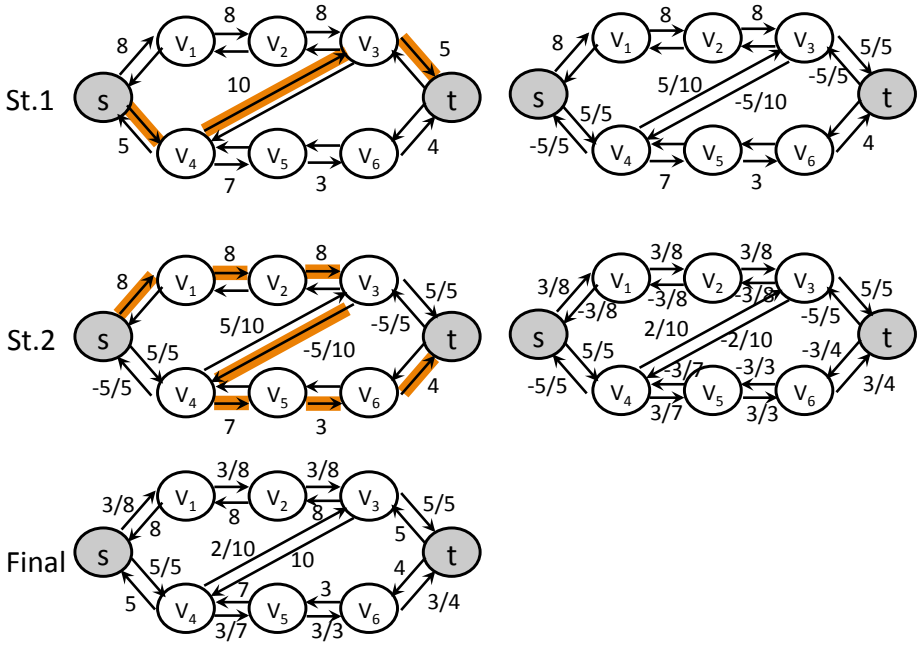

Figure 1: An example execution of the EDMONDS-KARP algorithm. The left-hand side presents the next path to saturate, along with the current flow, f, and capacity, c, of each edge (written f/c, or c when $f = 0$), while the right-hand side shows the flow after the saturation of the path. The only exception is the final step, where no more APs exist and, for clarity, flows of negative value are omitted. The value of the computed flow is 8.

1.3 Cuts

Now we present the definition of cut and some connections between flows and cuts.

Definition 4 (Cut) *A* cut *in the network $G(V,E)$ is a partition of the set of nodes V into two sets (S,T) such that $s \in S$ and $t \in T$.*[1] *The* cut-set *of the cut is the set of edges whose tail is in S and whose head is in T:*

$$\textit{cut-set}(S,T) = \{e(i,j) \in E \mid i \in S, j \in T\}.$$

The cut capacity *is the sum of the capacities of the cut-set edges:*

$$c(S,T) = \sum_{e(i,j) \in \textit{cut-set}(S,T)} c(i,j).$$

In the **MIN-CUT problem**, we are given a network $G(V,E)$ with source s and sink t and we want to find a cut whose capacity is minimum. The two following properties relate flow values to cut capacities.

Theorem 1 *Let G be a flow network, f be any flow in G, and (S,T) be any cut in G. Then, the value of f does not exceed the capacity of (S,T), i.e., $|f| \leq c(S,T)$.*

Theorem 2 (Max-Flow Min-Cut [1]) *If f is a flow in a network G, then the following conditions are equivalent:*

- *f is a maximum flow in G.*
- *The residual network G_f contains no augmenting paths.*
- *$|f| = c(S,T)$ for some cut (S,T) of G.*

Upon termination of Algorithm 1, no more augmenting paths exist. From the above theorems, this means that the value of the computed flow f is maximum and it is equal to the capacity of the minimum cuts. The next theorem shows how to compute a minimum cut from f.

Theorem 3 *Let $G(V,E)$ be a network with source s and sink t, and f be a maximum flow in G. Then, (S,T) is a minimum cut in G, where:*

- *$S = \{v \in V \mid \exists$ a simple path from s to v in the residual network $G_f\}$*
- *$T = V \setminus S$.*

In the example in Figure 1, nodes s, V_1, V_2, V_3, V_4, and V_5 are reachable from the source s in the final residual network. So, the cut $(\{s,V_1,V_2,V_3,V_4,V_5\},\{V_6,t\})$ is minimum and its capacity, which is given by $c(V_3,t)+c(V_5,V_6)$, is 8.

2 MIN-CUT and Data-Parallel Jobs

Although the EDMONDS-KARP algorithm manages to accurately solve the MIN-CUT problem, in this section we show that the problem itself does not fit our settings. The reason can be traced back to a form of *implicit* data replication, which is inherent in data-parallel jobs, and opens up more optimization opportunities, if leveraged. We term this form of replication, *Dataflow Forking*.

Dataflow Forking stands for the case where an operator, v, forwards its (entire) output to more than one downstream operator. Figure 2(a) presents an example, with a

[1] Since (S,T) is a partition of V, $\emptyset \neq S \subseteq V$, $\emptyset \neq T \subseteq V$, $S \cap T = \emptyset$, and $S \cup T = V$.

job with 2 inputs, V_1 and V_2, and one output V_7, where the output of V_3 is consumed by downstream operators V_5 and V_6. We assume that the input of V_2 is stored at the location where the output, V_7, is to be sent, while V_1 is at a different location, and that the weights on the edges correspond to the size of the output of the operator corresponding to each node.

At first sight, partitioning the different operators between the locations of V_1 and V_{2-7} while minimizing the traffic that has to be sent from one to the other, can be seen as a MIN-CUT problem, over the graph presented in Figure 2(b). In the figure, two fake nodes s and t are introduced to represent the source s and the sink t of our flow network. These are the special nodes to separate, and V_1 is connected to s while V_2 and V_7 are connected to t with edges of infinite capacity. This trick guarantees that these edges will not be part of the cut-set of a minimum cut, and that V_1 will belong to set S, while V_2 and V_7 will belong to T. Besides, the graph seems to be undirected, but there are two directed edges for every egde drawn, with the same weight.

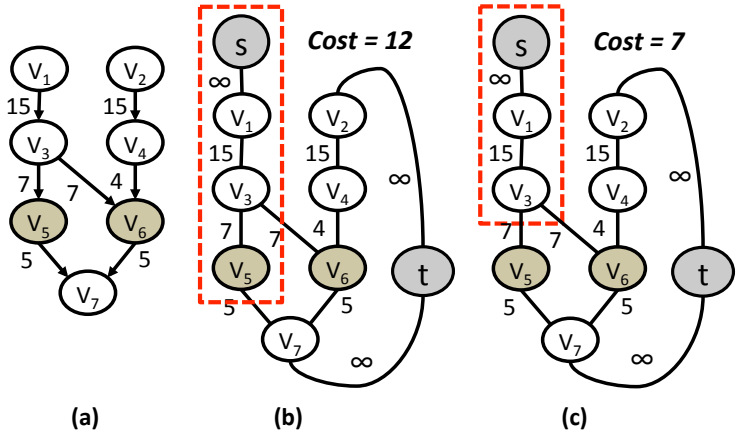

Figure 2: An example where leveraging implicit replication can lead to a cheaper "cut".

Simply solving the MIN-CUT problem in Figure 2(b), without taking into account that $e(V_3,V_5)$ and $e(V_3,V_6)$ refer to the same data would result in the partitioning depicted, with a cost of 12. However, the fact that $e(V_3,V_5)$ and $e(V_3,V_6)$ refer to the same data implies that, if V_5 and V_6 were placed together on a different location than V_3, then one transfer (instead of two), from the location of V_3 to that of V_{5-6}, would be enough. In this case, the weight of the edge $e(V_3,V_5)$ should be counted only once. Taking this into account would allow for the partitioning in Figure 2(c) which has a cost of 7.

From now on, we will refer to a node representing an operator that forwards its output to more than one downstream operator, *e.g.,* V_3, as a *special* node, denoted as σ. In addition, the set of special nodes in a graph $G(V,E)$ will be denoted as $\Sigma \subset V$. Note that the edges between a special node σ and its children, chld(σ), have *all* the same weight, denoted by $w(\sigma)$. That is because the weight of an edge represents the volume of data to be transmitted along that edge and, by definition, the full output of σ is sent to each of its children.

3 Problem Statement

Section 2 showed that the classic MIN-CUT problem statement fails to capture the impact of *Dataflow Forking* while partitioning a job's graph, thus missing potential

optimization opportunities. More specifically, in a graph with a *special* node σ with an output of size $w(\sigma)$ and n children, algorithms for the MIN-CUT problem would account for the cost of its transfer $n \times w(\sigma)$, irrespective of the final placement of the children.

Given that the resulting partition has two sets, S and T, with $s \in S$ and $t \in T$, the placement possibilities for the children of a *special* node, $\sigma \in \Sigma$, are: i) all in S, ii) all in T, and iii) some in S and some in T. Assuming $\sigma \in S$, in case i) none of the edges connecting σ to its children belongs to the cut-set, thus $w(\sigma)$ does not contribute to the cut capacity, while in cases ii) and iii) $w(\sigma)$ should be counted *exactly once*, as in both cases, for all its children that are in T, one cross-set transfer is sufficient.

From the above discussion, if the set of children of σ that *do not* end up in the same set as σ was known in advance, adding an *extra* node, ε_σ, between σ and these children would allow to use classic MIN-CUT algorithms to optimally solve the problem. The new node would be connected to σ and to each of σ's children in the remote set with edges of weight $w(\sigma)$. This transformation would allow classic algorithms to find the minimum cut because they would have the option to cut the edge between σ and ε_σ, thus accounting for the corresponding transfer only once. Given that this information is *not* known in advance, using existing MIN-CUT algorithms, an instance of the MIN-CUT problem could be solved for every possible distribution of the children of σ among two sets. In the case of Figure 2(a), this would be $\{\{V_5, V_6\}, \emptyset\}$ and $\{\{V_5\}, \{V_6\}\}$. For the first distribution, $\{\{V_5, V_6\}, \emptyset\}$, an *extra* node would be added between σ and $\{V_5, V_6\}$, as described previously. For the second distribution, $\{\{V_5\}, \{V_6\}\}$, there would be two *extra* nodes, one between σ and V_5, and the other between σ and V_6. The final solution would be one yielding a cut with the smallest capacity.

