Notes on connected cuts

Tomáš Turek

February 4, 2025

Contents

1 Preliminaries

We define graph $G = (V, E)$ as usual. Then we talk about edges defined by a cut in the following way. For vertices $S \subseteq V$ we define $E(S, V \setminus S) = \{e \in E \text{ s.t. } |e \cap S| = 1\}$, then the size of a cut is $e(S, V \setminus S) = |E(S, V \setminus S)|$. Also we will talk about the induced subgraphs which will be denoted as $G[S]$ for some vertices $S \subseteq V$. Just to recall that graph is called connected if between every pair of vertices there exists a walk. Also in most cases we will be considering graphs which are connected, but sometimes this is not necessary.

2 Connected cuts definitions

We will now proceed to some definitions of cuts which are in one way or another connected (meaning that they induce a connected subgraph). We will start simply and build on that.

2.1 Single cuts

Definition 1 (Connected cut)**.** *For a connected graph $G = (V, E)$ we define connected cut as $S \subseteq V$ for which $G[S]$ is connected. The cut itself is $E(S, V \setminus S)$. Later on we may exchange if we will talk about vertices or edges. Also $(V, E \setminus E(S, V \setminus S))$ is a disconnected graph.*

Now we will like to minimize the size of the cut, i.e. the value of $e(S, V \setminus S)$. Also note that the requirement for G being connected is actually not necessary. Sometimes we may even define a connected cut with specific source vertex. This can be formulated by the next definition 2.

Definition 2 (Connected s-cut)**.** *For a connected graph $G = (V, E)$ and given vertex $s \in V$ we define connected cut as $S \subseteq V$ for which $G[S]$ is connected and also $s \in S$.*

This is pretty much the same problem as in previous definition 1. Note that we would also like to minimize the size of the cut, i.e. $e(S, V \setminus S)$. And if we can solve it for the connected s cut we may also apply it for all $s \in V$ to get the value for general connected cut. Some may already see that the property G being connected is still not necessary.

Some may already know that commonly used cut is for defined source and target distinct vertices. We could also use it in our case and only extend the previous definition by saying that $t \notin S$. It is somewhat tempting to also require that $G[V \setminus S]$ is supposed to be also connected. Therefore we can get the full connected $s - t$ cut.

Definition 3 (Connected $s - t$ cut)**.** *For a connected graph $G = (V, E)$ and given two distinct vertices $s, t \in V$ we define connected $s - t$ cut as $S \subseteq V$ for which all following properties hold.*

1. *$s \in S$ and $t \notin S$.*
2. *Both $G[S]$ and $G[V \setminus S]$ are connected.*

2.2 Multi commodity cuts

Now we can furthermore generalize the notion of connected cuts to multi-way connected cuts.

Definition 4 (Multi-way connected cut)**.** *For a connected graph $G = (V, E)$ and pairwise distinct vertices $s_1, s_2, \ldots, s_k \in V$ for $k \in \mathbb{N}$ we define connected cut as partition $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ of vertices (that is $\bigcup_{i=1,\ldots,k} V_i = V$ and for $i \neq j$ $V_i \cap V_j = \emptyset$) such that the following holds:*

1. *$\forall i \in [k] : s_i \in V_i$ and*
2. *$\forall i \in [k] : G[V_i]$ is connected.*

In this specific definitions we may look at our problem from two perspective. Those two options can be seen by the optimization function for the given problem. Sum version is generally more easy to find the solution, or at least very good approximation. On the other hand optimizing over the max function can be way more tricky. Observe that the sum size is already computed with multi-commodity cut.

- *Sum* size as $\sum_{i<j} E(V_i, V_j)$.
- *Max* size as $\max_{i \in [k]} E(V_i, V \setminus V_i)$.

Also we may define **Flexible multi-way connected cut** as relaxing the previous problem. That is the partition will have l partitions where $0 < l \leq k$ and only l sources are representing their partition. So $\forall i \in [l], \exists k : s_k \in V_l$.

2.3 Other cuts

Now we can even further increase the number of requirements. In this case to the size of $|S|$. Now we will also state what is the optimization function and hence declare a problem.

