Kingdom of Saudi Arabia **Al-Imam Mohammad Ibn Saud Islamic University** **Faculty of Science**		**المملكة العربية السعودية** **جامعة الإمام محمد بن سعود الإسلامية** **كلية العلوم**

Department of Mathematics and Statistics

Master of Science in Mathematics

Research Project (MAT 699)

Project Title

The Minimum k-cut Problem

Presented by

May Abdullah Al-Askar

Supervised by

Dr. Brahim Chaourar

IMSIU – Riyadh – KSA

May 2017

I dedicate this project to my parents and my family.

Acknowledgments

I would like to express the deepest appreciation to my super devisor Dr. Brahim Chaourar whose support, stimulating suggestions and encouragement helped me all the time of the research and during writing of this project.

Contents

Table of figures:

Chapter 1: Introduction

Combinatorial optimization is a subset of mathematical optimization that is related to operations research, algorithm theory, and computational complexity theory. It is a topic that consists of finding an optimal object from a finite set of objects. In many such problems, an exhaustive search is not feasible. It operates on the domain of those optimization problems, in which the set of feasible solutions is discrete or can be reduced to discrete, and in which the goal is to find the best solution.

Combinatorial optimization has important applications in several fields, including artificial intelligence, machine learning, mathematics, auction theory, and software engineering.

One of the famous combinatorial optimization problems is the minimum cut problem which has been solved in polynomial time for the first time by using the max-flow min-cut theorem. An important and natural generalization of the minimum cut problem is the minimum k-cut problem (MKCP). In the contrast of the minimum cut problem, MKCP is NP-hard. MKCP has applications in very-large-scale integration (VLSI) design, data-mining, finite elements and communication in parallel computing.

In this thesis, we give a survey of MKCP.

Our contribution is to survey most known results for MKCP in one essay. For the best of our knowledge, this is done for the first time.

A minor contribution is the solvability of MKCP in polynomial time for forests in general.

Organization of the thesis

In chapter 2, we introduce some preliminaries on graph theory and complexity of algorithms.

In chapter 3, we define MKCP, determine its complexity, and present some of its variants.

In chapter 4, we present some applications of MKCP and some of its variants.

In chapter 5, we describe some solved special instances of MKCP and some of its variants.

In chapter 6, we describe known approximation algorithms for MKCP and some of its variants.

Finally, we conclude in chapter 7.

Chapter 2: Preliminaries

2.1 Graph theory

We will introduce some definitions on graph theory, and present the important theorem linking between max-flow and min-cut.

2.1.1 Fundamental definitions

Definition 1:

A graph G is a pair of sets (V(G), E(G)) where V(G) is the set of vertices and $E(G) \subseteq V(G) \times V(G)$ is the set of edges.

G is undirected if (u, v) and (v, u) represent the same edge e denoted {u, v} or uv or vu.

In this case, u and v are called the end points of the edge e. u and v are adjacent vertices.

All considered graphs are undirected except particular mentioned cases.

The ends of an edge are said to be incident with the edge. If two or more edges of a undirected graph have a common vertex, then the edges are called adjacent.

A loop is an edge with identical ends. **A link** is an edge with distinct ends. A **multiple edge** or a parallel edge is set of edges sharing the same end points.

The degree of a vertex v in graph G is the number of edges incident with v, and it is denoted by deg(v). The order of G is the number of vertices of G, denoted by n = $|V(G)|$, and the size of G is the number of edges of G, denoted by $m = |E(G)|$. A graph is finite if its order and size are finite.

A graph is **simple** if it has neither loops or multiple edges.

A **directed graph** or (digraph) is a graph where the edges (u, v) and (v, u) do not represent the same edge.

A graph that has a weight function $w: E \to R^{+}$ from the set of edges into the set of nonnegative real number is called a **weighted graph**. Weighted graphs may be either directed or undirected.

Graphs come in many several types; we will present some of them.

Definition 2

A complete graph, $\mathrm{K_n}$, is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.

Definition 3

A subgraph F = (V(F), E(F)) of a graph G = (V(G), E(G)) is a graph such that V(F) ⊆ V(G) and E(F) ⊆ E(G).

Let G = (V, E) be an undirected graph. A **clique** C is a complete subgraph of G. The **size of clique** is the number of vertices in clique.

A **maximum clique** of a graph is a clique, such that there is no clique with more vertices.

Given a subset U of vertices of G = (V, E), an **induced subgraph** G(U) is a subgraph of G, such that we keep all edges of G linking vertices of U in G(U).

Definition 4

A bipartite graph G (V, E) is a graph whose vertices set V can be separated into two subsets $\mathrm{V_1}$. and $\mathrm{V_2}$ such that every edge has an endpoint in $\mathrm{V_1}$ and another one in $\mathrm{V_2}$.

Definition 5

A **path** (or a linear graph) is a simple graph whose nodes can be ordered in a linear sequence $v_1, \dots, v_n$, such that for every i, $(1 \leq i \leq n-1)$, $v_i v_{i+1}$ are the edges of the graph. In this case, we called it a $v_1 v_n - path$.

A **connected graph** is a graph for which for any pair of vertices u and v, there exists a uv-path. Otherwise, it is called a disconnected graph.

A **connected component** U of a graph G is a maximal set of vertices such that the corresponding induced subgraph G(U) is connected. The number of connected components of G is denoted by comb(G).

A **cycle**, $\mathrm{C_n}$ (n ≥ 3), is a simple graph of n vertices $v_1, \dots, v_n$ and n edges ($v_1 v_2$, $v_2 v_3, \dots , v_{n-1} v_n$, $v_n v_1$). A **Hamilton cycle** of a graph G is a cycle that pass through every vertex of G exactly once.

Definition 6

A **planar** graph is a graph which can be drawn in the plane such that its edges intersect only at their end points.

Definition 7

A **forest** is an undirected graph in which any two vertices are connected by at most one path. A **tree** is a connected forest.

2.1.2 Cuts

Let G = (V, E) be an undirected graph. A **cut** related with a set of vertices A, $\mathrm{C}(\mathrm{A},\overline{\mathrm{A}})$ or C(A), is the set of edges with one endpoint in A and the other in $\overline{\mathrm{A}}$.

Given a partition $\{V_1, V_2, \dots, V_k\}$ of V, a **k-cut** C $(V_1, V_2, \dots, V_k)$ is the set of edges with one end point in one of the V_i, i = 1, 2, …, k, and the other one is in another distinct V_j, j = 1, 2, …, k, i.e., i ≠ j. Each V_i, i = 1, 2, …, k, is called a component of the k-cut.

In this case, a **self-edge** is an edge which both endpoints in the same component, and a **cross-edge** is an edge which endpoints in different components.

For any pair of vertices x and y in G, a xy-**cut** is any cut C $(\mathrm{X},\overline{\mathrm{X}})$ such that $\mathrm{x} \in \mathrm{X}$ and $\mathrm{y} \in \overline{\mathrm{X}}$.

The value of a cut (respectively a k-cut) in an unweighted graph is the number of edges belonging to this cut (respectively k-cut). The value of a cut (respectively a k-cut) in a weighted graph is the total sum of the weights of the edges belonging to this cut (respectively k-cut).

2.1.3 Minimum st-cut problem

The minimum st-cut problem can be defined as follows:

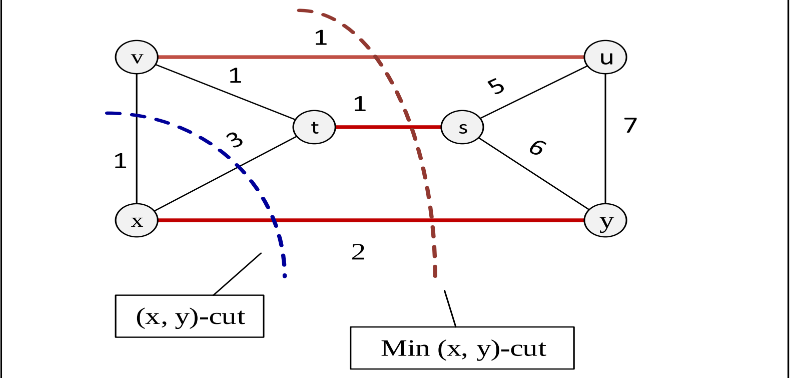
Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a pair of distinct vertices s and t.

Question: Find the st-cut with the minimum value.

Example

Consider the graph G = (V, E) in figure (2-1), the set of edges $\{\mathrm{xv}, \mathrm{xt}, \mathrm{xy}\}$ is an xy-cut, with value 4, the min xy-cut is the set of edges {vu, ts , xy} with weight 4.

There are many approaches to solve the min st–cut problem, we will present the one depending on the max-flow min-cut theorem.



Figure(2-1) : Example of min(x, y)-cut

We will define some basic concepts on directed graph necessary for the max-flow min-cut theorem.

2.1.3.1 Cuts in directed graph

In a directed graph G = (V, E), an **out cut** $C^{+}(A)$ or $\vec{C}(A, \bar{A})$ related to a set of vertices A, is the set of edges starting from A and ending in $\bar{A}$.

An **in cut** $C^{-}(A)$ or $\vec{C}(\bar{A}, A)$ related with a set of vertices A is the out cut $C^{+}(\bar{A})$.

A **cut** $C(A)$ **in a directed graph** G related with a set of vertices A, is the union of the corresponding out cut and in cut, i.e., $C(A) = C^{+}(A) \cup C^{-}(A)$.

2.1.3.2 Flows

A network N = N (s, t)) is a directed graph with two distinct vertices, a source s and a sink t, together with a nonnegative real function (capacity function) defined on its edge set E, and the remaining vertices, I, are called intermediate vertices.

The capacity of an edge is a mapping cap: E → R+. It represents an upper bound of flow that can pass through an edge.

A flow f in a network N(s, t) is mapping f: $E \rightarrow R^+$, that satisfies the following conditions:

- **Capacity constraint condition:** for every intermediate vertex v,

$$f^-(v) = \sum_{\{u:\, uv \in E\}} f(uv) = \sum_{\{u:\, vu \in E\}} f(vu) = f^+(v)$$

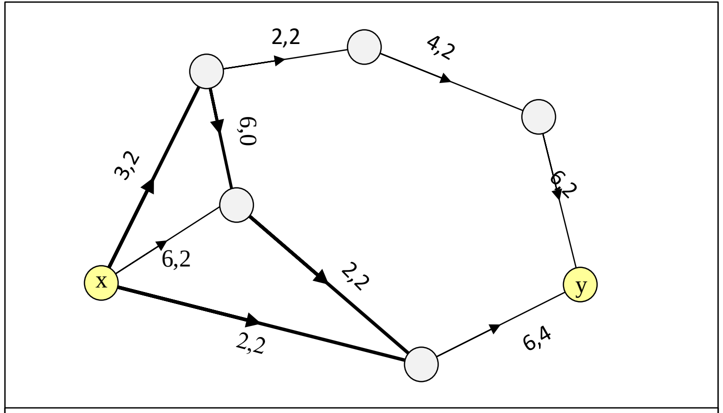
- **Conservation condition**: for every edge uv,

$$f(uv) \leq cap(uv).$$

The **value of the flow** f is defined by $val(f) = \sum_{v \in V} f(sv)$, where *s* is the source of the network N(s, t). It represents the amount of flow passing from the source to the sink.

If A is a set of vertices in the network, the net flow out of A is $f^+(A) - f^-(A)$, the net flow into A is $f^-(A) - f^+(A)$.

For example, network in figure (2-2) the value of flow is 6.



Figure(2-2) Example of flow in a network N(x, y). Each edge is labeled with its capacity, flow.

Proposition (2-1)

Let N(s, t) be a network, and A be any subset of vertices in N. For any flow f, we have:

$$\sum_{v \in A} \left(f^+(v) - f^-(v)\right) = f^+(A) - f^-(A)$$

Proof

$\mathrm{f}^+(\mathrm{A}/\{\mathrm{v}\}) = \mathrm{f}^+(\mathrm{A}) - \mathrm{f}^+(\mathrm{v}) + \mathrm{f}^+(\mathrm{v}, \mathrm{A}/\{\mathrm{v}\})$,

$\mathrm{f}^-(\mathrm{A}/\{\mathrm{v}\}) = \mathrm{f}^-(\mathrm{A}) - \mathrm{f}^-(\mathrm{v}) + \mathrm{f}^-(\mathrm{A}/\{\mathrm{v}\}, \mathrm{v})$,

$\sum_{\mathrm{v}\in \mathrm{A}/\{\mathrm{v}\}} \big(\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v})\big) = \mathrm{f}^+(\mathrm{A}/\{\mathrm{v}\}) - \mathrm{f}^-(\mathrm{A}/\{\mathrm{v}\})$

$\sum_{\mathrm{v}\in \mathrm{A}/\{\mathrm{v}\}} \big(\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v})\big) = \mathrm{f}^+(\mathrm{A}) - \mathrm{f}^+(\mathrm{v}) + \mathrm{f}^+(\mathrm{v}, \mathrm{A}/\{\mathrm{v}\}) - \mathrm{f}^-(\mathrm{A}) + \mathrm{f}^-(\mathrm{v}) + \mathrm{f}^-(\mathrm{A}/\{\mathrm{v}\}, \mathrm{v})$

$\sum_{\mathrm{v}\in \mathrm{A}/\{\mathrm{v}\}} \big(\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v})\big) = \mathrm{f}^+(\mathrm{A}) - \mathrm{f}^-(\mathrm{A})$. □

Lemma (2-1)

Let N(s, t) be a network, and A be any set of vertices in N such that $s \in A$ and $t \in \bar{A}$. For any flow f, we have: $\mathrm{val(f)} = \mathrm{f}^+(\mathrm{A}) - \mathrm{f}^-(\mathrm{A})$.