It is not difficult to see that the number of instances to solve, I, increases exponentially with: i) the number of children per *special* node, as for a *special* node with n children, there are 2^{n-1} possible distributions among two sets,[2] *and* ii) the number of *special* nodes in the graph, as for a graph with m *special* nodes, each with n_i children (for $i = 1, \ldots, m$), every distribution of the children of *special* node σ_i has to be combined with all possible distributions of the children of each subsequent *special* node σ_j, thus resulting in:

$$I = \prod_{i=1}^{m} 2^{n_i - 1} = 2^{\sum_{i=1}^{m} n_i - 1}. \tag{1}$$

To formalize the above discussion, our problem definition must capture the fact that, in the presence of *Dataflow Forking*, the job's graph gives rise to a *family of graphs*, with an *instance* for each possible distribution of the children of each *special* node among two sets. In fact, we assume that the directed job's graph (like that presented in Figure 2(a)) is first transformed into a symmetric directed graph, called the *original* graph, G^{or}, by adding two new nodes, the source s and the sink t, and all inverted edges (see Figure 2(b) or (c)). An instance of G^{or} is defined as:

Definition 5 (Instance) *An instance of $G^{or}(V^{or}, E^{or})$ is a weighted directed graph $G(V, E)$ obtained from G^{or} in the following way, where $\Sigma \subseteq V^{or}$ is the set of* special *nodes:*

- *All nodes of the original graph belong to the instance: $V^{or} \subseteq V$.*
- *Every edge of the original graph not connecting a* special *node to one of its children belongs to the instance, with the same weight:*

$$\{e(i, j) \in E^{or} \mid (i \in \Sigma \Rightarrow j \notin chld(i)) \wedge (j \in \Sigma \Rightarrow i \notin chld(j))\} \subseteq E.$$

[2] One of the sets can be empty.

- *For every* special *node,* $\sigma \in \Sigma$*, with children* $chld(\sigma)$*, there are one or two new nodes in* V*, called* extra *nodes of* σ*.*
  - *If there is only one* extra *node,* ε_σ*, the new vertex is connected to* σ *and to all* $y \in chld(\sigma)$*:*

$$\{e(\sigma,\varepsilon_\sigma), e(\varepsilon_\sigma,\sigma)\} \cup \{e(\varepsilon_\sigma,y), e(y,\varepsilon_\sigma) \mid y \in chld(\sigma)\} \subseteq E.$$

    In this case, all children of σ *are kept together.*
  - *When there are two* extra *nodes,* ε^1_σ *and* ε^2_σ*, the children of* σ *are distributed through both, according to any partition* $\{Y_1, Y_2\}$ *of* $chld(\sigma)$*. Vertex* ε^k_σ *is connected to* σ *and to every* $y \in Y_k$ *(for* $k = 1,2$*):*
    - $\{e(\sigma,\varepsilon^k_\sigma), e(\varepsilon^k_\sigma,\sigma) \mid k = 1,2\} \subseteq E$*;*
    - $\{e(\varepsilon^k_\sigma,y_k), e(y_k,\varepsilon^k_\sigma) \mid y_k \in Y_k \wedge k = 1,2\} \subseteq E$.

  In both cases, the weights of the new edges are equal to that of the edges in G^{or} *connecting* σ *to its children,* $w(\sigma)$*. The direct successors of an* extra *node except the* special *node are called the* children *of the* extra *node. That is,* $chld(\varepsilon_\sigma) = chld(\sigma)$ *and* $chld(\varepsilon^k_\sigma) = Y_k$.

With the above definition, the *Generalized* MIN-CUT problem can be formulated as follows.

Definition 6 (GENERALIZED MIN-CUT) *Given the original graph* G^{or} *with source* s *and sink* t*, compute a cut of some instance of* G^{or} *that separates* s *from* t *and whose capacity is minimum across all cuts of all instances of* G^{or}.

From the above definition, it follows that the classic MIN-CUT problem is a special case of the GENERALIZED MIN-CUT problem, where the graph has no *special* nodes.

4 Solution

After formulating the new GENERALIZED MIN-CUT problem, this section presents our novel, flow-based *approximation* algorithm for solving it. The main goal of our algorithm is that the weight, $w(\sigma)$, of the edges connecting a *special* node, σ, to each of its children contributes *at most once* to the capacity of the cut computed. The reason was explained in Section 3 and is related to the fact that all those edges refer to the same data.

In a nutshell, our algorithm: i) finds the *optimal* solution when there are no *special* nodes in the graph and in some cases with *special* nodes (*e.g.,* when each σ has 2 children), ii) when it does not, the upper bound of its approximation error is smaller than the error upper bound of classic MIN-CUT algorithms applied on the original graph, and iii) its runtime complexity is $O(VE^2)$, equal to that of the EDMONDS-KARP algorithm.

We start with an overview of the main stages of the algorithm, before presenting the details of each one of them. Finally, an example execution of the algorithm is given in Section 5.

4.1 Overview

An algorithm for the GENERALIZED MIN-CUT problem should achieve two goals. These are:

Algorithm 2: Generalized Min-Cut Algorithm.

1 $G^{init} \leftarrow$ `getInitialInstance(`G^{or}`);`
2 $f_{\text{base}} \leftarrow$ `computeBaseFlow(`G^{init}`);`
3 $G^{can} \leftarrow$ `getCanonicalInstance(`G^{init}, f_{base}`);`
4 $f_{\max} \leftarrow$ `computeMaxFlow(`G^{can}, f_{base}`);`
5 **return** `getMinCut(`G^{can}, $f_{\max}$`);`

1. Find an *instance* of the family of graphs whose minimum cuts are minimum across *all* cuts of *all* instances in the family, and
2. On the selected instance, compute a minimum cut.

The above goals are directly reflected in the general structure of the algorithm, which is presented in Algorithm 2. Lines 1 to 3 aim at constructing a "good" instance of the graph, while lines 4 and 5 take that instance and compute the final cut.

Looking at each step individually, `getInitialInstance` (line 1) builds a first instance of the original graph G^{or}, by adding one *extra* node, ε_σ, per *special* node, $\sigma \in \Sigma$. The new node intercepts the edges between σ and its children. The resulting instance of G is called G^{init}.

After constructing G^{init}, `computeBaseFlow` (line 2) applies our novel BASEFLOW algorithm on it. The purpose of this stage is to *estimate*, for each *special* node σ, which of its children should be kept in the same set as σ, and which should not. BASEFLOW has the same structure as the EDMONDS-KARP algorithm (Algorithm 1), but instead of searching for Augmenting Paths, it searches for Restricted Augmenting Paths (Definition 9).

With the flow computed by `computeBaseFlow`, `getCanonicalInstance` (line 3) constructs a second instance of the original graph, G^{can}, called a canonical instance of G^{or} (Definition 12). The cut computed is a minimum cut of G^{can}, which is obtained by applying the EDMONDS-KARP algorithm (line 4) and the classical minimum cut algorithm (line 5, Theorem 3).

The quality of the final solution depends on the success of the first three stages at locating an instance with a minimum cut across all instances. After these initial steps, the two remaining ones are guaranteed to find a minimum cut of the computed instance.

The above stages are discussed in detail in the remainder of this section, where proofs of their properties are also provided. At the end of this section, Theorem 5 gives an upper bound of the error of our estimation, while Theorem 6 shows that this upper bound is smaller than the one of classic MIN-CUT algorithms applied on G^{or}.

4.2 Initial Instance G^{init}

The initial instance of G^{or}, G^{init}, is created by adding an *extra* node, ε_σ, per *special* node, σ. The new node intercepts the edges connecting σ to its children, whose weights are all $w(\sigma)$. This transformation encodes the option of cutting the edge $e(\sigma, \varepsilon_\sigma)$ (or its reverse), if all $\text{chld}(\sigma)$ are put together in the same set and the cost of transferring the output of σ is counted at most once.

For graphs where all *special* nodes have exactly 2 children (*e.g.,* Figure 2), this transformation would allow any MIN-CUT algorithm to compute an optimal cut, as it gives all options: to cut the edge between σ and ε_σ, if both children of σ are placed in the complement set of σ, or to cut the edge between ε_σ and a child y of σ, if y is the only child that does not belong to the set of σ. Unfortunately, this does not hold in all

cases, as even after the transformation, MIN-CUT algorithms may fail at locating an optimal solution. Such an example is presented in Figure 3(a), where applying a MIN-CUT algorithm on the transformed graph gives a cut of capacity 11 (Figure 3(b)), while a cheaper cut (of another instance) with capacity 6 exists (Figure 3(c)). The following steps allow our algorithm to find an optimal solution in more cases and, even when the solution is not optimal, the upper bound of the approximation error is smaller than that of applying a MIN-CUT algorithm on the original graph.

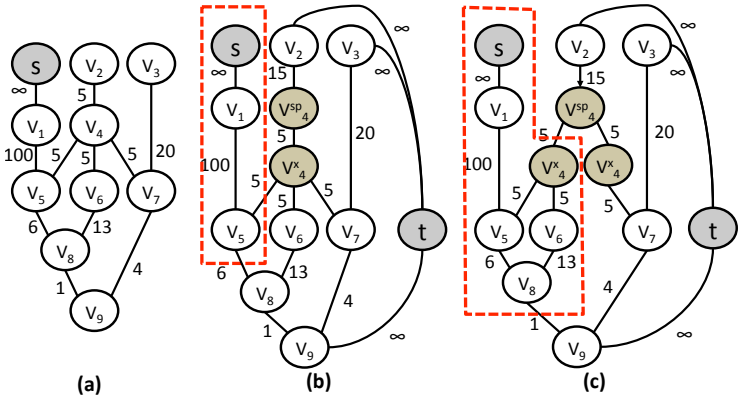

Figure 3: Example where the transformation in G^{init} is not enough to allow MIN-CUT algorithms to find an optimal solution.