Problem 1 (k-connected cut)**.** *For a connected graph $G = (V, E)$ we say $S \subseteq V$ is k connected cut such that all properties hold:*

1. *$|S| = k$.*
2. *$G[S]$ is connected.*
3. *And we want to minimize $e(S, V \setminus S)$.*

From algorithmic perspective we may also have given source vertex $s \in V$, but as it was stated before we may solve the general case by running $|V|$ times the algorithm for the problem with source.

Note that choosing only two properties from all three can be computed. If we skip the very first one, we may use the result from Garg, which states a linear program having all vertices as such result. Excluding the second one can be also computed via some approximation algorithm for bisection. And Overlooking the last one we just use some search, because we don't care about the size of the result.

3 Absorptive flow

We will be now talking about an absorptive flow. Firstly we will state the problem in a common sense. For a graph and a source we get a flow which flows through the graph and every time it goes through a vertex some of the flow gets absorbed into the given vertex. After that we will define a cut which is induced by such flow and later on state an integer program and its linear approximation. Now we properly state the instance.

Definition 5 (Absorptive flow)**.** *For a graph $G = (V, E)$ and a vertex $s \in V$, also called the source, and for $k \in \mathbb{N}$, such that $|V| \geq k$, we define **absorptive flow** as a tuple of functions denoted as (f_V, f_E), where $f_V : V \to \mathbb{R}$ and $f_E : E \to \mathbb{R}$. Now these two function must have these properties.*

1. $\sum_{v \in V} f_V(v) = k$*, that is every part of the flow gets absorbed,*
2. $f_V(s) = 1$*,*
3. $\forall v \in V : 0 \leq f_V(v) \leq 1$*, so all vertices have some limits,*
4. $\sum_{v \in V,(s,v) \in E} f_E(s, v) = k - 1$*, the flow starts from the source,*
5. $\forall e \in E : 0 \leq f_E(e)$*, the flow has to be non-negative, but can be unlimited,*
6. $\forall v \in V \setminus \{s\} : \sum_{u \in V,(u,v) \in E} f_E((u, v)) = \sum_{u \in V,(v,u) \in E} f_E((v, u)) + f_V(v)$*, thus the whole flow continue unless part of it is absorbed.*

One can already see that it resembles a linear program. Some can expect we would define a size of the flow, but in this special instance we won't be defining it, since the main purpose is to look at the cut, which is defined by the flow. So now we will define the cut.

3.1 Induced cut by the absorptive flow

Firstly we will define $S \subseteq V$ as the vertices which have nonzero function f_V, that is $\forall v \in S : f_V(s) > 0$. Then the **induced cut** defined by absorptive flow is defined as $E(S, V \setminus S)$ and its size as $e(S, V \setminus S)$. We will furthermore want to minimize the size of such cut.

So far the only property is that $s \in S$, which can be seen only from the definition. Next observations come from the linear program and its properties.

4 Integer program

In this section we will establish the linear program which works with this flow and its cut.

4.1 Variables

Firstly we declare the variables for edges and for vertices.

$$f_v = \begin{cases} 1 & \text{if it absorbs the flow} \\ 0 & \text{otherwise} \end{cases}$$

$f_{uv} \in [0, k]$ is for the amount of flow on the edge uv.

$$x_{uv} = \begin{cases} 1 & \text{if } uv \in E(S, V \setminus S) \\ 0 & \text{otherwise} \end{cases}$$

See that these variables arise only from the definition of the problem. There is only change of f_{uv} being limited by k, which can be easily observed to be the same.

4.2 Constraints

Now we need to state the constraints. Firstly set the connection between the absorbed vertices and the cut.

$$\begin{aligned} x_{uv} &\geq f_u - f_v \quad \forall \{uv\} \in E \\ x_{uv} &\geq f_v - f_u \quad \forall \{uv\} \in E \end{aligned}$$

Which is basically that $x_{uv} \geq |f_u - f_v|$. Next we have to set the flow, so its properties hold.

$$\begin{aligned} \textstyle\sum_{v \in V, sv \in E} f_{sv} &= k - 1 \\ f_s &= 1 \\ \textstyle\sum_{u \in V, uv \in E} f_{uv} = \sum_{u \in V, vu \in E} f_{vu} + f_v &\quad \forall v \in V, s \neq v \\ \textstyle\sum_{u \in V} f_u &= k \\ (k-1)) \cdot f_v \geq \textstyle\sum_{u \in V, \{uv\} \in E} f_{uv} &\quad \forall v \in V \setminus \{s\} \end{aligned}$$

The very last constraint is there to ensure some absorption, that is whenever there is some flow to the vertex we must absorb based on the flow and capacity.