Proof:

For every $\mathrm{v} \in \mathrm{A} - \{\mathrm{s}\}$, $\mathrm{f}^+(\mathrm{v}) = \mathrm{f}^-(\mathrm{v})$, then $\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v}) = 0$, hence

$\sum_{v\in A-\{s\}} \big(\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v})\big) = 0$ and since $\mathrm{f}^+(\mathrm{s}) - \mathrm{f}^-(\mathrm{s}) = \mathrm{val(f)}$, then

$\mathrm{f}^+(\mathrm{A}) - \mathrm{f}^-(\mathrm{A}) = \sum_{v\in A} \big(\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v})\big) = \mathrm{f}^+(\mathrm{s}) - \mathrm{f}^-(\mathrm{s}) + \sum_{v\in A-\{s\}} \big(\mathrm{f}^+(\mathrm{v}) - \mathrm{f}^-(\mathrm{v})\big)$.

Hence: $\mathrm{f}^+(\mathrm{A}) - \mathrm{f}^-(\mathrm{A}) = \mathrm{val(f)}$. □

The following theorem proves that the capacity of any st-cut in a network N(s, t) is bounded below by the value of any flow from s to t, or an st-flow.

Theorem (2-1)

Let N(s, t) be a network**.** For any cut $\mathrm{C}^+(\mathrm{A})$, such that $s \in A$, and for any flow f in N, we have:

$$\mathrm{val(f)} \le \mathrm{cap}(\mathrm{C}^+(\mathrm{A}))$$

Proof

By lemma (2-1), and since $\mathrm{f}^-(\mathrm{A}) \geq 0$, then $\mathrm{val(f)} = \mathrm{f}^+(\mathrm{A}) - \mathrm{f}^-(\mathrm{A}) \le \mathrm{cap}(\mathrm{C}^+(\mathrm{A}))$. □

2.1.3.3 Maximum flow problem

Input: Given a network $\mathrm{N(s,t)}$, of order n and size m, a nonnegative capacity function $\mathrm{cap}: E \longrightarrow N$.

Question: Find an st-flow f with a maximum value.

The first algorithm solving maximum flow problem is due to Ford and Fulkerson (1955).

2.1.3.4 The max-flow min-cut theorem

Theorem (2-2)

The maximum value of an st-flow is equal to the minimum capacity over all st-cuts.

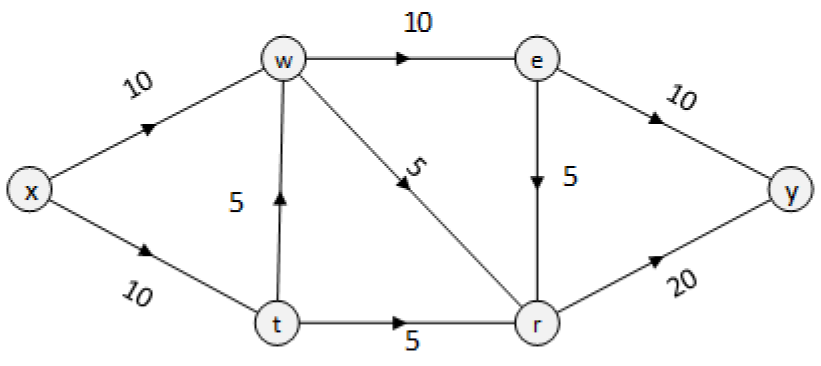
This theorem was proved by Elias et al. in 1956, and independently also by Ford and Fulkerson in 1956.

By theorem (2-2) we conclude that the maximum st-flow problem is equivalent to the min st-cut problem, i.e., by maximizing the value of the flow from s to t we can get the minimum st-cut.

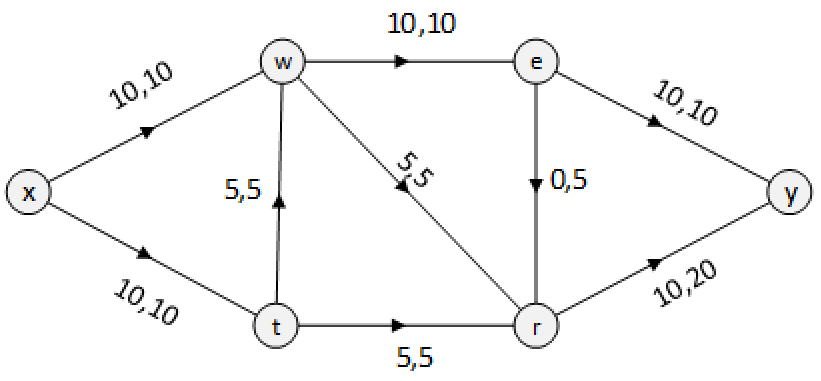
Example:

Let G = (V, E) be the directed graph of figure (2-3). To find a min xy-cut by using the maximum flow algorithm, we will use the final residual graph corresponding to graph G, and find the set of vertices that are reachable from source in the residual graph, all edges which start from a reachable vertex to a non-reachable vertex represent a min xy-cut. The maximum value of an xy-flow is 20, then by max- flow min -cut theorem the total weight of min xy-cut is 20.

Graph G=(V,E), each edge is labeled with its capacity.



Graph G=(V,E), a flow f in G, each edge is labeled with it capacity , flow.



Residual graph Gr(V,E) corresponding to graph G.The red edges are min(x, y)-cut.

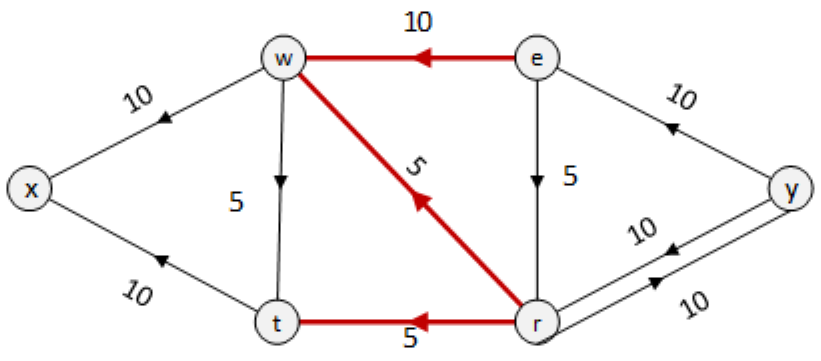


Figure (2-3):Example of application of max-flow min-cut theorem.

2.1.4 Minimum Cut Problem

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, and a nonnegative weight function $w\colon E \longrightarrow N$.

Question: Find a cut in G with the minimum total weight.

For every pair of vertices x and y in the graph G = (V, E) of order n, if we compute all min xy-cuts, and then pick the xy-cut having the smallest weight, we will get the minimum cut on G. Gomory and Hu (1961) proved that the number of different xy-cuts in a graph is at most n−1 cuts, and moreover that there is an effectual tree construction, representing the minimum xy-cuts for all pairs x and y in the graph.

For example, consider the graph G= (V, E) in figure (2,4). The weight of min xy-cut between any pair of vertices of G is equal to the weight of min xy-cut of the vertices of corresponding Gomory-Hu tree.

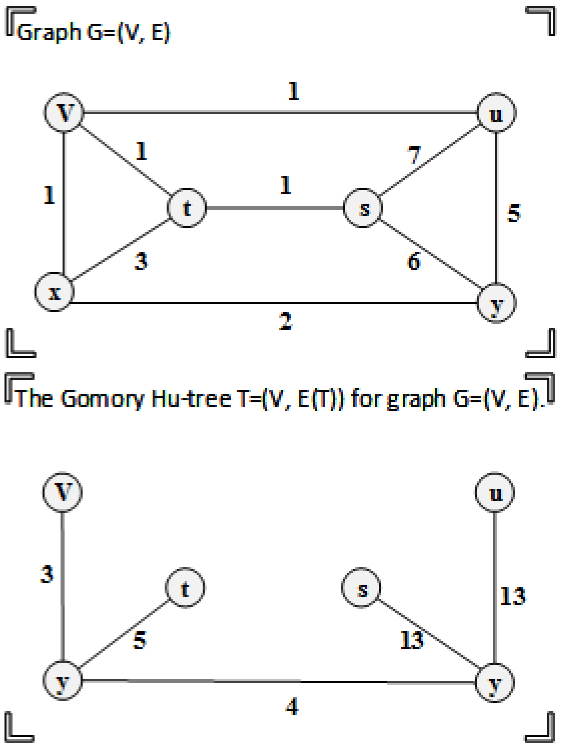


Figure (2-4): Example of Gomory Hu- tree

2.2 Complexity of Algorithms

An algorithm is a well-defined computational producer which takes any instance of the problem as input and return the solution as output. For any algorithm, the number of elementary computational steps in demand for the algorithm and done by it is the computational complexity of the algorithm. It depends on the size and the nature of the input.

We will measure the complexity not in absolute terms but as a function of the data size (sometimes it depends on more than one variable). It is usually expressed using big O notation that we can estimate the growth of a function without worrying about constant multipliers or smaller order terms.

Big O notation ,O(f(n)), is the set of functions that asymptotically grow in the same manner as f, i.e. $O(f(n)) = \{h(n)$: there exist c and $N > 0$ such that $\forall\ n > N, 0 \leq h(n) \leq c\ f(n)\}$

If the complexity of algorithm is $O(n^b)$, with a fixed $b > 0$, then the algorithm is called a **polynomial time algorithm.** If the complexity of an algorithm is $O(n)$, then the algorithm is called a **linear time algorithm.**

Strong polynomial time algorithm is algorithm satisfies the following:

- With arbitrary input, the complexity of algorithm is $O(n^b)$, with a fixed $b > 0$.
- the space used by the algorithm is bounded by a polynomial in the size of the input.

For problems, most important classifications depend on the difficulty of the problem. We will present some famous classes of problems.

2.2.1 The Class P

Definition

The class of problems solved by a polynomial time algorithm.

Example:

The maximum flow problem. This problem has been solved by a polynomial time algorithm. King et al. (1992) find a polynomial algorithm for maximum flow problem with running time $O(mn + n^{2+\varepsilon})$. The best running time for max flow is $O(mn)$, it is due to Orlin (2013).

2.2.2 The Class NP

Definition

A decision problem belongs to the class NP, if given any instance of the problem whose answer is **yes**, we can check it in polynomial time.

Example:

Input: Graph (V, E).

Question: Is G bipartite?

This problem belongs to the NP class since we can check in polynomial time, for given a bipartition [X, Y] of G if each edge in G has one endpoint in X and another end in Y.

2.2.3 The Class Co-NP

Definition

A decision problem belongs to the class co-NP, if given any instance of the problem whose answer is **No**, we can check it in polynomial time.

Example:

Input: Graph (V, E).

Question: Is G bipartite?

This problem belongs to the co-NP class since we can check in polynomial time by using the proposition (every non-bipartite graph contains an odd cycle).

From both definitions of NP and Co-NP classes, we can conclude a relation between the previous classes:

$\mathrm{P} \subseteq \mathrm{NP}$ and $\mathrm{P} \subseteq \mathrm{co-NP}$, then $\mathrm{P} \subseteq \mathrm{NP} \cap \mathrm{co-NP}$. (See figure (2-5)) (Wikipedia 2007).

A clearly but still open question is whether the classes NP and co-NP are different. It is generally believed that NP $\neq$ co-NP, but nobody knows how to prove it.

Example:

Input: A nonnegative integer x.

Question: Is x prime?

This problem belongs to the co-NP class since we can check in polynomial time, if there exists integer number y_1, y_2 such that $x = y_1 \times y_2$, then x not prime. But this problem not belongs to NP class, until now there is no polynomial algorithm to check whether an integer number is prime.

2.2.4 NP-complete problems

Definition

A **polynomial-time reduction**, Π, of a problem F to a problem Q is a polynomial time algorithm, which transforms each input S in F to an input T in Q, and which transforms an output for the input T to an output for input S in polynomial time, and we can say that problem F is polynomial reducible to problem Q by Π, denoted by $F \preccurlyeq_{\Pi} Q$. This relation is reflexive i.e. for any problem F, $F \preccurlyeq_{\Pi} F$. It is also transitive, i.e. let F, Q and R be problems, if $F \preccurlyeq_{\Pi 1} Q$ and $Q \preccurlyeq_{\Pi 2} R$ then $F \preccurlyeq_{\Pi 3} R$.

Definition

A problem F is called **NP-complete**, if F belongs to class NP, and for each problem Q in NP there exists a polynomial time reduction of Q to F, i.e.,

if $F \in NP$, and $\forall Q \in NP, Q \preccurlyeq F$, then $F \in$ Np-complete.