4.3 The BASEFLOW Algorithm

After creating G^{init}, `computeBaseFlow` applies our novel BASEFLOW algorithm on it. The ultimate goal of this step is to *estimate*, for each *special* node $\sigma \in \Sigma$, which of its children should be kept in the same set as σ, and which should not. As informally stated in Section 3, this information would allow us to build an instance of G^{or} with an optimal cut. The accuracy of this estimation will determine also the error of the final approximate solution.

The structure of the BASEFLOW algorithm is exactly the same as that of EDMONDS-KARP, presented in Algorithm 1. At each iteration, both algorithms explore the residual network in a *breadth-first* manner, searching for a *valid simple* path from the source, s, to the sink, t, to saturate. The only difference lies in which paths are considered valid by each one of them (and in the definition of residual capacity of a valid path). In EDMONDS-KARP, valid paths are called Augmenting Paths (Definition 3) and they are simple paths in the residual network (that is, paths with positive residual capacity, $c_f > 0$). BASEFLOW, on the other hand, searches for Restricted Augmenting Paths (RAPs) to saturate. RAPs, as their name reveals, are also APs, but with additional constraints that guarantee that the flow returned by BASEFLOW, which is called a **base-flow**, "could be a flow" in any instance of G^{or} after performing a few minor changes.

To achieve that goal, the flow in G^{init} between an *extra* node ε_σ and its children should satisfy one of the following three properties: either there is no flow at all, which happens immediately after the flow initialization, or it goes **down**, from the *extra* node to the children, or it goes **up**, from the children to the *extra* node. In essence, there cannot be two children $y_1, y_2 \in \text{chld}(\varepsilon_\sigma)$ where the flow has opposite directions: it goes down, $f(\varepsilon_\sigma, y_1) > 0$, and it goes up, $f(y_2, \varepsilon_\sigma) > 0$.

Definition 7 (Sets Σ_0, Σ_{down} and Σ_{up}) *Let f be a flow in the initial instance G^{init}, σ be a* special *node and ε_σ be its extra node.*

- *Node σ belongs to $\Sigma_0 \subseteq \Sigma$ if there is no flow in any edge between ε_σ and its children:*

$$\forall y \in chld(\varepsilon_\sigma)\ f(\varepsilon_\sigma, y) = 0.$$

  Notice that, by symmetry and flow conservation (see Definition 1), this implies that $\forall y \in chld(\varepsilon_\sigma)\ f(y, \varepsilon_\sigma) = 0$ and $f(\sigma, \varepsilon_\sigma) = f(\varepsilon_\sigma, \sigma) = 0$.
- *Node σ belongs to $\Sigma_{down} \subseteq \Sigma$ if there is only flow going* down *between ε_σ and its children:*

$$\exists y \in chld(\varepsilon_\sigma)\ f(\varepsilon_\sigma, y) > 0 \ \text{ and } \ \forall y \in chld(\varepsilon_\sigma)\ f(\varepsilon_\sigma, y) \geq 0.$$

  In this case, $f(\sigma, \varepsilon_\sigma) = \sum_{y \in chld(\varepsilon_\sigma)} f(\varepsilon_\sigma, y) > 0$.
- *Node σ belongs to $\Sigma_{up} \subseteq \Sigma$ if there is only flow going* up *between ε_σ and its children:*

$$\exists y \in chld(\varepsilon_\sigma)\ f(y, \varepsilon_\sigma) > 0 \ \text{ and } \ \forall y \in chld(\varepsilon_\sigma)\ f(y, \varepsilon_\sigma) \geq 0.$$

  In this case, $f(\varepsilon_\sigma, \sigma) = \sum_{y \in chld(\varepsilon_\sigma)} f(y, \varepsilon_\sigma) > 0$.

Notice that sets Σ_0, Σ_{down} and Σ_{up} are pairwise disjoint because the flow between the *special* node and its *extra* node satisfies the following (incompatible) properties: if $\sigma \in \Sigma_0$, then $f(\sigma, \varepsilon_\sigma) = 0$; if $\sigma \in \Sigma_{\text{down}}$, then $f(\sigma, \varepsilon_\sigma) > 0$; and if $\sigma \in \Sigma_{\text{up}}$, then $f(\sigma, \varepsilon_\sigma) < 0$.

A RAP is a *restricted path* from the source to the sink, and a restricted path is a simple path in the residual network that keeps the invariant that flows between any *extra* node and its children have the same "up-down" direction.

Definition 8 (Restricted Path (RP)) *Let f be a flow in the initial instance $G^{init}(V^{init}, E^{init})$, and $v \in V^{init}$. A* restricted path *p from s to v in the residual network G_f^{init} is a simple path from s to v in G_f^{init} that imposes the following constraints, for every extra node ε_σ in p:*

*(a) **[down section]** If $\sigma \to \varepsilon_\sigma \to y$ belongs to p (and $y \in chld(\varepsilon_\sigma)$), then:*
either $\sigma \in \Sigma_{down} \cup \Sigma_0$ or $f(y, \varepsilon_\sigma) > 0$.
In the second case, if $f(y, \varepsilon_\sigma) < f(\varepsilon_\sigma, \sigma)$, then the residual capacity of edge $e(\varepsilon_\sigma, y)$ in p and G_f^{init} is $c_f(\varepsilon_\sigma, y) = f(y, \varepsilon_\sigma)$.

*(b) **[up section]** If $y \to \varepsilon_\sigma \to \sigma$ belongs to p (and $y \in chld(\varepsilon_\sigma)$), then:*
either $\sigma \in \Sigma_{up} \cup \Sigma_0$ or $f(\varepsilon_\sigma, y) > 0$.
In the second case, if $f(\sigma, \varepsilon_\sigma) > f(\varepsilon_\sigma, y)$, the residual capacity of edge $e(y, \varepsilon_\sigma)$ in p and G_f^{init} is $c_f(y, \varepsilon_\sigma) = f(\varepsilon_\sigma, y)$.

*(c) **[up-down section]** If $y_1 \to \varepsilon_\sigma \to y_2$ belongs to p and $y_1, y_2 \in chld(\varepsilon_\sigma)$, then:*
either $f(\varepsilon_\sigma, y_1) > 0$ or $f(y_2, \varepsilon_\sigma) > 0$.
In the first case, the residual capacity of edge $e(y_1, \varepsilon_\sigma)$ in p and G_f^{init} is $c_f(y_1, \varepsilon_\sigma) = f(\varepsilon_\sigma, y_1)$; in the second case, the residual capacity of edge $e(\varepsilon_\sigma, y_2)$ in p and G_f^{init} is $c_f(\varepsilon_\sigma, y_2) = f(y_2, \varepsilon_\sigma)$.

The residual capacity of p is the minimum of the residual capacities of the edges in p:

$$c_f(p) = min\{c_f(i, j) \mid e(i, j) \in p\},$$

and the residual capacity of edge $e(i, j)$ is

$$c_f(i, j) = c(i, j) - f(i, j),$$

as usual, unless in the cases explicitly defined above.

It is important to see that the definition of RP is not ambiguous and assures that $\Sigma_0 \cup \Sigma_{\text{down}} \cup \Sigma_{\text{up}} = \Sigma$, that is, every *special* node σ belongs to (exactly) one of those three subsets, after each step of the BASEFLOW algorithm. In the begining, when the flow f is initialized, every *special* node belongs to Σ_0. The **down section** constraint does not limit flows going down where $\sigma \in \Sigma_{\text{down}} \cup \Sigma_0$ and, after saturating the corresponding path p, $\sigma \in \Sigma_{\text{down}}$. But p may cancel some flow going up. When $f(y, \varepsilon_\sigma) > 0$, then $\sigma \in \Sigma_{\text{up}}$ and there are two different cases. If $f(y, \varepsilon_\sigma) < f(\varepsilon_\sigma, \sigma)$, there is flow going up from some sibling of y. For this reason, the residual capacity of p cannot exceed $f(y, \varepsilon_\sigma)$ so that, after saturating p, $f(y, \varepsilon_\sigma) \geq 0$, and σ remains in Σ_{up}. Otherwise, $f(y, \varepsilon_\sigma) = f(\varepsilon_\sigma, \sigma)$ and σ may end in any of the subsets: σ remains in Σ_{up} if $c_f(p) < f(y, \varepsilon_\sigma)$; σ moves to Σ_0 if $c_f(p) = f(y, \varepsilon_\sigma)$; and σ moves to Σ_{down} if $c_f(p) > f(y, \varepsilon_\sigma)$. The **up section** constraint is dual to the first one: *special* nodes in $\Sigma_{\text{up}} \cup \Sigma_0$ end in Σ_{up} and, when the second case applies, p can only cancel some flow going down from ε_σ to a child y but cannot invert the direction of the flow between ε_σ and y, unless there is no flow from ε_σ to any other of its children. So, if $f(\sigma, \varepsilon_\sigma) > f(\varepsilon_\sigma, y)$, σ will remain in Σ_{down} and, when $f(\sigma, \varepsilon_\sigma) = f(\varepsilon_\sigma, y)$, σ will be in one of the subsets. The **up-down section** constraint prohibits paths of the form $y_1 \to \varepsilon_\sigma \to y_2$ in *special* nodes of Σ_0. In addition, the composition of the three subsets will remain unchanged. If $f(\varepsilon_\sigma, y_1) > 0$, $\sigma \in \Sigma_{\text{down}}$ and, since $c_f(p) \leq f(\varepsilon_\sigma, y_1)$, after saturating p, $f(\varepsilon_\sigma, y_1) \geq 0$ and $f(\varepsilon_\sigma, y_2) > 0$. Therefore, σ will be in Σ_{down}. Otherwise, $f(y_2, \varepsilon_\sigma) > 0$ and $\sigma \in \Sigma_{\text{up}}$. So, after updating the flow, $f(y_1, \varepsilon_\sigma) > 0$ and $f(y_2, \varepsilon_\sigma) \geq 0$, which implies $\sigma \in \Sigma_{\text{up}}$.