4.3 Optimization function

Lastly the optimization function will be to minimize the flow throughput and number of cut edges.

$$\min \sum_{e \in E} x_e$$

4.4 The whole formulation

$$\begin{aligned} \min \textstyle\sum_{e \in E} x_e & \\ x_{uv} \geq f_u - f_v &\quad \forall \{uv\} \in E \\ x_{uv} \geq f_v - f_u &\quad \forall \{uv\} \in E \\ \textstyle\sum_{v \in V, sv \in E} f_{sv} &= k - 1 \\ f_s &= 1 \\ \textstyle\sum_{u \in V, uv \in E} f_{uv} = \sum_{u \in V, vu \in E} f_{vu} + f_v &\quad \forall v \in V, s \neq v \\ \textstyle\sum_{u \in V} f_u &= k \\ (k-1) \cdot f_v \geq \textstyle\sum_{u \in V, \{uv\} \in E} f_{uv} &\quad \forall v \in V \setminus \{s\} \\ f_v \in \{0,1\} &\quad \forall v \in V \\ f_{uv} \in \mathbb{R}^+ &\quad \forall \{u,v\} \in E \\ x_{uv} \in \{0,1\} &\quad \forall \{u,v\} \in E \end{aligned} \tag{1}$$

4.5 Properties

Lets talk about some crucial properties of this integer program.

Observation 1. *Every vertex in the flow absorb.*

Proof. See that due to the last constraint whenever a flow goes inside the vertex we must set the vertex to absorb some portion of the flow. Exactly 1 in the case of integer program. □

Observation 2. *f_V defines a connected induced subgraph of G.*

Proof. Since $f_s = 1$ we know that s is inside the $G[S]$. For contradiction assume there is a vertex $v \in S$ which is not connected to the source s. Because $f_v = 1$ we must have that there is a flow over this vertex due to the fifth constraint. And since there is a flow to the vertex v which starts in s there is also a walk from s to v, therefore no such vertex v exists and all vertices are connected to s and hence they induce connected subgraph. □

Observation 3. *We have a minimum connected s cut from the optimal integer value of the lp formulation x^*.*

Now it is also a time to talk about integrality, moreover what will happen if we switch to lp relaxation.

4.6 Consider the distance from source

In this subsection we will provide even better model, which arises from the previously mentioned one. This time we will also use the notation $d(u, v)$ for shortest path between vertices u and v. This is a well known property of graphs, which can be computed in polynomial time (e.g. using Dijsktra's algorithm). Therefore we will increase what we want the vertex absorb by setting the fraction $\frac{1}{k-1}$ to $\frac{1}{k-d(s,u)}$ for every u. That is we combine two different metrics of path length. One in terms of standard length and the other of terms flow. The linear program is as follows.

$$\begin{aligned}
\min \textstyle\sum_{e\in E} x_e & \\
x_{uv} \geq f_u - f_v & \quad \forall \{uv\} \in E \\
x_{uv} \geq f_v - f_u & \quad \forall \{uv\} \in E \\
\textstyle\sum_{v\in V, sv\in E} f_{sv} = k-1 & \\
f_s = 1 & \\
\textstyle\sum_{u\in V, uv\in E} f_{uv} = \sum_{u\in V, vu\in E} f_{vu} + f_v & \quad \forall v \in V, s \neq v \\
\textstyle\sum_{u\in V} f_u = k & \\
(k - d(s,v)) \cdot f_v \geq \textstyle\sum_{u\in V, \{uv\}\in E} f_{uv} & \quad \forall v \in V \setminus \{s\} \\
f_v \in \{0,1\} & \quad \forall v \in V \\
f_{uv} \in \mathbb{R}^+ & \quad \forall \{u,v\} \in E \\
x_{uv} \in \{0,1\} & \quad \forall \{u,v\} \in E
\end{aligned} \tag{2}$$

5 Approximation

The final approximation we will employ is quite simple. That is we will have a vertex and a set of already chosen vertices. Firstly add s to the set and set it as a current vertex. Then always look at neighbors with non-zero vertex flow and consider them as another choice. With the flow values choose one with these probabilities.