It implies that if one NP-complete problem can be solved in polynomial time, then each problem in NP can be solved in polynomial time. Moreover, if F belongs to NP, Q is NP-complete and there exists a polynomial-time reduction of Q to F, then also F is NP-complete, i.e.,

If $F \in$ NP, $Q \in$NP-complete and $Q \preccurlyeq_{\Pi} F$ then $F \in$ NP-complete.

Theorem (2-3)

The satisfiability problem for Boolean formula is NP-complete.

This theorem was proved independently by Cook (1971) and Levin (1973).

Example

The maximum clique problem:

Input: Given an undirected graph $G = (V, E)$ and $M \in Z^+$.

Question: Is there a clique in G of size greater than or equal to M?

By applying Theorem (2-3), Karp (1972) proved that maximum clique problem is NP-complete. (He finds a polynomial time reduction from satisfiability to maximum clique problem)

2.2.5 NP-hard problems

Definition

A decision problem F is NP-hard if, for every problem Q in NP, there is a polynomial-time reduction from Q to F.

There is no polynomial-time algorithm (right now) that solves an NP-hard problem optimally unless P = NP.

In other words: F is NP-hard $\Longleftrightarrow$ If F can be solved in polynomial time, then P = NP.

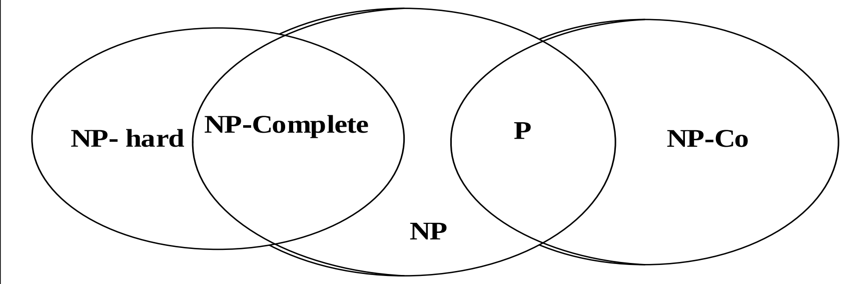
Note that some NP-hard optimization problems can be polynomial time approximated up to some constant approximation ratio.

Example

The traveling salesman problem:

Input: A weighted graph.

Question: Find a Hamilton cycle with minimum total weight.



Figure(2-5) P, NP, NP-complete, and NP-hard set of problems. The figure is valid under the assumption that P≠NP. (Wikipedia 2007)

2.2.6 Approximation algorithm

We will define the ratio between the result obtained by the algorithm and the optimal solution. It helps us to know how much the approximation is close to the optimal solution, and to compare between different algorithms.

Let cost(opt(I)) be the value of an optimal solution to a problem for input I, and cost(sol(I)) be the value of an algorithm solution to the same problem for the same input I.

Definition

For some $\alpha > 1$, the α-approximation algorithm for a minimization problem is an algorithm which produces the value of an optimal solution to the problem for any input I with at most $\alpha\times$cost(opt(I)). α is called the approximation factor, i.e.

cos(opt(I)) $\leq$ cost(sol(I)) $\leq \alpha\times$cost(opt(I)).

For some $\beta < 1$, the β-approximation algorithm for a maximization problem is an algorithm which produces the value of an optimal solution to the problem for any input I with at least $\beta\times$cost(opt(I)), i.e., $\beta\times$cost(opt(I)). $\leq$ cost(sol(I)) $\leq$ cost(opt(I)).

Chapter 3: The Min k-cut problem and some of its variants

In this chapter, we introduce the Min k-Cut Problem: definition, complexity and then discuss some of its variants.

3.1 Minimum k-Cut Problem

3.1.1 Definition

The minimum k-cut problem optimization version:

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a positive integer k.

Question: Find a k-cut with the minimum total weight.

The minimum k-cut problem decision version:

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a positive integer k and $M \in N$.

Question: Is there a k-cut with total weight less than or equal to M.

This problem is a generalization of the Minimum Cut Problem which can be solvable in polynomial time, based on the fundamental Max-Flow Min-Cut Theorem.

3.1.2 Complexity

For any graph and if k is a part of the input (k is variable) Goldschmidt and Hochbaum (1994) proved this theorem:

Theorem 3.1:

The Minimum k-cut decision problem is NP-complete (if k is a part of the input).

Proof:

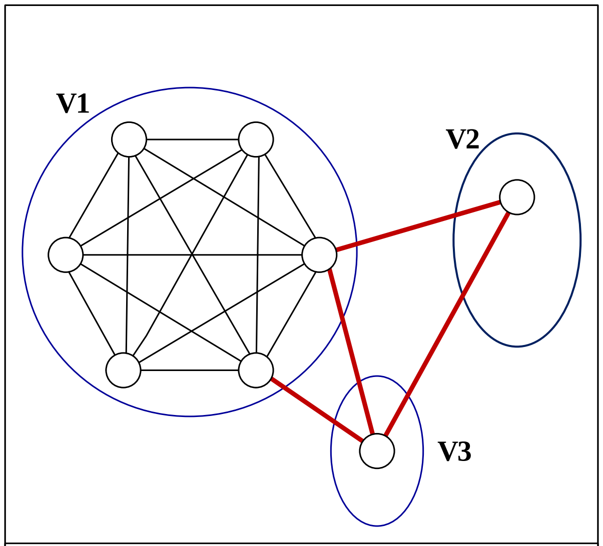
To prove this theorem, they have reduced Maximum Clique decision problem to Minimum k-cut decision problem in polynomial time.

By using the fact that the Maximum Clique decision problem is NP-complete (this problem was one of the 21 NP-complete problems that Karp (1972) enumerated)
We can present the Maximum Clique decision problem as follows:
Input: Given an undirected graph $G = (V, E)$ and $M \in Z^+$.
Question: Is there a clique in G of size greater than or equal to M?
Consider an undirected graph with $\{0, 1\}$ weights on the edges. The minimum k-cut is equivalent to separate the given graph into at least (nonempty) k connected components such that the number of self-edges on components is maximum.
Suppose that the graph G has a clique H of size $M = |V| - (k - 1)$; then the number of edges between any vertex $v \notin H$ and $u \in H$ is less than the edges between vertices inside H since in a clique there is an edge between every pair of vertices:

$$\sum_{\substack{u \in H \\ v \notin H}} w(e_{uv}) < \sum_{\substack{u \in H \\ v \in H}} w(e_{uv})$$

Then when we cut the graph G into k partitions with minimum cross-edges between components, the clique H must be one of these components.
Then the Maximum Clique decision problem can be reduced with polynomial time to the Min k-cut decision problem. Since Maximum clique problem is NP-complete, then The Min k-cut decision problem is NP-complete. □

For example: if we consider the graph G in (Figure 3.1), it has a clique of size 6 and when we find a minimum 3-cut, the clique must be one of the components.
The following proposition is due to Dahlhaus et al. (1992).



Figure(3-1) :Example of min 3-cut in graph has clique of size 6(the red edge is the min3-cut).

Proposition 3.1:

The Minimum k-cut optimization version problem (if k is a part of the input) is NP-hard.

Proof:

We know if the decision problem is NP-complete then the Optimization Problem is NP-hard.

And by theorem (3.1), the Min k-cut decision problem is NP-complete then the Min k-cut optimization problem is NP-hard. □

For planar graphs, Dahlhaus et al. (1992) showed that the optimization min k-cut problem, for a fixed k, can be solved in polynomial time.

For any graph and a fixed k, Goldschmidt and Hochbaum (1994) gave a polynomial algorithm with running time $O\left(n^{k^2}\right)$ for the Min k-cut optimization problem.

Table (3-1). Complexity of minimum k- cut problem

	Problem	**K variable**	**K fixed**
Complexity	Min k-cut optimization	NP-hard	P
	Min k-cut decision	NP-complete	P

3.2 Variants of the k cut problem

There are a lot of variants of the min k-cut problem; we give here the most important. We classify them into three main classes according to the change in the objective function (question) or the input.

3.2.1 Change in the Objective function

3.2.1.1 Ratio Cut Problem

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and positive integer k.

Question: Find a k-cut C that minimizes $\sum_{e \in C} w(e)/k$.

This problem was considered by Chvátal (1973). He proved following proposition:

Proposition (3.2):

The solution to the ratio problem is attained for k=2.

Proof:

Suppose Ω_2 is a min cut of the graph G = (V, E) and suppose that Ω_k is a min k-cut.

$w(\Omega_2) \leq w(V_i - \bigcup_{i \neq j} V_j)$, $1 \leq i \neq j, \leq k$.

$k\ w(\Omega_2) \leq w(V_1 - \bigcup_{1 \neq j} V_j) + w(V_2 - \bigcup_{2 \neq j} V_j) + \ldots + w(V_k - \bigcup_{k \neq j} V_j)$

$k\ w\ (\Omega_2) \leq 2\ w\ (\Omega_k)$

$\frac{w\ (\Omega 2)}{2} \leq \frac{w\ (\Omega k)}{k}$ □

3.2.1.2 Strength Cut Problem

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a positive integer k.

Question: Find a k-cut C that minimizes $\sum_{e \in C} w(e)/(k-1)$.

A strong polynomial algorithm was given by Cunningham (1985) with running time

O(n^4) to solve this problem.

3.2.1.3 Barahona Multi Cut Problem

In put: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and positive integer k.

Question: Find a k-cut C that minimizes $\sum_{e \in C} w(e) - k$.

Barahona (2000) has reduced this problem to minimizing a symmetric submodular function. He gives a polynomial algorithm to compute a convex hull of functions $f(k)$ (the intersection of all convex set containing $f(k)$)of an optimal solution of the problem with running time $O(n^4)$.

3.2.1.4 The Max k-Cut Problem

In put: Given an undirected graph G = (V, E), a nonnegative weight function w: E⟶R, and a positive integer k.

Question: Find a k-cut C that maximizes $\sum_{e \in C} w(e)$.

This problem is a generalization of the famous Max Cut problem.

The maximum cut problem is closely linked to the Max Bipartite Subgraph Problem: if the weights are positive, the Max Bipartite Subgraph Problem and the maximum cut problem are equivalent.

The Max Cut problem in general is NP-hard (Garey and Johnson 1979). But there are special cases for which the Max-Cut can be solved in polynomial time:

- In planar graphs: Orlova and Dorfman (1972) and Hadlock (1975) showed that the Max-Cut can be solved in polynomial time independently.
- If the graph is not contractible to K_5(K_5 can't be formed from the graph by deleting edges and vertices and by contracting edges), Barahona (1983) showed that the Max Cut problem can be solved in polynomial time.
- If the graph can be embedded in tours and the weights are equal to +1 or -1, Barahona (1981) showed that the Max Cut problem can be solved in polynomial

time.

- If the graph has only negative weights, McCormick and Rinaldi (2003) showed that the Max Cut can be solved in polynomial time by reducing the Max Cut problem to the Min-Cut problem with positive weights, then it can be solved, using flows, in polynomial time.

3.2.1.5 The max k-Un Cut Problem

In put: Given an undirected graph G = (V, E) and a nonnegative weight function w: E⟶R.

Question: Find a k-cut C that maximizes $\sum_{e \notin C} w(e)$.

This problem is equivalent to min k-cut problem, because maximizing the total self-edges weight is equivalent to minimizing the total cross-edges weight.

Since min k-cut is NP-hard then max k-uncut is NP-hard.

Proposition (3.4):

If approximation algorithm for min k-cut is α then the factor for max k-uncut is $\frac{\alpha}{\alpha-1}$.

Proof:

Consider graph G= (V, E) with |E| = m. Suppose OPT_Ω is an optimum solution for min k-cut, OPT_F is an optimum solution for max k-uncut.

$\sum_{e \notin C} w(e) + \sum_{e \in C} w(e) = m$, (self-edge: $\sum_{e \notin C} w(e)$, cross-edge: $\sum_{e \in C} w(e)$) then:

$OPT_F + OPT_\Omega = m$

$OPT_F = m - OPT_\Omega$

$OPT_\Omega \leq OPT_1 \leq \alpha\ OPT_\Omega$

$m – OPT_F \leq m - OPT_2 \leq \alpha\ OPT\Omega$

$m – OPT_F \leq \alpha\ (m – OPT_F)$

$-\alpha\ m \leq m – \alpha\ m \leq (1-\alpha)\ OPT_F$

$-\alpha\ m \leq (1-\alpha)\ OPT_F$

$\frac{\alpha}{\alpha-1}\ m \geq OPT_F$ □

3.2.2 Additional condition to the k-cut

3.2.2.1 Capacity Min k-cut problem

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, a positive integer k, and a set of capacities $\{s_1, s_2, \dots, s_k\}$ where $\sum_{i=1}^{k} s_i = |V|$.
Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with minimum total weight, such that each component V_i contain at most s_i vertices.
This problem is NP-complete even for k = 2 (Gary and Johnson (1976)), and there is no known approximation algorithm to solve this problem.

3.2.2.2 Minimum Multiway Cut Problem

In put: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, a positive integer k, and a subset of vertices $\mathrm{T} = \{t_1, t_2, \dots, t_k\} \subseteq V$ called terminals.
Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with minimum total weight, such that each component V_i contains exactly one terminal t_i .
When $k = 2$ (we have only two terminals: one source and one sink), then the multiway cut problem is equivalent to the st-cut problem which is solvable in polynomial time. The multiway cut problem is NP-hard for $k \geq 3$; due to Dahlhaus (1992).