Definition 9 (Restricted Augmenting Path (RAP)) *Let f be a flow in the initial instance G^{init} of G^{or}. A* restricted augmenting path *p is a restricted path from s to t in the residual network G_f^{init}.*

The first thing to notice is that RAPs are APs (in the sense that they are simple paths from s to t in the residual network) with additional constraints. Consequently, the BASEFLOW algorithm computes a flow in G^{init} (Section 1.1). For the sake of conciseness, it will be called a base-flow of G^{or}.

RAPs are crucial for the success of our algorithm, as they preserve the invariant that every *special* node belongs to $\Sigma_0 \cup \Sigma_{\text{down}} \cup \Sigma_{\text{up}}$. Figure 4 illustrates with an example a subtle but important point of Definition 8, the residual capacity of a RAP p, $c_f(p)$, which is the maximum flow that can be sent along p in order to saturate it. Figure 4(a) shows a part of a graph containing a *special* node V_1, in G^{init}, with the *extra* node V_e between V_1 and its children. The capacity of all edges incident to V_e is 10 and initially the flow is 0. Figure 4(b) shows G_f^{init} after saturating RAP $p_1 = s \rightsquigarrow V_1 \to V_e \to V_2 \rightsquigarrow t$ with 4 units of flow. Now, let us consider RAP $p_2 = s \rightsquigarrow V_2 \to V_e \to V_4 \rightsquigarrow t$, whose residual capacity is 4, by Definition 8 **up-down section**, instead of 10, as in the general case. Figures 4(c) and (d) depict the residual network after sending 4 and 10 units of flow along p_2, respectively. In (c), $V_1 \in \Sigma_{\text{down}}$ whereas, in (d), the invariant is violated, because flow goes up from V_2 to V_e and goes down from V_e to V_4.

In the remainder of this section, we will prove Corollary 1 which shows that the value of a base-flow *does not* exceed the capacity of any cut of any instance of G^{or}. Given that the instance we will use for computing the final cut is based on the flow returned by BASEFLOW, this somehow indicates that a base-flow is a step towards the correct solution. Here we have to remind the reader that BASEFLOW is not guaranteed to finish with a maximum flow, as after its termination there may still be APs that are not RAPs.

On the way towards proving Corollary 1, we also define the flow f induced by a base-flow in an instance of G^{or}. The next lemma helps proving that f is a flow.

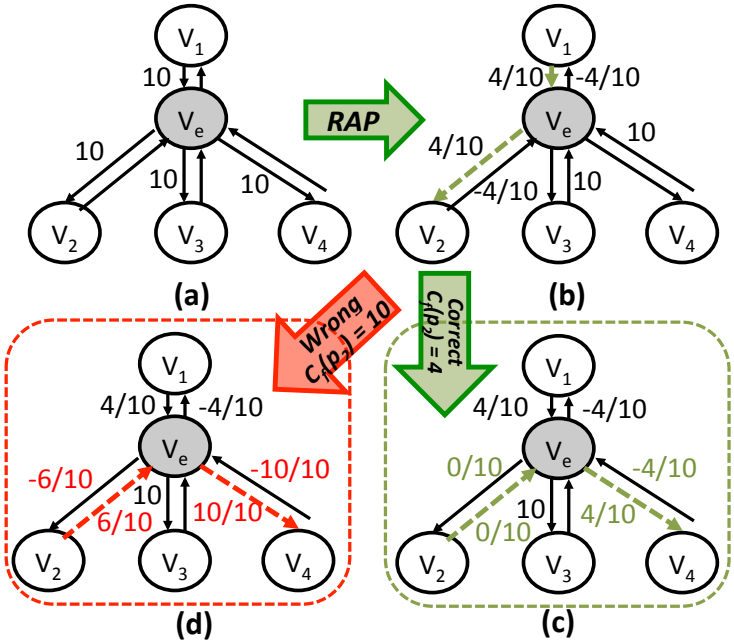

Figure 4: An example of right and wrong residual capacities of RAPs.

Lemma 1 *Let f_{base} be a base-flow. For every special node $\sigma \in \Sigma$:*

- $\sum_{y \in chld(\varepsilon_\sigma)} f_{base}(y, \varepsilon_\sigma) = f_{base}(\varepsilon_\sigma, \sigma) \in [-c(\varepsilon_\sigma, \sigma), c(\varepsilon_\sigma, \sigma)]$.
- $\sum_{y \in chld(\varepsilon_\sigma)} f_{base}(\varepsilon_\sigma, y) = f_{base}(\sigma, \varepsilon_\sigma) \in [-c(\sigma, \varepsilon_\sigma), c(\sigma, \varepsilon_\sigma)]$.

Proof Given that flows between ε_σ and its children have the same direction, flow conservation implies that the total flow entering the *extra* node equals the total flow exiting it. If flows go *up*, then the flow exiting ε_σ is upper bounded by $c(\varepsilon_\sigma, \sigma)$, due to the capacity constraint, while if they go *down*, the flow entering ε_σ is upper bounded by $c(\sigma, \varepsilon_\sigma)$. The lower bounds come from symmetry. For instance, if flows go up, $f_{\text{base}}(\sigma, \varepsilon_\sigma) = -f_{\text{base}}(\varepsilon_\sigma, \sigma) \geq -c(\varepsilon_\sigma, \sigma)$. So $f_{\text{base}}(\sigma, \varepsilon_\sigma) \geq -c(\sigma, \varepsilon_\sigma)$, because $c(\varepsilon_\sigma, \sigma) = c(\sigma, \varepsilon_\sigma) = w(\sigma)$. ∎

Definition 10 (Flow Induced by Base-Flow) *Let $G^{init}(V^{init}, E^{init})$ be the initial instance of G^{or}, $G = (V, E)$ be any instance of G^{or}, and f_{base} be a base-flow. The flow induced by f_{base} in G is the function $f : E \to \mathbb{R}$ defined as follows.*

- $f(v_i, v_j) = f_{base}(v_i, v_j)$
  - *if neither v_i nor v_j is an* extra *node or*
  - *if v_i or v_j is the* only *extra node of some* special *node $\sigma \in \Sigma$.*
- *For every* special *node $\sigma \in \Sigma$ with two* extra *nodes in V, ε^1_σ and ε^2_σ, let ε_σ denote the extra node in V^{init}. In this case, for $k = 1, 2$:*
  - *$f(\varepsilon^k_\sigma, y) = f_{base}(\varepsilon_\sigma, y)$ and $f(y, \varepsilon^k_\sigma) = f_{base}(y, \varepsilon_\sigma)$, for every $y \in chld(\varepsilon^k_\sigma)$;*
  - *$f(\varepsilon^k_\sigma, \sigma) = \sum_{y \in chld(\varepsilon^k_\sigma)} f(y, \varepsilon^k_\sigma)$ and $f(\sigma, \varepsilon^k_\sigma) = \sum_{y \in chld(\varepsilon^k_\sigma)} f(\varepsilon^k_\sigma, y)$.*

Notice that the flow induced by a base-flow f_{base} in the initial instance is f_{base}.

Lemma 2 *Let f_{base} be a base-flow and G be an instance of G^{or}. The flow induced by f_{base} in G is a flow in G whose value is $|f_{base}|$.*

Proof Let f be the flow induced by f_{base} in G. To prove that f is a flow in G, we have to show that it satisfies i) the capacity constraint, ii) symmetry and iii) flow conservation. Symmetry comes directly from the definition of f and f_{base} being a flow. The others follow from Lemma 1 and the fact that $\{\text{chld}(\varepsilon_\sigma^1), \text{chld}(\varepsilon_\sigma^2)\}$ is a partition of $\text{chld}(\varepsilon_\sigma)$. Flow values are equal because all vertices adjacent to the source belong to the original graph. ∎

The main result of this section follows.

Corollary 1 *Let G be any instance of G^{or}. The value of a base-flow does not exceed the capacity of any cut in G.*

Proof This is a direct consequence of Theorem 1 and Lemma 2. ∎

4.4 Canonical Instance

The purpose of computing a base-flow in the previous step is to estimate which of the children of a *special* node σ should be kept in the same set as σ, and which should not. With this information available, `getCanonicalInstance` creates a *canonical instance* of G^{or}, G^{can}, which is the instance on which we are going to compute a minimum cut. In this instance, each special node is connected to one or two *extra* nodes, based on the "natural partition" of G^{or} w.r.t. f_{base} (Definition 11). The *natural partition* separates the nodes reachable from the source in the residual network $G^{init}_{f_{\text{base}}}$, from the unreachable ones, like the partition defined in Theorem 3, but considering restricted paths.

Definition 11 (Natural Partition of G^{or}) *Let G^{init} be the initial instance and f_{base} be a base-flow of $G^{or}(V^{or}, E^{or})$. The* natural partition *of G^{or} w.r.t. f_{base} is the pair (S^{or}, T^{or}), where:*

- $S^{or} = \{v \in V^{or} \mid \exists$ *a restricted path from s to v in the residual network $G^{init}_{f_{base}}$*$\}$
- $T^{or} = V^{or} \setminus S^{or}$.

From the definition, we see that S^{or} contains all nodes v of the original graph for which there is a restricted path from s to v, while T^{or} contains the rest. The two sets are guaranteed to be disjoint, $s \in S^{or}$ because s is a restricted path from s to s, and $t \in T^{or}$ because there is no RAP after termination of the BASEFLOW algorithm.

We are now ready to define the *canonical instance* of G^{or} w.r.t. a base-flow.