Algorithm 1 Approximation of the values from linear program.

Require: A graph G with source s and capacity k and flow f on vertices from LP.
Ensure: A connected cut S.

```
S ← {s} and cv ← s.
while |S| < k do
    for all neighbors v of cv do
        If v ∉ S and f(v) > 0 add it to consideration.
    end for
    Choose one vertex u from the considered ones based on the f.
    S ← S ∪ {u} and cv ← u.
end while
return S.
```

6 Improving the result

Also there is the fact that we want to have cut edges only when the flow is really low. Otherwise it does not make any sense. So either we may be improving our linear program along the way or just consider this during the approximation. That is we will call the edges with flow > 1 and cut > 0 as a bad edges. Then we will have a procedure which will enhance the linear program. We will gradually add bad edges to the LP statement where we set these to 0.

Note that this enhancement will in worse case go through all edges, thus $m = |E|$. We should take a while and convince ourselves that this improvement does not diverge from the optimal solution, moreover it will get closer to some optimal solution.

7 Example graphs

We will be looking at a few graphs and mostly their classes and try to observe some properties.

7.1 Star graph

Definition 6. *Star graph is a graph with one vertex with degree Δ and the rest of the vertices have degree 1 and are connected to the main one.*

Algorithm 2 Enhancement

Require: Graph G with source s and capacity k.
Ensure: Enhanced LP.

```
Solve the original LP.
F = ∅ and cont = false.
while cont do
    cont = false.
    for all edges e in G do
        if f(e) > 1 and x(e) > 0 then
            F = F ∪ {e}.
            cont = true.
        end if
    end for
    for all e in F do
        Set x(e) = 0 in the LP.
    end for
    Solve updated LP and update solution.
end while
return Last LP.
```

Algorithm 3 Connected cut algorithm.

Require: A graph G with source s and capacity k.
Ensure: A connected cut S which minimizes its cost.

```
Create a LP.
Enhance the LP.
Solve the LP.
Use approximation algorithm to obtain S.
return S.
```

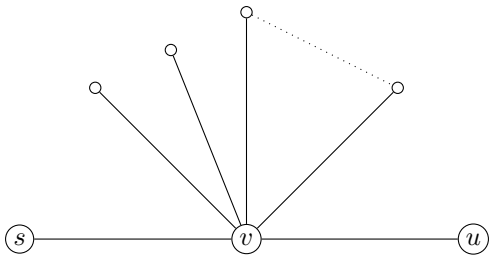

Figure 1: Star graph G with special vertices v, s and u.

7.2 Comet graph

Definition 7. *Comet graph contain a star and a tail – hence the name comet.*

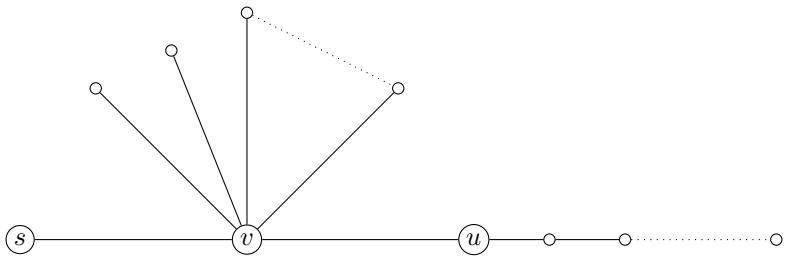

Figure 2: Comet graph G.

8 NP hardness

In this section we would like to provide a proof that k-connected cut with weights is NP complete problem. One can already see whats the definition of wights, but we will still clarify this. In k-connected cut we will also

have a wight function $w : E \to \mathbb{R}_0^+$ and we want to minimize the sum of weights in the edge cut. That is the non-weighted version has constant weight function equal to 1.