3.2.2.3 The Minimum Steiner k-Cut Problem

The Minimum Steiner k-cut problem is a generalization of both the minimum k-cut problem and the minimum multiway cut problem. It is defined as follows:
In put: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, a positive integer r, a subset of vertices $\mathrm{T} = \{t_1, t_2, \dots, t_r\} \subseteq V$ called terminals, and a positive integer $k \leq r$.

Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with minimum total weight, such that each component V_i contains exactly one terminal t_i .

If $k = r$, we have the minimum multiway cut problem. If $\mathrm{T} = V$, we have the minimum k-cut.

3.2.2.4 Cardinality Min Cut Problems

- **k-card cut:**

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a positive integer k.

Question: Find the set of edges, having cardinally k, with minimum total weight, when deleted, partitions the graph into 2 components.

- **≥ k-card cut:**

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a positive integer k.

Question: Find the set of edges, has cardinally greater than or equal k, with minimum total weight, when deleted, partitions the graph into 2 components.

In general, for any graph, k-card cut and ≥k-card cut are NP-hard even for unweighted graph due to Bruglieri et al. (2003).

3.2.3 Other Variants

3.2.3.1 Capacitated Max k-Cut Problem

In put: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, a positive integer k, and a set of capacities $\{s_1, s_2, \dots, s_k\}$ where $\sum_{i=1}^{k} s_i = |V|$.

Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with maximum total weight, such that each component V_i contains at most s_i vertices.

3.2.3.2 Capacitated Max k-Un Cut Problem

In put: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, a positive integer k, and a set of capacities $\{s_1, s_2, \dots, s_k\}$ where $\sum_{i=1}^{k} s_i = |V|$.

Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with maximum total weight of self-edges, such that each component V_i contains at most s_i vertices.

Choudhury (2008) consider two integer linear programs and show that the integrality gap (ratio between the optimal solution to the linear programming relaxation and the optimal solution to the integer linear program) is not bounded.

3.2.3.3 Multiway Un Cut Problem

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, a positive integer k, and a subset of vertices $\mathrm{T} = \{t_1, t_2, \dots, t_k\} \subseteq V$ called terminals.

Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with maximum total weight of self-edges, such that each component V_i contains exactly one terminals t_i .

3.2.3.4 The k-route cut problem

Input: Given an undirected graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, an integer connectivity requirement k, and a collection $\{(s_1, t_1), \dots, (s_r, t_r)\}$ of source-sink pairs.

Question: Find k-cut $C(V_1, V_2, \dots, V_k)$ with minimum total weight, such that each (s_i, t_i) is disconnected, i.e., does not belong to the same component.

3.2.3.5 Directed multiway cut problem

Input: Given a directed graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, a nonnegative weight function $w: E \longrightarrow N$, and a set $K \subseteq V \times V$ of order pair of vertices of G.

Question: Find a k-cut $C(V_1, V_2, \dots, V_k)$ with minimum total weight, such that each component V_i contains exactly one terminal t_i .

Table (3-2). The complexity of minimum k- cut problem and some of its variants.

problem	Special graph	Special k	Complexity	Reference
Min k-cut	General graph		NP-hard	Dahlhaus et al. (1992).
Min k-cut	General graph	K=2	P	King et al. (1992).
Min k-cut	General graph	Fixed k	P	Goldschmidt and Hochbaum (1994)
Min k-cut	tree		p	
Ratio cut	General graph		P	Chvátal (1973)
Strength cut	General graph		P	Cunningham (1985).
Barahona multi cut	General graph		P	Barahona (2000).
Max k-cut	General graph		NP-hard	Garey and Johnson (1979).
Max k-cut	planar		p	Orlova and Dorfman (1972) and Hadlock (1975).
Max k-cut	not contractible to K_5		P	Barahona (1983).
Max k-cut	graph can be embedded in tours and the weights are equal to +1 or -1		P	Barahona (1981).
Max k-cut	graph has only negative weights		P	McCormick and Rinaldi (2003)
Max k-un cut	General graph		NP -hard	Choudhury (2004)

Capacity Min k-cut			NP-complete	Gary and Johnson (1976)
Multiway cut	General graph	K > 2	NP -hard	Dahlhaus (1992).
Multiway cut		K=2	P	King et al. (1992).
Multiway cut	tree		P	Erdős et al. (1994)
Cardinality k-cut	General graph		NP -hard	Bruglieri et al. (2003).
Cardinality k-cut	Tree		P	Bruglieri et al. (2003).

Chapter 4: Some Applications

The minimum cut problem has several applications. Picard and Queyranne (1982) survey applications including graph partitioning problems, the study of project networks, and partitioning items in a database.

An important application of graph partitioning is data clustering using a graph model - the pairwise similarities between all data objects form a weighted graph adjacency matrix that contains all necessary information for clustering.

The problem of determining the connectivity of a network arises frequently in issues of network design and network reliability, and exploits an extremely tight connection between minimum cuts and network reliability.

Minimum cut computations are used to find the subtour elimination constraints that are needed in the implementation of cutting plane algorithms for solving the traveling salesman problem. Padberg and Rinaldi (1990) and Applegate (1992) have reported that solving min-cut problems was the computational bottleneck in their state-of-the-art cutting plane based TSP algorithm, as well as other cutting-plane based algorithms for combinatorial problems whose solutions induce connected graphs.

Minimum cut problems also play an important role in large-scale combinatorial optimization, finite elements and, VLSI circuit partitioning, which is a key step in VLSI CAD.

Minimum cut problems arise in the design of compilers for parallel languages (Chaterjee et al. 1996). Consider a parallel program that we are trying to execute on a distributed memory machine. In the alignment distribution graph for this program, vertices correspond to program operations and edges correspond to flows of data between program operations.

More detailed applications are presented here below: spin glass models, unconstrained 0-1 quadratic programming and image segmentation problem.

4.1. Spin glass model

It is a very interesting problem in numerical physics ground state problem. This problem is NP-hard in general. A spin-glass is a disordered magnet, where the magnetic spin of the component atoms is not ranged in a regular pattern. In normal magnets, magnetic moments of interacting atoms align in one particular fashion. The interactions are either ferromagnetic (the spins align in the same direction) or anti-ferromagnetic (the spins align in the opposite directions). In disordered magnet, we may have several types and strengths of interactions at the same time. Figure 4.1

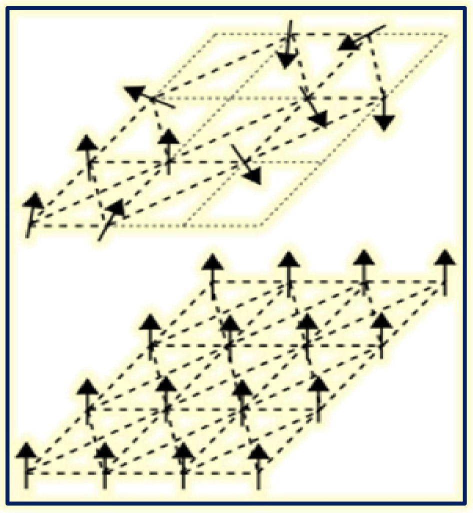


FIGURE (4-1): Graphic representation of the random spin structure of a spin glass (top) and the ordered one of a ferromagnet (bottom). (Wikipedia 2011)

Between every atom there is an interaction energy given by: $H_{ij} = - J(R)\, S_i\, S_j$, where S_i: is the magnetic spin of the atom i, and $J(R) = A\, \frac{\cos(Dr_{ij})}{B^3 (r_{ij})^3}$ such that A, B and C depending on the material, r_{ij} is the range between the atom i and j.

Physics Model:

Sherrington and Kirkpatrick (1975) proposed a simple spin-glass model to construe the physical properties of the systems, consisting of interacting spins. This famous model can be defined as follows:

Assume that the spins are existing at the nodes of net (usually square or cubic) and each spin atom is represented by Vector, take the values: $S_i = 1, -1$ ($i=0,1, 2, \ldots, n$).

Assume the interactions between the spins happen for the neighbors.

The energy function of the system is given by:

$$H = - \sum_{(i,j)\in L} J_{ij} S_i S_j \quad \ldots (4.1) \text{ (Hamilton function)}$$

The ground state problem is to find minimum energy system

i.e. min $\{H(S) = \sum_{(i,j)\in L} J_{ij} S_i S_j$, where S is spin configuration$\}$.

Graphic Model:

Spin-glass can be reduced to the MAX-CUT problem. (Barahona 1987)

We can be modeled the problem on graph G= (V, E) where the vertices correspond to the spins, and if there is an interaction between the spins S_i, S_j the corresponding vertices are connected by an edge S_i S_j. We can associate the weight: $w_{ij} = - J_{ij}$, then the ground state problem is equivalent to the problem:

$$\min \{H(S) = \sum_{(i,j)\in E} w_{ij} S_i S_j : S_i \in \{+1, -1\}, i \in V\}.$$

This problem is to be established an assignment of +1 and -1 to the vertices of the graph such that $\sum_{(i,j)\in E} w_{ij} S_i S_j$ is minimum. Figure (4-2).

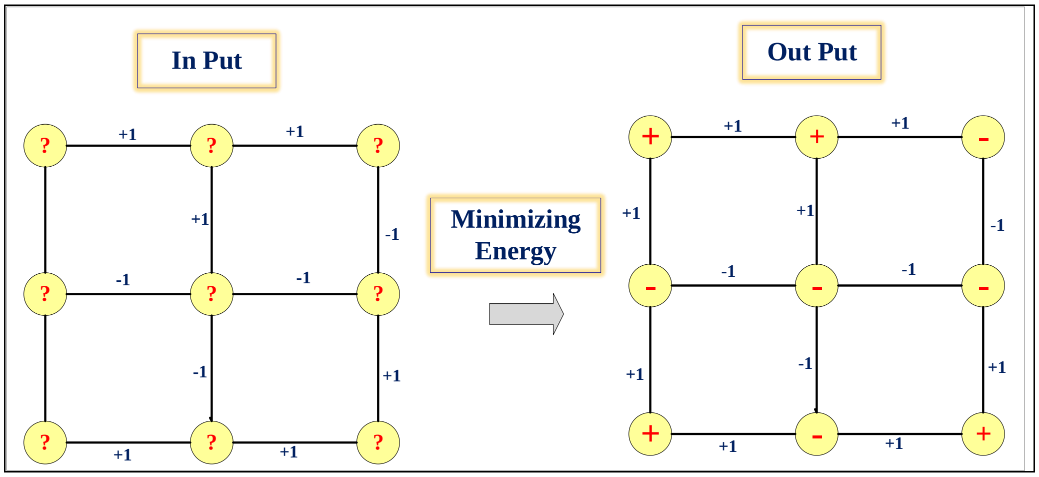


Figure (4-2): Spin glass problem

Every assignment induces a partition of the nodes in graph into node sets V^+ and V^-,where : $V^+ = \{i \in V: S_i = +1\}$, $V^- = \{i \in V: S_i = -1\}$.

$$\min \sum_{(i,j)\in E} w_{ij}\, S_i\, S_j = \min \left[\sum_{i,j\in V^+} W_{ij} + \sum_{i,j\in V^-} W_{ij} + \sum_{\substack{i\in V^+ \\ j\in V^-}} -W_{ij} \right],$$

$$\min \sum_{(i,j)\in E} w_{ij}\, S_i\, S_j = \min \left[\left(\sum_{i,j\in V^+} W_{ij} + \sum_{i,j\in V^-} W_{ij} + \sum_{\substack{i\in V^+ \\ j\in V^-}} -W_{ij}\right) + \left(\sum_{\substack{i\in V^+ \\ j\in V^-}} W_{ij} + \sum_{\substack{i\in V^+ \\ j\in V^-}} -W_{ij}\right) \right],$$

$$\min \sum_{(i,j)\in E} w_{ij}\, S_i\, S_j = \min \left[\left(\sum_{i,j\in V^+} W_{ij} + \sum_{i,j\in V^-} W_{ij} + \sum_{\substack{i\in V^+ \\ j\in V^-}} W_{ij}\right) + \left(\sum_{\substack{i\in V^+ \\ j\in V^-}} -W_{ij} + \sum_{\substack{i\in V^+ \\ j\in V^-}} -W_{ij}\right) \right],$$

$$\min \sum_{(i,j)\in E} w_{ij}\, S_i\, S_j = \min \left[\sum_{i,j\in E} W_{ij} - 2\sum_{\substack{i\in V^+ \\ j\in V^-}} W_{ij} \right],$$

$$\min \sum_{(i,j)\in E} w_{ij}\, S_i\, S_j = \min \left[C - 2\sum_{\substack{i\in V^+ \\ j\in V^-}} W_{ij} \right],$$ where $C = \sum_{i,j\in E} W_{ij}$ is a constant.

To find $\min \left[C - 2\sum_{\substack{i\in V^+ \\ j\in V^-}} W_{ij} \right]$ is equivalent to find maximum$\left[\sum_{\substack{i\in V^+ \\ j\in V^-}} W_{ij} \right]$.