Definition 12 (Canonical Instance) *Let f_{base} be a base-flow of G^{or} and (S^{or}, T^{or}) be the natural partition of G^{or} w.r.t. f_{base}. The* canonical instance *G^{can} of G^{or} w.r.t. f_{base} is defined as follows. For every* special *node $\sigma \in \Sigma$:*

- *if $chld(\sigma) \subseteq S^{or}$ or $chld(\sigma) \subseteq T^{or}$, there is only one extra node in G^{can};*
- *otherwise, there are two extra nodes in G^{can}, ε_σ^1 and ε_σ^2, and:*

$$chld(\varepsilon_\sigma^1) = chld(\sigma) \cap S^{or} \;\text{ and }\; chld(\varepsilon_\sigma^2) = chld(\sigma) \cap T^{or}.$$

A canonical instance is an instance of G^{or} because, whenever there are two *extra* nodes of a *special* node, $\{\text{chld}(\varepsilon_\sigma^1), \text{chld}(\varepsilon_\sigma^2)\}$ is obviously a partition of $\text{chld}(\sigma)$. That is, besides none of the sets being empty, we have:

$$\text{chld}(\varepsilon_\sigma^1) \cup \text{chld}(\varepsilon_\sigma^2) = \text{chld}(\sigma) \;\text{ and }\; \text{chld}(\varepsilon_\sigma^1) \cap \text{chld}(\varepsilon_\sigma^2) = \emptyset.$$

4.5 Final Cut and Approximation Error

The final step, after finding the canonical instance of G^{or}, is the computation of a minimum cut in that particular instance. To do this, we apply the classic EDMONDS-KARP algorithm on G^{can}. Here we have to note that we do not re-explore already saturated (restricted) augmenting paths because, instead of initializing the flow with 0, it is initialized with the flow in the canonical instance induced by the base-flow.

The remainder of the section is dedicated to providing upper bounds for the approximation error and showing that the upper bound of our solution is smaller than that of directly applying a MIN-CUT algorithm on G^{or}. In doing so, we define the *natural cut* of G^{can} (Definition 13), which extends the natural partition of G^{or} (Definition 11) to the set of nodes of G^{can}, distributing the *extra* nodes among the partion sets.

Definition 13 (Natural Cut of Canonical Instance) *Let (S^{or}, T^{or}) be the natural partition and $G^{can}(V^{can}, E^{can})$ be the canonical instance of $G^{or}(V^{or}, E^{or})$ w.r.t. the same base-flow f_{base}. The* natural cut *of G^{can} is the pair (S,T) built as follows:*

- *$S^{or} \subseteq S$ and $T^{or} \subseteq T$*
  (nodes shared by all instances of G^{or} belong to the corresponding sets);
- *every* extra *node $\varepsilon \in V^{can} \setminus V^{or}$ belongs to the set of its children:*
  - *if $chld(\varepsilon) \subseteq S^{or}$, then $\varepsilon \in S$;*
  - *if $chld(\varepsilon) \subseteq T^{or}$, then $\varepsilon \in T$.*

Note that, although a cut, the natural cut is *not necessarily* a minimum cut of G^{can}. Its capacity is given by Theorem 4. The proof of Theorem 4 makes use of several results, which basically analyse all types of edges that may belong to the cut-set. We present them in an appendix, for ease of reading.

Theorem 4 *Let f_{base} be a base-flow of G^{or}, G^{can} be the canonical instance of G^{or} w.r.t. f_{base} and (S,T) be its natural cut. Let also P be the set of special nodes $\sigma \in \Sigma$ that satisfy one of the two following conditions:*

(a) $\sigma \in \Sigma_{up} \cup \Sigma_0$, $\sigma \in S$ and σ has two extra nodes in G^{can};

(b) $\sigma \in \Sigma_{down}$, $\sigma \in T$ and σ has two extra nodes in G^{can}.

Then,

$$c(S,T) = |f_{base}| + \sum_{\sigma \in P} w(\sigma)\,,$$

where $w(\sigma)$ is the weight of the edges between σ and its children.

Proof Let f be the flow induced by f_{base} in G^{can} and $e(i,j)$ be an edge of G^{can} such that $i \in S$ and $j \in T$. Then, there are three possible cases for i and j.

1. Neither i nor i is an extra node. From Lemma 4 (a), $f(i,j) = c(i,j)$.
2. $i \in \Sigma$ and j is an *extra* node of i (so $\text{chld}(j) \subseteq T$):
   (a) If $i \in \Sigma_{\text{up}}$, by Lemma 5 (a), it has two *extra* nodes and, by Lemma 3 (b), $f(i,j) = 0$. But $i \in P$ and $w(i)$ belongs to the summation.
   (b) If $i \in \Sigma_0$, by Lemma 7 (a), it has two *extra* nodes and, by Lemma 3 (a), $f(i,j) = 0$. But $i \in P$ and $w(i)$ belongs to the summation.
   (c) If $i \in \Sigma_{\text{down}}$, by Lemma 4 (c), $f(i,j) = c(i,j)$.
3. i is an *extra* node of $j \in \Sigma$ (so $\text{chld}(i) \subseteq S$):
   (a) If $j \in \Sigma_{\text{up}}$, by Lemma 4 (b), $f(i,j) = c(i,j)$.
   (b) $j \notin \Sigma_0$, by Lemma 7 (b).
   (c) If $j \in \Sigma_{\text{down}}$, by Lemma 5 (b), it has two *extra* nodes and, by Lemma 3 (c), $f(i,j) = 0$. But $j \in P$ and $w(j)$ belongs to the summation.

■

The next proposition states some sufficient conditions for the computed cut to be an optimal cut.

Corollary 2 *Let f_{base} be a base-flow of G^{or}, G^{can} be the canonical instance of G^{or} w.r.t. f_{base}, (S,T) be its natural cut, and β be the minimum capacity of a cut of some instance of G^{or}. If no special node $\sigma \in \Sigma$ satisfies one of the following conditions:*
(a) $\sigma \in \Sigma_{up} \cup \Sigma_0$, $\sigma \in S$ and σ has two extra nodes in G^{can};
(b) $\sigma \in \Sigma_{down}$, $\sigma \in T$ and σ has two extra nodes in G^{can},
then $c(S,T) = \beta$.

Proof By Theorem 4, $c(S,T) = |f_{\text{base}}|$. By Corollary 1, $|f_{\text{base}}| \leq \beta$. Finally, $c(S,T)) \not< \beta$ because (S,T) is a cut of an instance of G^{or}. ■

Corollary 2 is especially important as it shows that the computed cut is optimal, if any of the following holds:

- there is no *special* node in the graph;
- the cut-set of the natural cut does not contain any edge between a *special* node with two children and one of its *extra* nodes;
- the residual network of the canonical instance (with respect to the flow induced by f_{base}) has no APs.

Finally, Theorem 5 gives an upper bound of the approximation error of the cut computed by our algorithm, *i.e.,* after computing a *minimum* cut of a canonical instance of G^{or}.

Theorem 5 *Let (S,T) be a minimum cut of a canonical instance of G^{or} and β be the minimum capacity of a cut of some instance of G^{or}. Then,*

$$c(S,T) \leq \beta + \sum_{\sigma \in \Sigma} w(\sigma)\,,$$

where $w(\sigma)$ is the weight of the edges between σ and its children.

Proof
First of all, $c(S,T) \leq c(S',T')$, where (S',T') is the natural cut of the canonical instance. In addition, by Theorem 4, $c(S',T') \leq |f_{\text{base}}| + \sum_{\sigma \in \Sigma} w(\sigma)$, where f_{base} is a base-flow of G^{or}, and $|f_{\text{base}}| \leq \beta$ by Corollary 1. ■

The above shows that the upper bound of the approximation error of our solution is $\sum_{\sigma \in \Sigma} w(\sigma)$. With this in mind, Theorem 6 computes the upper bound of the approximation error of applying a MIN-CUT algorithm on G^{or}.

Theorem 6 *There are original graphs G^{or} with any positive number of* special *nodes that satisfy the following property, where (S,T) is a minimum cut of G^{or}, β is the minimum capacity of a cut of some instance of G^{or}, and $w(\sigma)$ is the weight of the edges between* special *node σ and its children:*

$$c(S,T) = \beta + \sum_{\sigma \in \Sigma} (|chld(\sigma)| - 1) \times w(\sigma)\,.$$

Proof We initially analyse a graph with a single *special* node (see Figure 5(a)), before generalizing the result for a graph with an arbitrary number of *special* nodes (Figure 5(b)). In both figures, the minimum cut is shown with a dashed line.

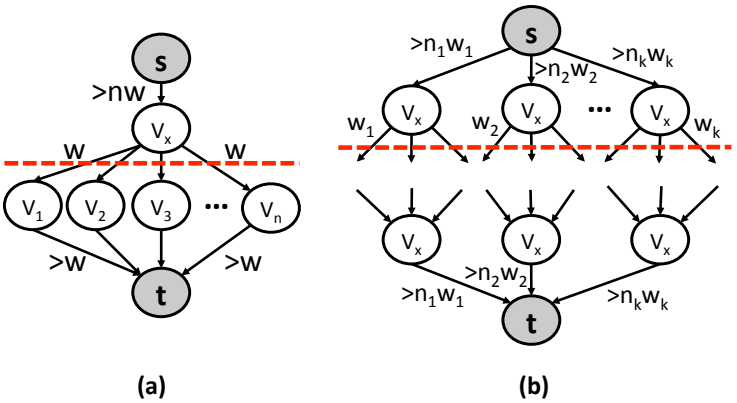

Figure 5: Approximation error of a MIN-CUT algorithm on the GENERALIZED MIN-CUT problem.

In the case of a graph with a single *special* node, V_X, shown in Figure 5(a), the capacity of the minimum cut is equal to $n \times w$, where n is the number of children of V_X, whereas $\beta = w$, as all edges belonging to the cut-set refer to the same data. Notice that β is the capacity of the minimum cut of the initial instance of the graph.

Generalizing this to a graph with multiple *special* nodes (Figure 5(b)), the capacity of the minimum cut in 5(b) is $\sum_{i=0}^{k} n_i \times w_i$, while $\beta = \sum_{i=0}^{k} w_i$, which is the capacity of the minimum cut of the initial instance. ■

Theorem 6 shows that the capacity of the cut computed by a MIN-CUT algorithm can differ by up to $\sum_{\sigma \in \Sigma}(|\text{chld}(\sigma)| - 1) \times w(\sigma)$ from the optimal solution, with the latter being bigger than the upper bound of the error of our algorithm (Theorem 5).

5 Putting it all together

This section presents an example execution of our algorithm for solving the GENERALIZED MIN-CUT problem (see Figure 6). Following the example in Figure 1, in the first two steps, on the left-hand side we present the path to saturate along with the current flow and capacity of each edge (f/c), while on the right-hand side we show the flow after the saturation of the path. Flows of value 0 are omitted.