Now we will state a well known NP complete problem: *Knapsack*. That is we have n items which we can enumerate with the set $[n] = \{1, 2, \ldots, n\}$. Then we have weights of the items: $w[n] \to \mathbb{R}^+$ and also values: $v : [n] \to \mathbb{R}^+$. Lastly we also have a capacity k. The goal is to pick a subset $A \subseteq [n]$ such that $\sum_{a \in A} w(a) \leq k$ and the value $\sum_{a \in A} v(a)$ is maximized.

Lets have an instance of knapsack problem; that is (n, v, w, k) and we will create a graph where $G = (V, E)$ where k-connected weighted cut will correspond to the solution of knapsack problem.

8.1 Reduction

Firstly we will introduce a source vertex s. Then we will add $k - 1$ vertices which we will call null-vertices. These will be connected to s and the edges will have weight 0. Then for every item in $[n]$ we will create a clique $K_{w(n)}$; that is the size os equal to the weight og an item. We will connect each clique to the source by one edge with the weight equal to the value of an item. Also the edges inside each clique will be equal to the value of the item.

Note that we assume the weights and values are in $\mathbb{N}$. Which can be for numbers in $\mathbb{Q}$ done easily by scaling the numbers.

Observation 4. *For a solution $S \subseteq V$ of k-connected weighted cut when a vertex v from a clique K is inside S then $K \subseteq S$.*

Proof. Suppose that is not the case and there is $u \in K$ such that $u \notin S$. Hence the cut inside the clique adds $(w(a) - 1) \cdot v(a)$ which is clearly larger or equal then $v(a)$. Because if $w(a) = 1$ then it does not make sense, because if one vertex is chosen, then all are; since there is only one. So we would get equal or better solution by choosing null-vertices instead. □

If we assume the weights are at least 3, then it is strictly better.

Observation 5. *If at least two items can be chosen, then s is inside best solution.*

Proof. This property arises from the fact that s connects all parts together. □

Hence when we compute the k-connected weighted cut we obtain a set S where $s \in S$ and for some indices $i_1, i_2, \ldots, i_m$ the cliques $K_{i_1}, K_{i_2}, \ldots, K_{i_m} \subseteq S$ and also some null-vertices are in S as well. These cliques correspond to the chosen elements of the knapsack problem.

Theorem 6. *This solution of the knapsack problem is optimal.*

Proof. Imagine we would have a better solution. Firstly consider case where an element a can be added to the solution. This would mean that we have some unused capacity. Therefore in our cut we have used null vertices and we could instead use corresponding clique. But that would get us a better solution which is a contradiction. Hence in some sense it is at least good as greedily creating maximal set.

Lets now suppose that there are items $a_1, a_2, \ldots a_l \in S$ and $b_1, b_2, \ldots b_m \notin S$ which can be exchanged for a better solution. In the cut we would have capacity for doing so. Also b's add $\sum v(b)$ to the cut cost whereas the a's would add $\sum v(a)$. But since $\sum v(b) > \sum v(a)$ it means that we would get a better cut, which is a contradiction with the optimality of the cut. □

Observation 7. *The reduction is in polynomial time.*

Proof. We create 1 source vertex, $O(k)$ null vertices and $O(\sum_{a \in A} w(a))$ vertices for cliques. Then we add $O(k)$ edges for null vertices, $O(|A|)$ connections to the cliques and $O(\sum_{a \in A} w^2(a)$ edges inside each clique. Hence all together it is $O(W^2 + k)$ where $W = \sum_{a \in A} w(a)$. *Perhaps there is a little catch. That is it is considering the wights as a values, which may be pseudo polynomial.* □

9 Links

Here I am keeping some useful links.

1. Approximating unique games
2. Minimum Bisection
3. Min max and small set expansion

4. Approximation algorithms for maximally balanced connected graph partition
5. On size-constrained minimum s–t cut problems and size-constrained dense subgraph problems
6. On the minimum $s - t$ cut problem with budget constraints
7. Balanced Crown Decomposition for Connectivity Constraints