4.2 Unconstrained 0-1 Quadratic Programming

Consider the Quadratic 0-1 program as follows:

Minimum $\left\{ f(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} x_i x_j + \sum_{i=1}^{n} c_i x_i : x \in \{0,1\}^n \right\}$. ----- (4.2)

This problem in general is NP- hard. It can be reduced to Max Cut problem (Ameur et al 2001).

Let $s_i = 2x_i - 1$ then $x_i = \frac{s_i+1}{2}$, if $x_i = 0$ then $s_i = -1$, and if $x_i = 1$ then $s_i = 1$, hence if $x_i = \{0,1\}$ then $s_i = \{+1, -1\}$.

$$f(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} \left(\frac{s_i+1}{2}\right)\left(\frac{s_j+1}{2}\right) + \sum_{i=1}^{n} c_i \left(\frac{s_i+1}{2}\right) : s_i \in \{1,-1\}^n,$$

$$f(x) = \frac{1}{4} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} q_{ij} (s_i s_j + s_i + s_j + 1) + \frac{1}{2} \sum_{i=1}^{n} c_i s_i + \frac{1}{2} \sum_{i=1}^{n} c_i : s_i \in \{1,-1\}^n,$$

$$f(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\frac{1}{4} q_{ij}\right) s_i s_j + \sum_{i=1}^{n} \left[\frac{1}{4} \left\{ \sum_{j=1}^{i-1} q_{ij} + \sum_{j=i+1}^{n} q_{ij} \right\} + \frac{1}{2} c_i \right] s_i +$$
$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{4} q_{ij} + \frac{1}{2} \sum_{i=1}^{n} c_i : s_i \in \{1,-1\}^n,$$

And by setting: $w_{ij} = \frac{1}{4} q_{ij}$, $w_{0j} = \frac{1}{4} \left\{ \sum_{j=1}^{i-1} q_{ij} + \sum_{j=i+1}^{n} q_{ij} \right\} + \frac{1}{2} c_i$, $c_1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{4} q_{ij} + \frac{1}{2} \sum_{i=1}^{n} c_i$, $s_0 = 1$.

We obtain the following equivalent problems:

Minimum $\left\{g(s) = \sum_{i=0}^{n-1}\sum_{j=i+1}^{n} w_{ij}s_is_j : s_i \in \{1,-1\}^{n+1}\right\}$. ---- (4-3)

And we can reduce (4-3) to the Max Cut problem by using the spin glass model.

4.3 Image Segmentation problem

A fundamental problem in computer vision is that of segmenting an image into coherent regions. A basic segmentation problem is that of partitioning an image into a foreground and a background: assign each pixel in the image as belonging to the foreground or the background. Figure (4-3).

Figure (4-3): In the right input Image, in the left optimal segment. (Boykov and Jolly (2011)

The main contribution to the problem with graphs was proposed by Boykov et al. (2001), inspired by a previous work by Greig et al. (1989).

Let V be the set of pixels in an image, and E be the set of pairs of adjacent pixels. V and E produce an undirected graph G (V, E).

Each pixel i has a likelihood $a_i > 0$ that it belongs to the foreground and a likelihood $b_i > 0$ that it belongs to the background. These likelihoods are specified in the input to the problem.

We want the foreground/background boundary to be smooth: for each pixels i and j , there is a separation penalty $p_{ij} \geq 0$ for placing one of them in the foreground and the other in the background.

Image Segmentation problem

Given: Pixel graphs G (V, E), likelihood functions a, b: V $\rightarrow R^+$, and a penalty function p: E $\rightarrow R^+$.

Question: (Find the optimum labelling) Partition the pixels into two sets A be the set of pixels assigned to foreground and B be the set of pixels assigned to background such that:

Maximize $q(A,B) = \sum_{i\in A} a_i + \sum_{i\in B} b_i - \sum_{\substack{ij\in E \\ |A\cap\{i,j\}|=1}} p_{ij}$.

Rewrite $q(A,B)$ as follows:

$q(A,B) = \sum_{i\in V}(a_i+b_i) - \sum_{i\in A} b_i - \sum_{i\in B} a_i - \sum_{\substack{ij\in E \\ |A\cap\{i,j\}|=1}} p_{ij}$.

Since $C = \sum_{i\in V}(a_i+b_i)$ is constant then:

$$q(A,B) = C - \left[\sum_{i\in A} b_i + \sum_{i\in B} a_i + \sum_{\substack{ij\in E \\ |A\cap\{i,j\}|=1}} p_{ij}\right].$$

Maximizing $q(A,B)$ is equivalent to minimizing:

$$\tilde{q}(A,B) = \sum_{i\in A} a_i + \sum_{i\in B} b_i + \sum_{\substack{ij\in E \\ |A\cap\{i,j\}|=1}} p_{ij}$$

To minimize $\tilde{q}(A,B)$, we can formulate it as a Min-Cut problem by constructing a network where the source (dummy vertex s) is connected to all the pixels with likelihood function a_i , and the sink (dummy vertex t) is connected to all the pixels with likelihood function b_i; two edges (i,j) , and (j,i) with p_{ij} penalty function are added between two adjacent pixels.

An st-cut then represents pixels assigned to the foreground in A and pixels assigned to the background in B. Figure (4-4).

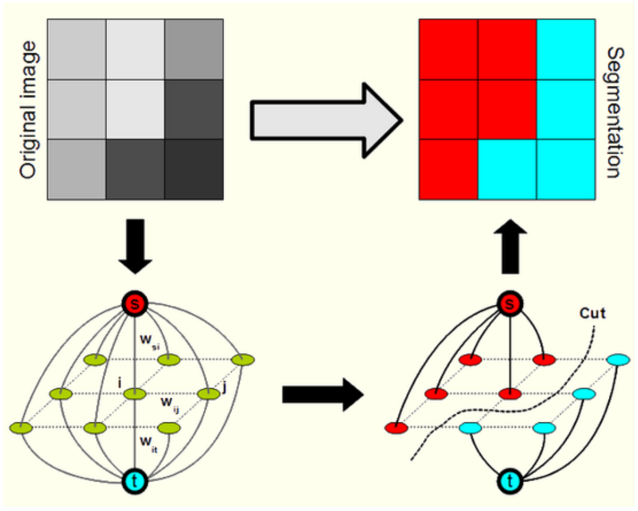


Figure (4-4): Using the min-cut approach for image segmentation. Boykov and Jolly (2011) a.

Chapter 5: Special Instances of MKCP

In this chapter, we introduce some special instances of the minimum k-cut problem and some of its variants.

5.1. Minimum k-cut for fixed k

A natural approach for solving MKCP with a fixed k is the greedy approach.

5.1.1. Greedy Algorithm

A greedy algorithm is an algorithmic method that builds up a solution step by step, and at every step, we can make a choice that looks best at the moment, it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution. We get the optimal solution of the complete problem at the end of the algorithm.

Greedy algorithms are used for optimization problems.

If a greedy algorithm can solve a problem, then it generally becomes the best method to solve that problem as the greedy algorithms are in general more efficient than other techniques. But greedy algorithms cannot always be applied.

Greedy approach for k-cut:

The base of this algorithm is to apply the minimum st- cut using the max-flow min-cut theorem repeatedly.

Consider a graph G of size n, we will have $\binom{n}{k}$ subsets of size k.

For each subset $V_j = \{v_1, v_2, \dots, v_k\};\ (1 \le j \le \binom{n}{k})$, choose one vertex v_i $(1 \le i \le k)$ and partition it from V using (v_i-V) min cut and again choose another vertex v_i $(1 \le i \le k-1)$ and partition it from V using minimum (v_i-V) min cut, repeat this process to get k components in each one there exactly one vertex v_i from V_j.

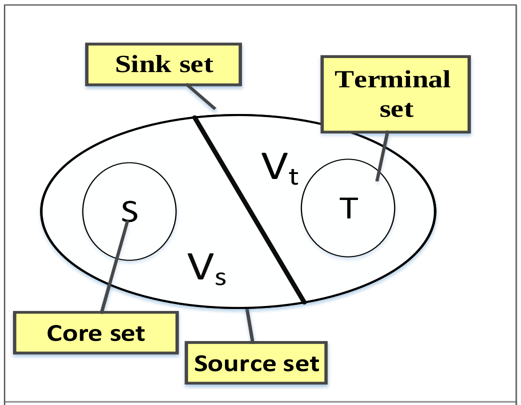
From $\{v_1, v_2, \dots, v_k\}$ we pick v_i and apply (v_i-V) min cut it cost (k) k-cut and from $\{v_1, v_2, \dots, v_{k-1}\}$ we pick v$_i$ and apply (v$_i$-V) min cut it cost (k-1) k-cut

and from $\{v_1, v_2, \dots, v_{k-2}\}$ we pick v$_i$ and apply (v$_i$-V) min cut it cost (k-2) k-cut

we produce $k.(k-1).(k-2)\dots(2)(1) = k!$ k-cut, the running time of this algorithm will be $O(n^k)$.

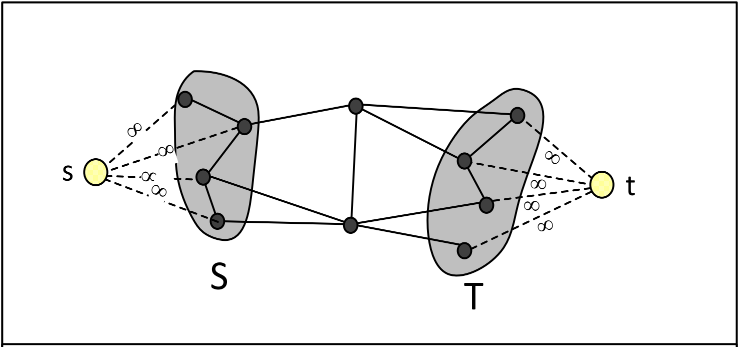
This algorithm fails even in a planar graph. We can develop this algorithm by exchange min (s-t) cut by min (S-T) cut such that S and T have more than one vertex, S is called core set, and T is called terminal set.

A min (S- T) cut is the set of edges of minimum total weight which partition graph into two connected components V_s (source set) and V_t (sink set), such that $V_s \cap V_t = \phi$, $V_s \cup V_t = V$, and the core set is sub set of source set and the terminal set is sub set of sink set. Figure (5-1).



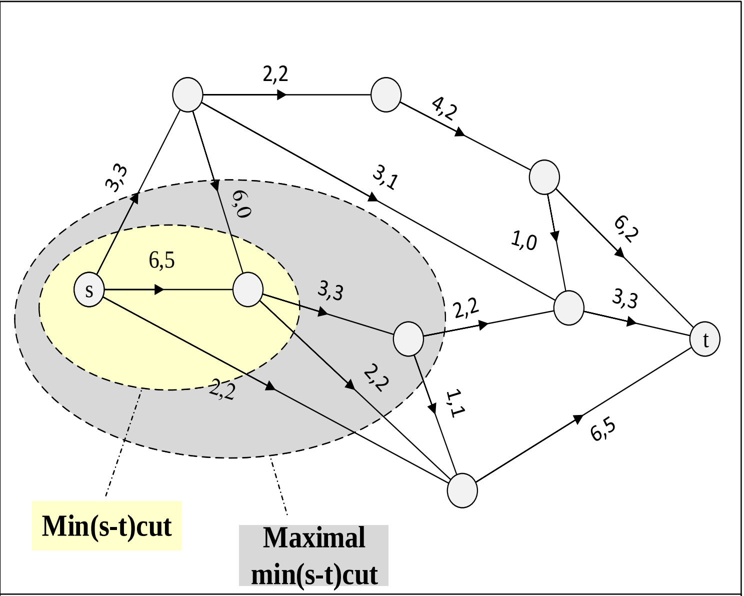
Figure(5-1)

The min (S-T) cut can be found in polynomial time by reducing it to min (s-t) cut which finds by Max flow – Min cut theorem, we joined every vertex in the core set S to pretend point s by infinite weight arcs, and joined every vertex in the terminal set T to pretend point t by infinite weight arcs, and then we can find min (s-t) cut. Figure (5-2).



Figure(5-2):reduce min(S-T)cut to min (s-t)cut

A maximal min (S-T) cut is a min (S-T) cut with maximal source, to find it we will find all the min (S-T) cut and pick the one with maximal source, obviously, we reduce min (S-T) cut to min (s-t) cut then apply Max flow algorithm, and in the final residual graph find all nodes reachable from the terminal which is satisfies the min cut, the remaining nodes is a maximal source. Figure (5-3).



Figure(5-3): Example of maximal minimum (S-T)cut

5.1.2. A polynomial algorithm

Goldschmidt and Hochbaum (1994) give a polynomial algorithm to find the minimum k-cut for a fixed k with running time $O(n^{k^2}\ T(n,m))$, where T(n, m) is the (best) running time complexity for the minimum (s, t) cut algorithm. The idea for finding an optimal solution is to find a maximal minimum (S-T) cut (which is unique), i.e., a minimum (S-T) cut with a maximal source set, the core set S having at most k-2 vertices, and the terminal set T having at most k-1 vertices.

The following result is an important proposition for the minimum k–cut problem. It is a basic result for a lot of algorithms.