As we can see from the figure, Step 1 (left) presents the initial instance of the graph, G^{init}, along with the first RAP to saturate. G^{init} has two RAPs, which are saturated in Steps 1 and 2. Notice that the AP:

$$s \to V_1 \to V_5 \to V_8 \to V_6 \to V_4^x \to V_7 \to V_3 \to t$$

is not a RAP because the up-down section $V_6 \to V_4^x \to V_7$ violates Definition 8(c). Since there are no more RAPs, the flow obtained after the second step is the base-flow.

To create the canonical instance G^{can}, in Step 3, we need to find the set of nodes of the original graph that are reachable from s by a restricted path, which is $S^{or} = \{s, V_1, V_5, V_8, V_6\}$. Since V_5 and V_6 belong to S^{or} but V_7 does not, *special* node V_4^{sp} has two *extra* nodes in G^{can}: one, $V_{4,1}^x$ connects to V_5 and V_6 and the other, $V_{4,2}^x$, connects to V_7. The flow in G^{can} induced by the base-flow is straightforwardly computed. In the natural cut of G^{can}, (S,T), set $S = S^{or} \cup \{V_{4,1}^x\}$, because the children of $V_{4,1}^x$ belong to S. Likewise, $T = T^{or} \cup \{V_{4,2}^x\}$.

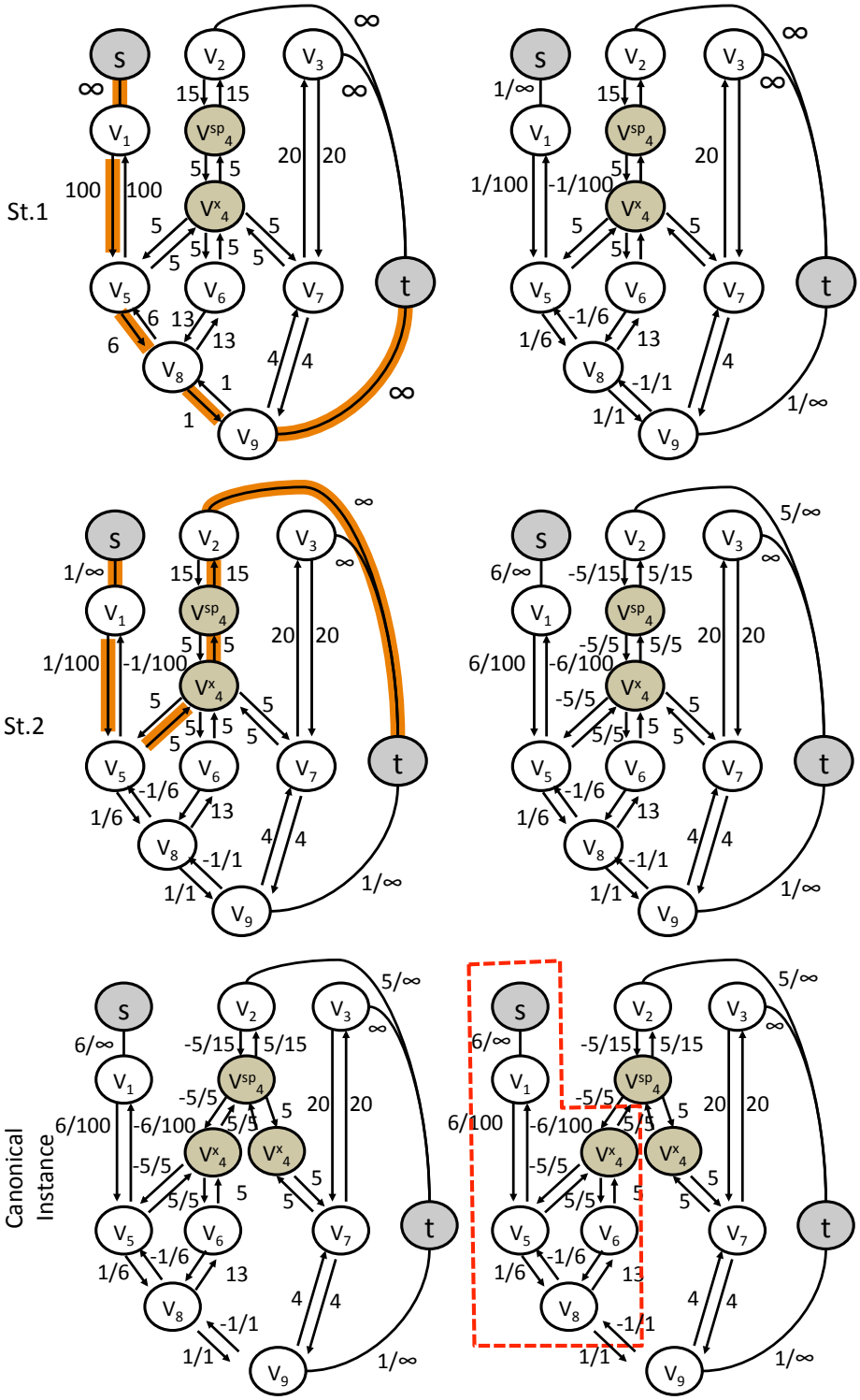

Figure 6: Example execution of our algorithm for solving the GENERALIZED MIN-CUT problem. Steps 1 and 2 saturate the existing RAPs, while Step 3 creates the resulting canonical instance of the graph. Given that there are no APs in the canonical instance, the resulting cut is optimal and has capacity 6.

In this example, since there is no AP in the resulting residual network, the cut computed is the natural cut of G^{can} (depicted in the figure), which is optimal.[3]

To show that, even after the initial transformation from G^{or} to G^{init}, a classic algorithm like EDMONDS-KARP would fail to locate the optimal solution, Figure 7 presents how EDMONDS-KARP would continue after Step 2 from Figure 6. The reason why this would happen is because RAPs impose additional constraints to APs. The resulting cut, presented in Figure 7, has capacity 11.

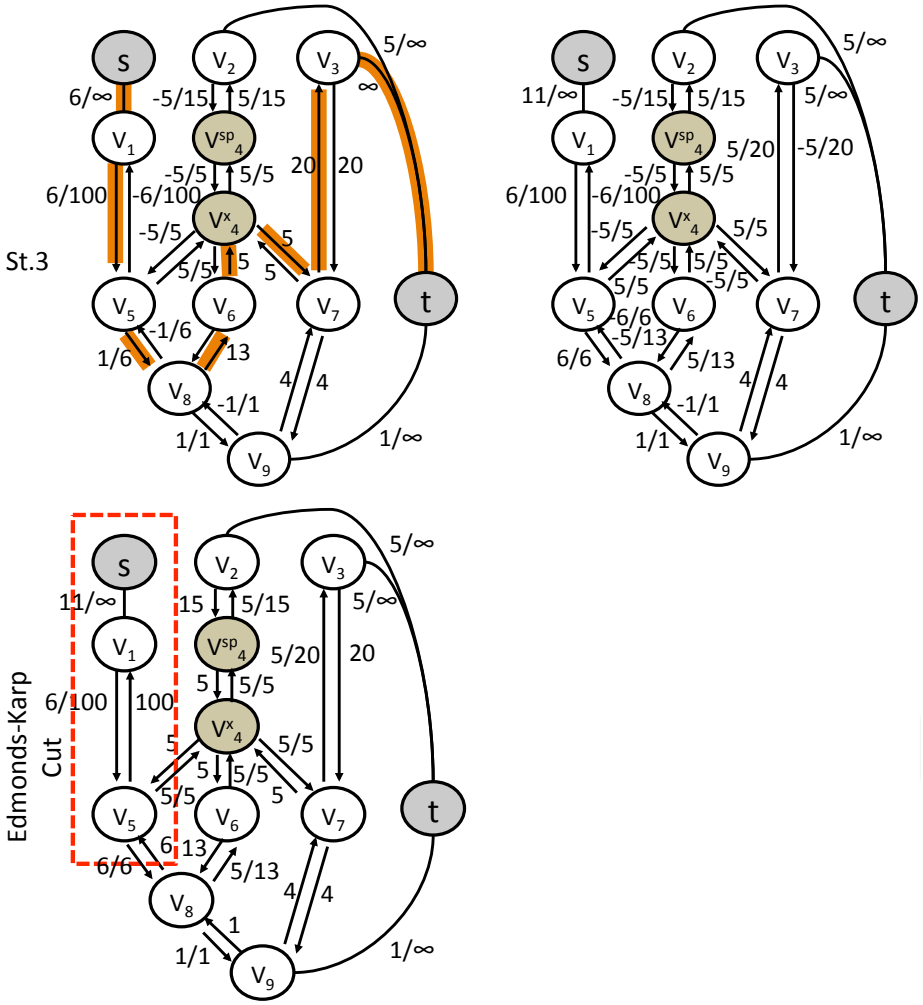

Figure 7: Example where the transformation performed to obtain G^{init} is not enough to allow EDMONDS-KARP to find the optimal solution. Notice that the AP computed in Step 3 is not a RAP. The resulting cut has capacity 11, which is larger than the one in Figure 6.

[3]The natural cut of G^{can} had to be an optimal cut by Corollary 2, because the only *special* node belongs to Σ_{up} and to T'.

6 The GENERALIZED K MIN-CUT Problem.

To solve GENERALIZED K MIN-CUT problem, we rely on the algorithm presented before, and we apply the Isolating Cut heuristic. This heuristic is used to approximate the solution of the classic K MIN-CUT problem, by computing a set of MINIMUM ISOLATING CUTS.

Definition 14 MINIMUM ISOLATING CUT: *A minimum isolating cut for terminal $s_i \in S$ is a minimum weight set of edges $E_i \subseteq E$ whose removal disconnects s_i from all terminals $s_j \in S \setminus \{s_i\}$.*

The Isolating Cuts heuristic computes for every terminal $s_i \in S$, a minimum isolating cut that separates s_i from all the rest. Assuming that there are k terminals to separate, *i.e.,* $|S| = k$, the final solution is the union of the $k-1$ cheapest cuts. This is also the approach taken in our solution.