Proposition (5-1)

For any graph G, let C be a minimum k-cut in G = (V, E) separating V into k components $V_1, \ldots, V_k$, and let $C_i = C_i(V_i, \overline{V}_i) \subseteq C$ be the set of edges separating V_i from $\overline{V}_i$, then:

$w(C) = \frac{1}{2}\sum_{i=1}^{k} w(C_i)$.

Proof:

$w(C_i) = w[C_i(V_i, \overline{V}_i)] = \sum_{\substack{j=1 \\ i\neq j}}^{k} w\left[C_i\left(V_i, V_j\right)\right]$,

$\sum_{i=1}^{k} w(C_i) = \sum_{i=1}^{k} \sum_{\substack{j=1 \\ i\neq j}}^{k} w\left[C_i\left(V_i, V_j\right)\right] = \sum_{\substack{i=1,j=1 \\ i<j}}^{k} w\left[C_i\left(V_i, V_j\right)\right] + \sum_{\substack{i=1,j=1 \\ i>j}}^{k} w\left[C_i\left(V_i, V_j\right)\right]$,

Since $w(C) = \sum_{\substack{i=1,j=1 \\ i<j}}^{k} w\left[C_i\left(V_i, V_j\right)\right] = \sum_{\substack{i=1,j=1 \\ i>j}}^{k} w\left[C_i\left(V_i, V_j\right)\right]$, then:

$\sum_{i=1}^{k} w(C_i) = 2w(C)$.

$w(C) = \frac{1}{2}\sum_{i=1}^{k} w(C_i)$. □

Let C be a minimum k-cut that partitions the graph into k components $V_1 \ldots, V_k$, and C_i be the subset of the cut C, denoted $C_i(V_i, \overline{V_i}\ )$, that have one endpoint in V_i and the other end in $\overline{V_i}$ $(1 \leq i \leq k)$.

We sort these components: $w(C_1) \leq w(C_2) \ \leq \cdots \ \leq w(C_k)$.

If we consider all min k-cuts, we will pick the one, that generates V_1 as a maximal component, i.e., there is no min k-cut $\check{C}$, such that the cut $\check{C}_i$ is a subset of $\check{C}$, and partitions the graph into $\left(\breve{V}_i, \overline{\breve{V}_i}\right)$ and $w(\check{C}_1) \leq w(\check{C}_i)$ for every i , and $V_1 \subseteq \breve{V}_1$.

Theorem (5-1):

For any graph G, for k > 3, if V_1 contains at least k-2 vertices then there exist a set S$\subseteq V_1$ with $|S|$=k-2, and there exist a set T= $\{t_1, \ldots, t_{k-1}\}$ where $t_i \in V_i$ $(2 \leq i \leq k)$ such that C_1 is the maximal min (S-T) cut.

Proof:

Suppose that C_1 is not a min (S-T) cut. We will prove that there is a contradiction with the fact that C is a minimum k- cut. (See Figure (5-4)).

- First, we prove three statements:

Statement 1: for any min (S, T) cut, if S is contained in V_1 then the source set $V_s \subseteq V_1$.

Suppose that there exists a min (S, T) cut denoted Ć, S$\subseteq V_1$ and $V_s \subsetneq V_1$,. (See Figure (5-5)).

Let the set of edges $C_{1x} \subseteq C_1$, the set of edges $Ć_x \subseteq Ć$, such that for every edge uv in C_{1x}, u and v belong to V_s, for every edge uv in $Ć_x$, u and v belong to $V - V_1$. (See Figure (5-6)).

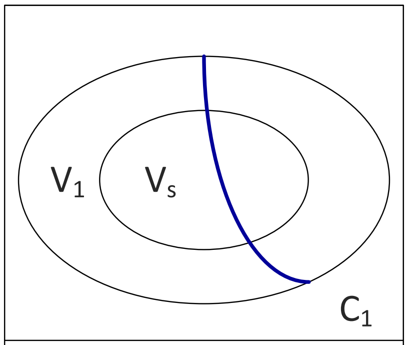


Figure (5-4)

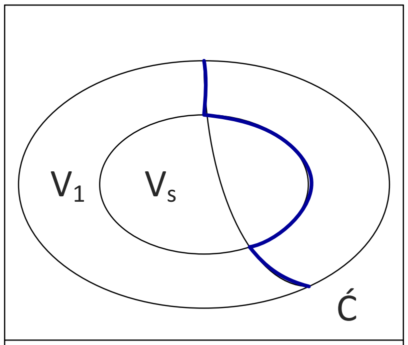


Figure (5-5)

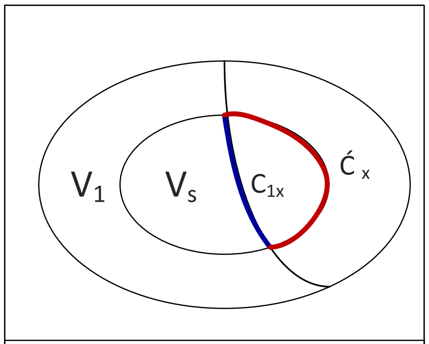


Figure (5-6)

We reverse the two choices: $w(C_{1x}) < w(Ć_x)$ and $w(C_{1x}) \geq w(Ć_x)$.

➢ If $w(C_{1x}) < w(Ć_x)$, in the cut Ć, if we replace the section of the cut $Ć_x$ by C_{1x} , we will get the cut $\left[\left(Ć \cup C_{1x}\right) - \left(Ć_x\right)\right]$ and it min (S, T) cut, hence it is a contradiction with claim Ć is min (S-T) cut.

➢ If $w(C_{1x}) \geq w(Ć_x)$, assume that $\mathbb{C}$ is a cut that partitions G in to k components $\ddot{V}_1, \ddot{V}_2, .., \ddot{V}_k$, such that: $\ddot{V}_1 = V_1 \cup V_s$, $\ddot{V}_2 = V_2 - V_s$, …, $\ddot{V}_k = V_k - V_s$. Let $\mathbb{C}_i \subseteq \mathbb{C}$

is the cut partitioning $\grave{\grave{\mathrm{V}}}_i$ from $\overline{\overline{\grave{\grave{\mathrm{V}}}}}_i$.

To get a feasible k- cut, every component should be not empty.

For every i ($2 \leq i \leq k$) $t_i \in V_i$, and $t_i \notin V_s$ since $T \cap V_s = \phi$, then $V_i - V_s \neq \phi$.

Let $\mathrm{C_x} \subseteq \mathrm{C}$ be the set of edges, such that for every edge uv in $\mathrm{C_x}$, u and v belong to V_s.

We can replace the cut $\mathrm{C_x}$ by $\acute{\mathrm{C}}_\mathrm{x}$, hence: $\mathbb{C} = (\mathrm{C} \cup \acute{\mathrm{C}}_\mathrm{x}) - \mathrm{C_x}$.

I. The set of edges partitioning the connected component $\boldsymbol{V_i, V_j}$ is equivalent to the set of edges partitioning the connected component $\grave{\grave{\mathrm{V}}}_{\boldsymbol{i}}, \grave{\grave{\mathrm{V}}}_{\boldsymbol{j}}$,

$C\left(V_i, V_j\right) = C\left(\grave{\grave{\mathrm{V}}}_i, \grave{\grave{\mathrm{V}}}_j\right)$ for $i \neq j$, $i, j \geq 2$.

Since $\mathrm{C_{1x}} \subseteq C_\mathrm{x}$ and $w(\mathrm{C_{1x}}) \geq w(\acute{\mathrm{C}}_\mathrm{x})$, then $w(\mathrm{C_x}) \geq w(\acute{\mathrm{C}}_\mathrm{x})$.

If $w(\mathrm{C_x}) > w(\acute{\mathrm{C}}_\mathrm{x})$ and since $\mathbb{C} = (\mathrm{C} \cup \acute{\mathrm{C}}_\mathrm{x}) - \mathrm{C_x}$, then $w(\mathbb{C}) < w(\mathrm{C})$ and this is a contradiction with that C is a min k-cut.

Hence $w(\mathrm{C_x}) = w(\acute{\mathrm{C}}_\mathrm{x})$ then $C_x = C_{1x}$, then:

$C\left(V_i, V_j\right) \cap C_x = \phi$ for $i \neq j$, $i, j \geq 2$. [5-1]

If there exist an edge e $\in C\left(V_i, V_j\right) \cap \acute{\mathrm{C}}_x$ then:

$w(\mathbb{C}) = w((\mathrm{C} \cup \acute{\mathrm{C}}_\mathrm{x}) - \mathrm{C_x}) \leq w(\mathrm{C}) + \mathrm{w}(\acute{\mathrm{C}}_\mathrm{x}) - \mathrm{w}(\mathrm{C_{1x}}) - w(e)$

Since: $\mathrm{w}(\acute{\mathrm{C}}_\mathrm{x}) = \mathrm{w}(\mathrm{C_{1x}})$ then $w(\mathbb{C}) \leq w(\mathrm{C}) - w(e)$

Since $w(e) > 0$ then $w(\mathbb{C}) < w(\mathrm{C})$ and it is a contradiction with C is a min k-cut.

then $C\left(V_i, V_j\right) \cap \acute{\mathrm{C}}_\mathrm{x} = \phi$ for $i \neq j$, $i, j \geq 2$. [5-2]

from [5-1] and [5-2]: $C\left(V_i, V_j\right) = C\left(\grave{\grave{\mathrm{V}}}_i, \grave{\grave{\mathrm{V}}}_j\right)$ for $i \neq j$, $i, j \geq 2$.

II. The total weight of the set of edges partitioning the connected component V_1, V_i is equivalent to the total weight of the set of edges partitioning the connected component $\grave{\grave{\mathrm{V}}}_{\boldsymbol{1}}, \grave{\grave{\mathrm{V}}}_{\boldsymbol{i}}$, $w(C\ (V_1, V_i)) = w(C\left(\grave{\grave{\mathrm{V}}}_1, \grave{\grave{\mathrm{V}}}_i\right))$ for, $i \geq 2$.

It is enough to prove that $w(C\ (V_1, V_i \cap V_s)) = w(C\ (V_i, V_i \cap V_s))$, and we know that $w(C\ (V_1, V_i \cap V_s) \geq w(C\ (V_i, V_i \cap V_s))$, or $w(C\ (V_1, V_i \cap V_s) \leq w(C\ (V_i, V_i \cap V_s))$, since if it is not, we have a contradiction with our claim C is a min k-cut.

By **I** and **II**: for every i: $w(\mathbb{C}_i) = w(\mathrm{C}_i)$ and then there exist a min k cut $\mathbb{C}$, $w(\mathbb{C}_1) \leq w(C_i)$, $i \geq 2$, and $V_1 \subset \grave{\grave{\mathrm{V}}}_1$, which contradicts the maximality claim. It follows that, for any min (S, T) cut, if $S \subset V_1$, $|S| = k-2$, then the source set $V_s \subseteq V_1$. □

If $|V_1| = k-2$ then $S = V_1 \subseteq V_s$, hence $V_1 = V_s$ and $C_1 = \acute{C}$.

Statement 2: 1f $|V_1| > k-2$ and there exist C_2 minimum (S, T) cut, such that $C_2 \neq C_1$ then $w(C_2) < w(C_1)$.

We prove this statement by contradiction. Suppose that w $(C_2) \geq w(C_1)$. Let V_{s2} be the source set of the cut C_2 then from statement 1, $V_{s2} \subseteq V_1$.

If $V_{s2} \subset V_1$, and w $(C_2) \geq w(C_1)$, then C_1 does not have a maximal source cut, and it is a contradiction. □

Statement 3: If $|V_1| > k-2$ and every minimum (S, T) cut have weight strictly less than $w(C_1)$, then it generates min k- cut have total weight less than the cut C.

Consider every possible core set $\{s_1, \dots, s_{k-2}\}$ in V_1 pick the set $\{s_1, \dots, s_{k-2}\}$ such that min ($\{s_1, \dots, s_{k-2}\} - T$)cut is of maximum weight. From statement 1, the source set of this cut is a proper subset of V_1, then there exists at least s_0 in V_1 not belonging to the source set. Now consider the cut C^{s_i} a min ($\{s_0, s_1, \dots s_{i-1}, s_{i+1}, \dots, s_{k-2}\} - T$) cut. The size of the core set of this cut is k-2. Let the source set of the cut C^{s_i} is V_{s_i}, and from statement 1, $V_{s_i} \subset V_1$. Let the cut $C_{S_i} \subset C^{s_i}$ be the set of edges with both end points in V_1.and $\overline{V}_{s_i} = V_1 - V_{s_i}$

Claim 1: $s_i \notin V_{s_i}$, for every i = 0,1, ..., k-2.

By using the shrink producer on V_1, we will get C_1, and V_1 satisfies claim 2.

Claim 2: $w(C_{S_i}) < w(C_1(\overline{V}_{s_i}))$

Let S_i be the component contain s_i, $C_1$is the output of the shrink producer, and C^* be the cut partitioning the graph into S_0, S_1, …,S_{k-2}, $V - \left(\bigcup_{i=0}^{k-2} S_i\right)$.

Claim 3: $w(C^*) < 2w(C_1)$.

By proposition (5-1): $w(C) = \frac{1}{2}\sum_{i=1}^{k} w(C_i) \geq \frac{k}{2}\, w(C_1)$.

By claim 3: $w(C) = \frac{1}{2}\sum_{i=1}^{k} w(C_i) \geq \frac{k}{2}\, w(C_1) = \frac{k}{2}\,\frac{2}{2} w(C_1) > \frac{k}{4} w(C^*)$,

hence $w(C) > \frac{k}{4} w(C^*)$, and at this moment the cut C^*separate graph in to k components and has a weigh less than the optimal one, which is a contradiction. □

The theorem (5-1) consider the maximal minimum (S-T) cut for k $>$ 3 in the following theorem we will consider the maximal minimum (S-T) cut for $k = 3$.

Let C be the minimum k-cut that partition graph into 3 connected components V_1, V_2, V_3 , and let C_i be the subset of the cut C that $C_i(V_i, \overline{V_i}$) denoted the set of edges have one endpoint in V_i and the other end in V_i complement ($1 \le i \le 3$), we sort this component: $w(C_1) \le w(C_2) \le w(C_3)$.

If we consider all min 3-cuts, we will pick the one, that generates V_1 maximal component.

Theorem (5-2):

For any graph G, for k= 3, if V_1 have size at least 2 vertices, then there exist a set S⊆ V_1 with $|S|$=2, and there exist a set T= $\{t_1, t_2\}$ where $t_i \in V_{i+1}$ ($1 \le i \le 2$) such that $C_1$is the maximal min (S-T) cut.

Proof:

let C_1 is not a min (S-T) cut and then we will have contradiction with C is minimum 3-cut.

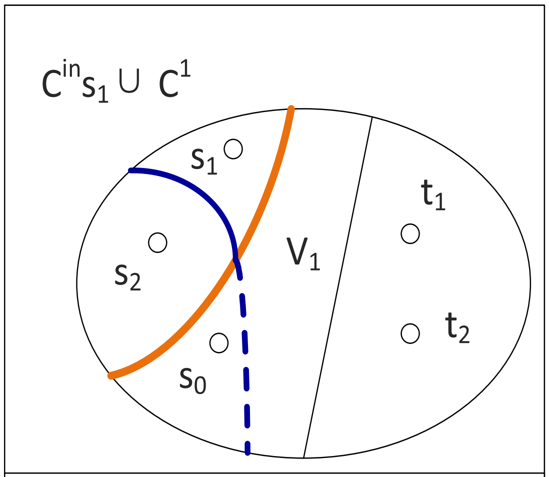
Suppose there exist min (S, T) cut denoted Ć, $S = \{s_1, s_2\}$ contend in V_1 and $T = \{t_1, t_2\}, t_1 \in V_2$ *and* $t_2 \in V_3$, if we enumerate all min (S-T) cut, we pick the cut Ć such that w (Ć) be maximum one.

By statement 1 of theorem (5-1), $V_s \subset V_1$, then there exist $s_0 \in (V_1 - V_s) \subseteq \overline{V_s}$. Let C_{s_1}be min ($\{s_2, s_0\} - \{t_1, t_2\}$)cut, and let V_{s_1} be maximal source set of the cut C_{s_1}, by claim1: $s_1 \notin V_{s_1}$, then the cut Ć ∪ C_{s_1} generates a 4-cut, the cut Ć make two connected componentsV_s and $\overline{V_s}$, and the cut C_{s_1} make two connected components V_{s_1} and $\overline{V}_{s_1}$, since s_2 in the core set of C_{s_1}, then $s_2 \in V_{s_1}$,and since $s_1 \notin V_{s_1}$th en $s_1 \in \overline{V}_{s_1}$ then we will have 4 connected components: $V_{s_1} \cap V_s$, $\overline{V}_{s_1} \cap V_s$, $V_{s_1} \cap \overline{V_s}$, $\overline{V}_{s_1} \cap \overline{V_s}$ and every components not empty: $s_2 \in V_{s_1} \cap V_s$, $s_1 \in \overline{V}_{s_1} \cap V_s$, $s_0 \in V_{s_1} \cap \overline{V_s}$, $\{t_1, t_2\} \subseteq \overline{V}_{s_1} \cap \overline{V_s}$.

Let the cut $C^{in}{}_{s_1}$ be part of the cut $C_{s_1}$with all end points in V_s, and the cut $C^{out}{}_{s_1}$ be part of the cut C_{s_1} with all end points in $\bar{V}_s$

The cut $C_2 = C^{in}{}_{s_1} \cup \acute{C}$ generates a 3-cut: $V_{s_1} \cap V_s$, $\bar{V}_{s_1} \cap V_s$, $\bar{V}_s$. Figure (5-7)

The cut $C_3 = C^{out}{}_{s_1} \cup \acute{C}$ generates a 3-cut: V_s ,$V_{s_1} \cap \bar{V}_s$, $\bar{V}_{s_1} \cap \bar{V}_s$. Figure (5-8)



Figure(5-7)

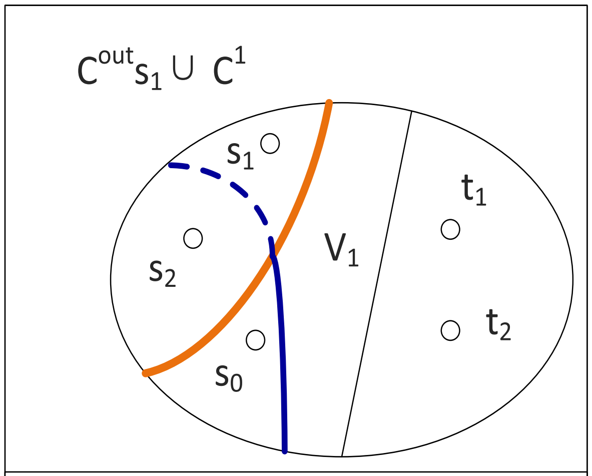


Figure (5-8)

And from assumption ($\acute{C}$ be maximum min (S-T) cut and C_{s_1}be min (S-T)) $w(C_{s_1}) \leq w(\acute{C}) < w(C_1)$. If we assume $w(C^{in}{}_{s_1}) \leq w(C^{out}{}_{s_1})$ then

$2w(C^{in}{}_{s_1}) \leq w(C^{in}{}_{s_1} \cup C^{out}{}_{s_1}) \leq w(C_{s_1}) < w(C_1)$. Hence $w(C^{in}{}_{s_1}) < \frac{1}{2}w(C_1)$, and

then $w(C_2) = w(C^{in}{}_{s_1} \cup \acute{C}\ )\ ) < \frac{1}{2}w(C_1) + w(C_1) = \frac{3}{2}\ w(C_1)$

By proposition (5-1): $w(C) = \frac{1}{2}\sum_{i=1}^{3} w(C_i) \geq \frac{3}{2}\ w(C_1) > w(C_2)$

Then the cut C_2 separates the graph into 3 connected components and have a weight less than the optimal one, it’s a contradiction. □

The following algorithm is due to Goldschmidt and Hochbaum (1994).

5.1.3. The min k- cut Algorithm for fixed k (k=3)

Input: A graph G = (V, E), and k=3. Let $W_\circ = \infty$.

Step 1:

For i ($1 \leq i \leq |V|$), let $V_1 = \{v_i\}$, such that $v_i \in V$, i.e., we enumerate all subset of V having one vertex. Let w_1 be the total weight of the incident edges in v_i, and w_2 be the min 2–cut on $\bar{V}_1$. If $w_1+w_2 < W_\circ$, then put $W_\circ = w_1+w_2$, (it makes 3 components V_1, V_2, V_3 , and $|V_1| = 1$)

else do step 2.

Step 2:

Let w_3 be the weight of maximum min (S-T) cut such that $|S| = 2$, and $|T| = 2$, i.e. we enumerate all subset of V have two vertices. Pick S= $\{s_1, s_2\}$ and T= $\{t_1, t_2\}$ such that min(S-T) cut have a maximum source set V_s. Let w_4 be the weight of a min $(t_1 - t_2)$-cut in the set $\bar{V}_s = V_t$. If $w_3+w_4 < W_\circ$, then put $W_\circ = w_3+w_4$, (it makes 3 components V_1, V_2, V_3).

Output: $W_\circ$ the total weight of min 3-cut, and the components V_1, V_2, V_3.

Running time complexity:

Let G = (V, E) be an undirected graph. We know that the running time for min 2-cut is $\mathrm{T(n, m)}$ such that $|\mathrm{V}| = \mathrm{n}$, and $|\mathrm{E}| = \mathrm{m}$. (The best running time for min 2-cut is $\mathrm{O}(mn)$, it is due to Orlin (2013)).

In step 1, to pick v_i from V we have n ways, then the complexity to find $(w_1 + w_2)$ is $\mathrm{O(n\, T(n, m))}$, and in step 2, to pick S and T from V we have $\binom{n}{2}\binom{n-2}{2}$, i.e., $\left(\frac{n(n-1)}{2}\frac{(n-2)(n-3)}{2}\right)$ ways, then the complexity to find $(w_3 + w_4)$ is $\mathrm{O}(n^4\,\mathrm{T(n, m)})$.

Then the whole complexity of the algorithm is : $\mathrm{O}((n^4 + \mathrm{n})\mathrm{T(n, m)})$ =$\mathrm{O}(n^4\mathrm{T(n, m)})$.

5.1.4. The min k- cut Algorithm for fixed k (k > 3)

Input: graph G = (V, E), and k > 3. Let $W_\circ = \infty$.

Step 1:

For i ($1 \le i \le k-3$), let $V_1 = \{v_1, \dots, v_i\}$ such that $v_i \in V,$ i.e. we enumerate all subset S of V have i vertices, let w_1 be the weight of the min (S-T) cut partition G in to V_s and V_t ,let w_2 be the min(k-1) – cut on $\bar{V}_s$. If $w_1+w_2 < W_\circ$, then put $W_\circ = w_1+w_2$, (it makes k components $V_1, \dots, V_k$, and $|V_1| = i$)

else do step 2.

Step 2:

Let w_3 be the weight of maximal min (S-T) cut such that $|S| = k-2$, and $|T| = k-1$, i.e. we enumerate all subset of V have [(k-1) + (k-2) =(2k-3)] vertices pick S= $\{s_1, \ldots, s_{k-2}\}$ and T= $\{t_1, \ldots, t_{k-1}\}$ that the min(S-T) cut have maximum source set V_s.

Let w_4 be the weight of min $(k-1)$-cut in the set $\bar{V}_s = V_t$. If $w_3 + w_4 < W_\circ$, then put $W_\circ = w_3 + w_4$, (it makes k components $V_1, \ldots, V_k$).

Output: $W_\circ$ the total weight of min k-cut, and the components $V_1, \ldots, V_k$.

Running time complexity:

In step 1, to pick V_1 from V, we have $\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{k-3}$ ways, then the complexity to find $(w_1 + w_2)$ is $\mathrm{O}\left(\mathrm{n}^{k-3}\big(\mathrm{T(n,m)} + \mathrm{R(k-1)}\big)\right)$. In step 2, to pick S and T from V, we have $\binom{n}{k-2}\binom{n-(k-2)}{k-1} = \binom{n}{2k-3}$ ways, then the complexity to find $(w_3 + w_4)$ is $\mathrm{O(n^{2k-3}(T(n,m) + R(k-1)))}$.

Then the whole complexity of the algorithm is:

$\mathrm{R(k) = O((n^{2k-3} + n^{k-3})(T(n,m) + R(k-1))) \approx O(n^{2k-3}(T(n,m) + R(k-1)))}$.

Proposition (5-2):

The running time of the given algorithm is $\mathrm{O(n^{k^2})}$.

Proof:

The whole running time is: $\mathrm{R(k) = O(n^{2k-3}(T(n,m) + R(k-1)))}$.

$$\mathrm{R(k-1) = O(n^{2(k-1)-3}(T(n,m) + R(k-2)))},$$

$$\mathrm{R(k-i) = O(n^{2(k-i)-3}(T(n,m) + R(k-(i+1))))},$$

then $\mathrm{R(k) = O(n^{2k-3}T(n,m) + n^{2k-3}n^{2(k-1)-3}(T(n,m) + R(k-2)))}$.

$\mathrm{R(k) = O(n^{2k-3}T(n,m) + n^{2k-3}n^{2(k-1)-3}T(n,m) + n^{2k-3}n^{2(k-1)-3}n^{2(k-2)-3}(T(n,m) + R(k-3)))}$.

$\mathrm{R(k) = O((n^{2k-3} + n^{2k-3}n^{2(k-1)-3} + n^{2k-3}n^{2(k-1)-3}n^{2(k-2)-3})T(n,m) + n^{2k-3}n^{2(k-1)-3}n^{2(k-2)-3}R(k-3)))}$.