To find a minimum isolating cut for a terminal s_i, all remaining terminals $s_j \in S - s_i$ are connected to a new vertex, t_i, with edges of infinite weight, and a GENERALIZED MIN-CUT problem is solved for $\{s_i, t_i\}$. The result for terminals $\{s_i, t_i\}$ corresponds to a partition $\{P_i, T_i\}$ of the set of all vertices such that $s_i \in P_i$ and $t_i \in T_i$. The edges with one endpoint in P_i and the other endpoint in T_i form the cut. The final solution is composed of the partitions $P = [P_1, P_2, ..., P_k]$ that contain the terminals $[s_1, s_2, ..., s_k]$.

6.1 Impact of Dataflow Forking

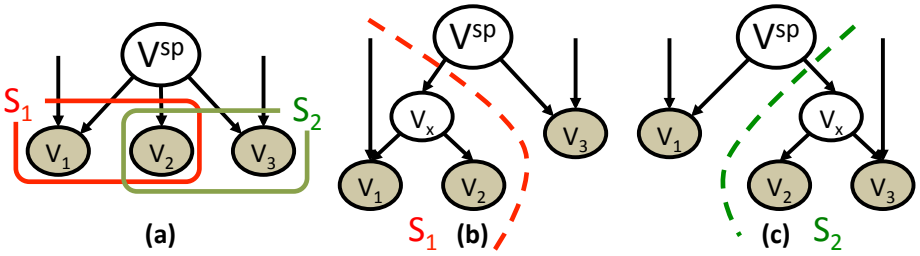

Figure 8: Children of σ partitioned in an overlapping way in different GENERALIZED MIN-CUT instances of the same GENERALIZED K MIN-CUT problem.

As explained earlier, in the face of Dataflow Forking, and for each of these GENERALIZED MIN-CUT problems, the original graph G(V, E)represents a family of graphs (or *instances*), with a member for each combination of different partitions of the children of the graph's *special* nodes, $\sigma \in \Sigma$. To solve each of these problems, our algorithm initially finds the instance (among this set of instances) on which to compute the minimum cut. This instance is termed the Canonical Instance G^{can} of G^{or}. Given that for each GENERALIZED MIN-CUT problem, its G^{can} is computed independently from the ones selected for the remaining problems, this may lead to the chld(σ) being partitioned differently between the two canonical instances. Figure 8 shows an example of a *special* node σ with its children, $v_1, v_2, v_3 \in V$. Figure 8(a) shows how the children are partitioned when computing G^{can} for separating s_1 (red), and s_2 (green) from the remaining terminals. As shown, v_2 is put with v_1 in s_1 and with v_3 in s_2. The canonical instances for the two GENERALIZED MIN-CUT problems are shown in Figure 8(b) and (c), respectively, where an *extra* node is added between σ and the children that are on the same partition. Computing the minimum-weight cuts on the aforementioned

canonical instances may result in some nodes ending up in two different partitions in the final solution. This is shown in Figures 8(b) and (c), where v_2 ends up in both S_1 and S_2, where S_1 is the partition containing terminal s_1 and S_2 the one for s_2.

6.2 Solution

To eliminate the problem of the overlapping resulting partitions, when two or more partitions, *e.g.,* $P_i, P_j \in P$, overlap, $G^{over} = P_i \cap P_j \neq \emptyset$, we apply the following strategy. We compute the cost of removing G^{over} from each one of them by summing the weights of the edges with one end in P_i (or P_j) and the other in G^{over}, and we rank them in descending order of total weights. This implies that the partition where we have to pay the most to remove G^{over}, is first. For this example, we assume that removing G^{over} from P_i is more expensive than removing it from P_j. After this ranking, we select the first partition, *i.e.,* P_i, and we keep G^{over} there, while removing it from all the rest. In the general case, we keep G^{over} to the partition where we have to pay the most for removing it, and we remove it from all the rest. In this example, instead of $P = [P_1, ..., P_i, P_j, ..., P_k]$, the result to the GENERALIZED K MIN-CUT problem will be $P' = [P_1, ..., P_i, P_j - G^{over}, ..., P_k]$.

7 Appendix

In the following lemmas, let:

- $G^{or}(V^{or}, E^{or})$ be the original graph;
- $G^{init}(V^{init}, E^{init})$ be its initial instance;
- f_{base} be a base-flow of G^{init};
- (S^{or}, T^{or}) be the natural partition of G^{or} w.r.t. f_{base};
- $G^{can}(V^{can}, E^{can})$ be the canonical instance of G^{or} w.r.t. f_{base};
- f be the flow induced by f_{base} in G^{can};
- (S, T) be the natural cut of G^{can}.

An *RP* stands for a restricted path in some flow network.

Lemma 3 *Let* σ *be a* special *node with two* extra *nodes in* G^{can}, $\varepsilon_\sigma^1 \in S$ *and* $\varepsilon_\sigma^2 \in T$. *Let also* ε_σ *be the* extra *node of* σ *in* G^{init}.

(a) If $\sigma \in \Sigma_0$, *then* $f(\varepsilon_\sigma^1, \sigma) = 0$ *and* $f(\varepsilon_\sigma^2, \sigma) = 0$.

(b) If $\sigma \in \Sigma_{up}$, *then* $f(\varepsilon_\sigma^1, \sigma) = f_{base}(\varepsilon_\sigma, \sigma)$ *and* $f(\varepsilon_\sigma^2, \sigma) = 0$.

(c) If $\sigma \in \Sigma_{down}$, *then* $f(\sigma, \varepsilon_\sigma^2) = f_{base}(\sigma, \varepsilon_\sigma)$ *and* $f(\sigma, \varepsilon_\sigma^1) = 0$.

Proof

(a) The first property follows directly from $\sigma \in \Sigma_0$ (Definition 7) and the definition of f (Definition 10).

(b) In this case, $\sigma \in \Sigma_{up}$, so $f_{base}(y, \varepsilon_\sigma) \geq 0$, for every $y \in chld(\varepsilon_\sigma)$. Let us assume that there is some $y_2 \in chld(\varepsilon^2_\sigma)$ such that $f_{base}(y_2, \varepsilon_\sigma) > 0$, in order to reach a contradiction. Then, for every $y_1 \in chld(\varepsilon^1_\sigma)$, $f_{base}(y_1, \varepsilon_\sigma) < w(\sigma)$, because $f_{base}(y_1, \varepsilon_\sigma) + f_{base}(y_2, \varepsilon_\sigma) \leq f_{base}(\varepsilon_\sigma, \sigma) \leq w(\sigma)$.

Now let us analyse two cases.

- If there is a RP p from s to some $y_1 \in chld(\varepsilon^1_\sigma)$ in $G^{init}_{f_{base}}$ that does not contain ε_σ, $p \to \varepsilon_\sigma \to y_2$ is a RP from s to y_2 in $G^{init}_{f_{base}}$ (Definition 8 **up-down section**).
- Otherwise, all nodes $y_1 \in chld(\varepsilon^1_\sigma)$ are only reachable from s by RPs of the form $s \rightsquigarrow \sigma \to \varepsilon_\sigma \to y_1$ and $s \rightsquigarrow \sigma \to \varepsilon_\sigma \to y_2$ is also a RP from s to y_2 in $G^{init}_{f_{base}}$ (Definition 8 **down section**).

In both cases, we would conclude that $y_2 \in S^{or} \subseteq S$, which is a contradiction.

Therefore, $f_{base}(y_2, \varepsilon_\sigma) = 0$, for every node $y_2 \in chld(\varepsilon^2_\sigma)$, which implies $f(\varepsilon^2_\sigma, \sigma) = 0$ and $f(\varepsilon^1_\sigma, \sigma) = f_{base}(\varepsilon_\sigma, \sigma)$.

(c) Now, $\sigma \in \Sigma_{down}$, so $f_{base}(\varepsilon_\sigma, y) \geq 0$, for every $y \in chld(\varepsilon_\sigma)$. Let us assume that there is some $y_1 \in chld(\varepsilon^1_\sigma)$ such that $f_{base}(\varepsilon_\sigma, y_1) > 0$, in order to reach a contradiction. Let also $y_2 \in chld(\varepsilon^2_\sigma)$. We know that $f_{base}(\varepsilon_\sigma, y_2) < w(\sigma)$, because $f_{base}(\varepsilon_\sigma, y_1) + f_{base}(\varepsilon_\sigma, y_2) \leq f_{base}(\sigma, \varepsilon_\sigma) \leq w(\sigma)$.

Now let us analyse three cases.

- If there is a RP p from s to y_1 in $G^{init}_{f_{base}}$ that does not contain ε_σ, $p \to \varepsilon_\sigma \to y_2$ is a RP from s to y_2 in $G^{init}_{f_{base}}$ (Definition 8 **up-down section**).
- If there is a RP from s to y_1 of the form $s \rightsquigarrow y \to \varepsilon_\sigma \to y_1$, for some $y \in chld(\varepsilon_\sigma)$, then by Definition 8 **up-down section**, $f_{base}(\varepsilon_\sigma, y) > 0$ and $s \rightsquigarrow y \to \varepsilon_\sigma \to y_2$ is also a RP from s to y_2 in $G^{init}_{f_{base}}$.
- Otherwise, y_1 is only reachable from s by RPs of the form $s \rightsquigarrow \sigma \to \varepsilon_\sigma \to y_1$ and $s \rightsquigarrow \sigma \to \varepsilon_\sigma \to y_2$ is also a RP from s to y_2 in $G^{init}_{f_{base}}$ (Definition 8 **down section**).

In all cases, we would conclude that $y_2 \in S^{or} \subseteq S$, which is a contradiction.

Therefore, $f_{base}(\varepsilon_\sigma, y_1) = 0$, for every node $y_1 \in chld(\varepsilon^1_\sigma)$, which implies $f(\sigma, \varepsilon^1_\sigma) = 0$ and $f(\sigma, \varepsilon^2_\sigma) = f_{base}(\sigma, \varepsilon_\sigma)$. ∎

Lemma 4 *The following properties hold.*

(a) If $e(i,j) \in E^{can}$, $i \in S$, $j \in T$ and neither i nor j is an extra *node, then $f(i,j) = c(i,j)$.*

(b) If $\sigma \in \Sigma_{up}$, $\sigma \in T$ and σ has an extra *node ε'_σ in G^{can} such that $\varepsilon'_\sigma \in S$, then $f(\varepsilon'_\sigma, \sigma) = c(\varepsilon'_\sigma, \sigma) = w(\sigma)$.*

(c) If $\sigma \in \Sigma_{down}$, $\sigma \in S$ and σ has an extra *node ε'_σ in G^{can} such that $\varepsilon'_\sigma \in T$, then $f(\sigma, \varepsilon'_\sigma) = c(\sigma, \varepsilon'_\sigma) = w(\sigma)$.*

Proof

(a) In this case, $e(i,j) \in E^{init}$, so $f(i,j) = f_{base}(i,j)$. Let p be a RP from s to i in $G^{init}_{f_{base}}$. If $f_{base}(i,j) < c(i,j)$, $p \to j$ would be a RP from s to j in $G^{init}_{f_{base}}$, and $j \in S^{or} \subseteq S$. Therefore $f(i,j) = c(i,j)$.

(b) First of all, let us see that $f(\varepsilon'_\sigma, \sigma) = f_{\text{base}}(\varepsilon_\sigma, \sigma)$, where ε_σ is the *extra* node of σ in G^{init}. When ε'_σ is the only *extra* node in G^{can}, the conclusion is immediate and, when there are two *extra* nodes in G^{can}, it follows from Lemma 3 (b).

Suppose $f_{\text{base}}(\varepsilon_\sigma, \sigma) < w(\sigma)$, in order to derive a contradiction, and let $y \in \text{chld}(\varepsilon'_\sigma)$ such that there is a RP p from s to y in $G^{init}_{f_{\text{base}}}$ that does not contain ε_σ. That node must exist as otherwise all nodes in $y' \in \text{chld}(\varepsilon'_\sigma)$ were only reachable from s by RPs of the form $s \rightsquigarrow \sigma \to \varepsilon_\sigma \to y'$ and $\sigma \in S$.

If $f_{\text{base}}(y, \varepsilon_\sigma) = w(\sigma)$, then $f_{\text{base}}(\varepsilon_\sigma, \sigma) = w(\sigma)$ and we reach a contradiction.

If $f_{\text{base}}(y, \varepsilon_\sigma) < w(\sigma)$, $p \to \varepsilon_\sigma \to \sigma$ would be a RP from s to σ in $G^{init}_{f_{\text{base}}}$ and $\sigma \in S$.

Therefore, $f(\varepsilon'_\sigma, \sigma) = f_{\text{base}}(\varepsilon_\sigma, \sigma) = w(\sigma)$.

(c) First of all, let us see that $f(\sigma, \varepsilon'_\sigma) = f_{\text{base}}(\sigma, \varepsilon_\sigma)$, where ε_σ is the *extra* node of σ in G^{init}. When ε'_σ is the only *extra* node in G^{can}, the conclusion is immediate and, when there are two *extra* nodes in G^{can}, it follows from Lemma 3 (c).

Suppose $f_{\text{base}}(\sigma, \varepsilon_\sigma) < w(\sigma)$, in order to derive a contradiction, and let $y \in \text{chld}(\varepsilon'_\sigma)$.

If $f_{\text{base}}(\varepsilon_\sigma, y) = w(\sigma)$, then $f_{\text{base}}(\sigma, \varepsilon_\sigma) = w(\sigma)$ and we reach a contradiction.

So, $f_{\text{base}}(\varepsilon_\sigma, y) < w(\sigma)$, If there is a RP p from s to σ in $G^{init}_{f_{\text{base}}}$ that does not contain ε_σ, then $p \to \varepsilon_\sigma \to y$ would be a RP from s to y in $G^{init}_{f_{\text{base}}}$ and $y \in S$. Otherwise, there must be a RP from s to σ in $G^{init}_{f_{\text{base}}}$ of the form $s \rightsquigarrow y' \to \varepsilon_\sigma \to \sigma$, for some $y' \in \text{chld}(\varepsilon_\sigma)$, which implies $f_{\text{base}}(\varepsilon_\sigma, y') > 0$, because $\sigma \in \Sigma_{\text{down}}$. But then, $s \rightsquigarrow y' \to \varepsilon_\sigma \to y$ would be a RP from s to y in $G^{init}_{f_{\text{base}}}$ and $y \in S$.

Therefore, $f(\sigma, \varepsilon'_\sigma) = f_{\text{base}}(\sigma, \varepsilon_\sigma) = w(\sigma)$. ∎

Lemma 5 *The following properties hold.*

(a) If $\sigma \in \Sigma_{up}$, $\sigma \in S$ and ε_σ is the only extra *node of σ in G^{can}, then $\varepsilon_\sigma \in S$.*

(b) If $\sigma \in \Sigma_{down}$, $\sigma \in T$ and ε_σ is the only extra *node of σ in G^{can}, then $\varepsilon_\sigma \in T$.*

Proof

(a) Let p be a RP from s to σ in $G^{init}_{f_{\text{base}}}$. If p contains ε_σ, it has the form $s \rightsquigarrow y \to \varepsilon_\sigma \to \sigma$, for some $y \in \text{chld}(\varepsilon_\sigma)$, so $y \in S$. Otherwise, let $y' \in \text{chld}(\varepsilon_\sigma)$ such that $f_{\text{base}}(y', \varepsilon_\sigma) > 0$, which exists because $\sigma \in \Sigma_{\text{up}}$. Then, $p \to \varepsilon_\sigma \to y'$ is a RP from s to y' in $G^{init}_{f_{\text{base}}}$ and $y' \in S$.

Since there is only one *extra* node of σ in G^{can}, $\text{chld}(\varepsilon_\sigma) \subseteq S$ and $\varepsilon_\sigma \in S$.

(b) Let $y \in \text{chld}(\varepsilon_\sigma)$ such that $f_{\text{base}}(\varepsilon_\sigma, y) > 0$, which exists because $\sigma \in \Sigma_{\text{down}}$. Let us assume that $y \in S$, in order to derive a contradiction, and let p be a RP from s to y in $G^{init}_{f_{\text{base}}}$.

If p that does not contain ε_σ, $p \to \varepsilon_\sigma \to \sigma$ is a RP from s to σ in $G^{init}_{f_{\text{base}}}$ and $\sigma \in S$. Otherwise, p must have the form $s \rightsquigarrow y' \to \varepsilon_\sigma \to y$, for some $y' \in \text{chld}(\varepsilon_\sigma)$ and, by Definition 8 **up-down section**, $f_{\text{base}}(\varepsilon_\sigma, y') > 0$. But then $s \rightsquigarrow y' \to \varepsilon_\sigma \to \sigma$ is a RP from s to σ in $G^{init}_{f_{\text{base}}}$ and $\sigma \in S$.

Therefore, $y \in T$, $\text{chld}(\varepsilon_\sigma) \subseteq T$, and $\varepsilon_\sigma \in T$. ∎

Lemma 6 *The following properties hold, for every $\sigma \in \Sigma_0$, where ε_σ is its* extra *node in G^{init}:*

(a) If there is a RP from s to σ in $G^{init}_{f_{base}}$ that does not contain ε_σ, then $\{\sigma\} \cup chld(\varepsilon_\sigma) \subseteq S$.

(b) If there is a RP from s to y in $G^{init}_{f_{base}}$ that does not contain ε_σ, for some $y \in chld(\varepsilon_\sigma)$, then $\sigma \in S$.

Proof

(a) If there is a RP p from s to σ in $G^{init}_{f_{\text{base}}}$ that does not contain ε_σ, then $p \to \varepsilon_\sigma \to y$ is a RP from s to y in $G^{init}_{f_{\text{base}}}$ that contains ε_σ only once, for every $y \in \text{chld}(\varepsilon_\sigma)$.

(b) If there is a RP p from s to y in $G^{init}_{f_{\text{base}}}$ that does not contain ε_σ, for some $y \in \text{chld}(\varepsilon_\sigma)$, then $p \to \varepsilon_\sigma \to \sigma$ is a RP from s to σ in $G^{init}_{f_{\text{base}}}$ that contains ε_σ only once. ∎

Lemma 7 *The following properties hold, for every $\sigma \in \Sigma_0$:*

(a) If $\sigma \in S$ and ε_σ is the only extra *node of σ in G^{can}, then $\varepsilon_\sigma \in S$.*

(b) If $\sigma \in T$, then there is only one extra *node ε_σ in G^{can} and $\varepsilon_\sigma \in T$.*

Proof There are only three cases.

- When, for every $v \in \{\sigma\} \cup \text{chld}(\varepsilon_\sigma)$, there is no RP without ε_σ from s to v in $G^{init}_{f_{\text{base}}}$, $\{\sigma\} \cup \text{chld}(\varepsilon_\sigma) \subseteq T$. In this case, $\sigma \in T$ and (b) holds.
- Now, let us assume that p is a RP without ε_σ from s to v in $G^{init}_{f_{\text{base}}}$, for some $v \in \{\sigma\} \cup \text{chld}(\varepsilon_\sigma)$.
  - If $v = \sigma$, by Lemma 6 (a), $\{\sigma\} \cup \text{chld}(\varepsilon_\sigma) \subseteq S$. So, $\sigma \in S$, ε_σ is the only *extra* node in G^{can} and $\text{chld}(\varepsilon_\sigma) \subseteq S$.
  - Otherwise, $v \in \text{chld}(\varepsilon_\sigma)$ and, by Lemma 6 (b), $\sigma \in S$. Since $v \in S$, if there is only one *extra* node in G^{can}, $\text{chld}(\varepsilon_\sigma) \subseteq S$. ∎

References

[1] CORMEN, T. H., STEIN, C., RIVEST, R. L., AND LEISERSON, C. E. *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[2] GOLDBERG, A. V., AND TARJAN, R. E. A new approach to the maximum-flow problem. *J. ACM 35*, 4 (Oct. 1988), 921–940.