$\mathrm{R(k) = O(\big(n^{2k-3} + n^{2k-3}n^{2(k-1)-3} + \cdots + n^{2k-3} \ldots n^{2(k-(k-4))-3}\big)T(n,m) + R(3))}$,

since $\mathrm{R(3) = O\big(}\, n^4\mathrm{T(n,m)\big)}$, then

$$\mathrm{R(k)} = \mathrm{O}\left(\left(\mathrm{n}^{2k-3}\mathrm{n}^{2(k-1)-3}\mathrm{n}^{2(k-2)-3} \ldots \mathrm{n}^{5}\right)\mathrm{T(n,m)} + n^{4}\mathrm{T(n,m)}\right),$$

$$\mathrm{n}^{2k-3}\mathrm{n}^{2(k-1)-3}\mathrm{n}^{2(k-2)-3} \ldots \mathrm{n}^{5} = \mathrm{n}^{2k-3+2(k-1)-3+..+5} = \mathrm{n}^{(2k-3)(k-3)-2(1+2+\cdots+(k-4))}$$

Let $\beta = (2k-3)(k-3) - 2(1+2+\cdots+(k-4))$

$$\beta = (2k-3)(k-3) - 2\left(\frac{(k-4)(k-3)}{2})\right) = (k-3)(2k-3-k+4)$$

$$\beta = (k-3)(k+1) = k^2 - 2k - 3$$

$$\mathrm{R(k)} = \mathrm{O}\left(n^{k^2-2k-3}\mathrm{T(n,m)} + n^{4}\mathrm{T(n,m)}\right)$$

$$\mathrm{R(k)} = \mathrm{O}\left(\left(n^{k^2-2k-3} + n^{4}\right)\mathrm{T(n,m)}\right),$$

$$\mathrm{R(k)} = \mathrm{O}\left(\left(n^{k^2-2k+1}\right)\mathrm{T(n,m)}\right) = \mathrm{O}(\mathrm{n}^{k^2}) \qquad \square$$

5.1.5. The development of polynomial algorithms

Goldschmidt and Hochbaum (1994) give the first polynomial algorithm to solve MKCP for a fixed k by finding a minimum (S-T) cut with maximal source set, the core set S has at most k-2 vertices; and the terminal set T has at most k-1 vertices such that $V_s \subseteq V_1$ and $V_t \subseteq \bar{V}_1$, and by use the size of source and sink sets , denoted the min (S-T) cut by: min(1, k-1) cut .The complexity of this algorithm is O($n^{k^2}T(n,m)$). If we modified the bound of sizes of core set S and the terminal set T we will get better polynomial algorithm.

Kamidoi et al. find a minimum (S-T) cut with maximal source set, the core set S has at most k-2 vertices; and S is sub set of $V_D = \{V_1, \ldots, V_d\}, d = \lceil (k-\sqrt{k})/2 \rceil - 1$; and the terminal set T has at most k-2 vertices, such that $V_s \subseteq V_D$ and $V_t \subseteq \bar{V}_D$. By using the size of source and sink sets , we denote min (S-T) cut by: min (d, k-d) cut . The complexity of this algorithm is O ($n^{\frac{4k}{(1-1.71/\sqrt{k})}-34}T(n,m)$).

Xiao et al. (2011), find a minimum (S-T) cut, the core set S has at most 2 $\left\lfloor \frac{k}{2} \right\rfloor$ vertices; and S is sub set of $V_D = \{V_1, \ldots, V_d\}, d = \left\lfloor \frac{k}{2} \right\rfloor$;and the terminal set T has at most k-1 vertices, such that $V_s \subseteq V_D$ and $V_t \subseteq \bar{V}_D$, denoted the min (S-T) cut by: min($\left\lfloor\frac{k}{2}\right\rfloor$, k-$\left\lfloor\frac{k}{2}\right\rfloor$) cut . The complexity of this algorithm is O ($n^{4k-logk}T(n,m)$).

Table (5-1). Development of polynomial algorithms for min k- cut problem for a fixed k

	Goldschmidt and Hochbaum	**Kamidoi et al.**	**Xiao et al.**
Bounds on $\|S\|$ and $\|T\|$	k-2 and k-1	k-2 and k-2	$2\left\lfloor\frac{k}{2}\right\rfloor$ and k-1
The min (S-T) cut	(1, k-1) cut	(d, k-d) cut $d = \left\lceil(k-\sqrt{k})/2\right\rceil - 1$	($\left\lfloor\frac{k}{2}\right\rfloor$, k-$\left\lfloor\frac{k}{2}\right\rfloor$) cut
Complexity of the min k-cut problem	O ($n^{k^2}T(n,m)$)	O ($n^{\frac{4k}{(1-1.71/\sqrt{k})}-34}T(n,m)$).	O ($n^{4k-logk}T(n,m)$).

5.2. Minimum k-cut in forests

The connectivity number is the minimum number of vertices, whose removal makes graph disconnected or trivial. It is denoted $\lambda(G)$. The connectivity number of a disconnected graph is 0, and the connectivity number of a tree is 1.

5.2.1. The min k-cut for a tree

Lemma (5-1):

If we remove (n-1) edges from a tree, we will get n connected components, where n is the number of vertices for the considered tree.

Proof:

We prove the lemma by induction on n.

1- When n = 2 (if we remove one edge e = uv from a tree), we will get two connected components.
2- Suppose the lemma is true when n = k-1, then if we remove (k-2) edges from a tree we will get k-1 connected component, each one is a subtree, and if we remove one extra edge from one of them we will get k connected components.

Lemma (5.2):

Let T= (V, E) be a tree, with nonnegative edge weights. If we sort edges by weight function ($w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1})$) then a minimum k -cut is the set of edges C= $\{e_1, e_2, \dots, e_{k-1}\}$.

Proof:

Suppose $C^{\sim}$ is a minimum k- cut with $w(C^{\sim}) < w(C)$. Thus $C^{\sim}$ must have less than k-1 edges. It follows, by using Lemma 5.1, $C^{\sim}$ forms less than k components, and this is a contradiction.

Corollary (5-1)

The minimum k-cut problem in trees is solvable in polynomial time.

5.2.2. The min k- cut problem for forests

Proposition (5-3)

Let t be the number of the connected component on a forest.

For any $n \geq t$, if we remove (n-t) edges from a forest we will get n connected components.

Proof:

We will prove this proposition by induction on n.

Suppose that this is true when n=k-1, then if we remove (k-t-1) edges from a forest we will get k-1 = n connected components, each one is a tree, and if we remove one extra edge from one of them we will get k connected components, and the total edges we have removed is (k-t-1) +1 = (k-t) edges □

Lemma (5.3)

Let F= (V, E) be a forest, with nonnegative edge weights. If we sort edges by weight function ($w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{k-t}) \leq w(e_{|V|})$) then a minimum k -cut is the set of edges C= $\{e_1, e_2, \dots, e_{k-t}\}$, where t is the number of connected components of F.

Proof

Suppose $\tilde{C}$ is minimum k- cut with $w(\tilde{C}) < w(C)$. Thus $\tilde{C}$ must have less than k-t edges. It follows then, by lemma 5.2, $\tilde{C}$ forms less than k components, and this is a contradiction.

5.3. Special Instances of Multiway cut problem

We will present some basic definitions necessary for the proof of the main result of this section.

Coloration:

Let G = (V, E) be a simple graph, and a set of colors N=$\{1,2,\dots,r\}$. A map $\tilde{\chi}$: V→ N is a vertex coloration on G.

Partial Coloration:

Let G = (V, E) be a simple graph, a set of colors N=$\{1,2,\dots,r\}$, and let T⊆ V. A map χ: T→ N is a partial coloration.

A partial coloration defines a partition of T by $T_i = \{u \in T: \chi(u)=i\}$.

Color dependent weight function:

Color dependent weight function W: E→N∪ {0} determine for every edge uv and colors i , j the positive integer number w(uv; i, j) meaning the weight of edge uv, in which the color of u is i and the color of v is j.

We suppose the following in color dependent weight function:

- If u and v have the same color i then the color weight of edge uv is zero.
  i.e. w (uv; i,i) = 0
- In undirected graphs, the weight of edge uv is equal to the weight of edge vu.

i.e. w(uv;i,j) = w(vu;j,i)

In color independent weight function: for any edge uv ∈ E there is no effect of color on the edge weight, w(uv; i,j) = w(uv; a,b) such that $i \neq a$, $j \neq b$.

Color-Changing edge:

In the colouration $\tilde{\chi}$, an edge uv is color-changing if $\tilde{\chi}(u) \neq \tilde{\chi}(v)$.

Cut in graph coloring:

The set of color-changing edge form cut.

Color-Changing path:

In the colouration $\tilde{\chi}$, a path P is color-changing if $\tilde{\chi}(s(P)) \neq \tilde{\chi}(t(P))$

(s(P) is starting point of path and t(P) is the end) and P has no internal vertex.

Changing of coloration:

Let graph G= (V, E) and define color dependent weight function on G, the changing number of the coloration χ is the sum of weights of color-changing edge

change (G, $\tilde{\chi}$) = $\sum_{uv\in E} w(uv; i, j)$; $\tilde{\chi}(u) = i, \tilde{\chi}(v) = j$.

Multiway cut problem:

The multi way cut problem for the color weight independent function is the length of the pair (G, χ) and defined by:

L (G, χ) = minimum **change** (G,$\tilde{\chi}$).

An optimal coloration is a coloration $\tilde{\chi}$ such that **change** (G, $\tilde{\chi}$)= L (G, χ).

Since the multiway cut is NP-hard problem, we will present a lower bound for the optimal solution (Erdős et al. 1994).

5.3.1. Lower bound of multiway cut problem

Let G = (V, E) be simple graph, and χ: T→ N be a partial coloration, T⊆ V. let w be a color dependent weight function. For any collection $\mathcal{P}$ of color changing path define $n_i(uv; \mathcal{P}) = \# \{p \in \mathcal{P}: uv \in p; \chi(t(p) = i\}$.

Definition:

Path packing ($\mathcal{P}$) is a family of color-changing path such that: for all edge uv and for all color $i \neq j$, $n_i(uv; \mathcal{P}) + n_j(vu; \mathcal{P}) \leq w(uv; j,i)$

Ᵽ (G, χ) = maximum $|\mathcal{P}|$

Theorem 5.3:

For any graph, G and partial coloration χ, we have:

Ᵽ $(G, \chi) \leq L(G, \chi)$

Proof:

Let $\tilde{\chi}$: $V \rightarrow N$ be an optimal coloration and $\mathcal{P}$ be path packing.

Define function f: $\mathcal{P} \rightarrow E$ such that f(p) = e (e: is the last color-changing edge in path p in optimal coloration $\tilde{\chi}$)

For any color-changing edge e = uv:

$\# \{p \in \mathcal{P}: f(p) = e\} \leq n_i(uv; \mathcal{P}) + n_j(vu; \mathcal{P}) \leq w(uv; j,i)$.

$\# \{p \in \mathcal{P}: f(p) = e\} \leq w(uv; j,i)$.

$\sum_{e\in E} \# \{p \in \mathcal{P}: f(p) = e\} \leq \sum_{e\in E} w(uv; j, i)$.

$\sum_{e\in E} \# \{p \in \mathcal{P}: f(p) = e\} = |\mathcal{P}| \leq \sum_{e\in E} w(uv; j, i) = \mathbf{change}(G, \tilde{\chi})$.

$|\mathcal{P}| \leq \mathbf{change}(G, \tilde{\chi})$.

since $\tilde{\chi}$ is an optimal coloration then change $(G, \tilde{\chi}) = L(G, \chi)$

Max $|\mathcal{P}| \leq \mathbf{change}(G, \tilde{\chi})$.

Ᵽ $(G, \chi) \leq \mathbf{change}(G, \tilde{\chi})$. □

5.3.2. Multi way cuts in trees

Definition:

In put: Given a tree T = (V, E) with n vertices, nonnegative edge weights and a set of terminals $S = \{s_1, s_2, \ldots, s_k\}$, where $S \subseteq V$.

Question: Find a minimum total weight subset $E' \subseteq E$ such that its removal from the tree separates vertices in to k components each one have one terminals s_i.

Erdős et al. (1994) give a polynomial time algorithm to find multiway cut (in trees).

Coloration Algorithm:

Let T = (V, E) be a tree with n vertices, and a set of terminals S=L(T) (set of leaves on T), and χ: S →C be a partial coloration. let w be a color dependent weight function.
Define Penalty function: Pen:V→$(\mathcal{N} \cup \infty)^r$, if the color of v is i, $Pen_i(v)$ is the length of subtree partition by v from the root .

Step1:

Determine **pen**(v) for every vertex in tree (start from leaves to the root):

- For every v∈ L(T); let $\mathbf{pen_i}(v) = \begin{cases} 0 & \text{if } v \in S_i, \\ \infty & \text{else,} \end{cases}$
- For every v∉ L(T); and $\mathbf{pen}_i(v)$ is not computed yet and every **pen** (son (v)) is known; let $\mathbf{pen_i}(v) = \sum_{u \in son(v)} \min_{j=1,\dots,r}\{ w(uv; ji) + \mathbf{pen}_j(v)\}$

Step2:

Determine an optimal coloration $\tilde{\chi}$ of tree. (start from root to the **Father** (leaves))

- $\tilde{\chi}$(root) = i, such that i minimizes $\mathbf{pen}_i$(root).
- For vertex v that not determinant $\tilde{\chi}(v)$, let Father (v) = u, and $\tilde{\chi}(u)$ is known, then: $\tilde{\chi}(v)$ = i, such that i minimizes {w (vu; i, $\tilde{\chi}(u)$) + $\mathbf{pen}_i(v)$}.
- $\tilde{\chi}$(leave) = χ(leave).

Running time complexity:

The running time complexity is O (n×r^2× (max weight)), since in every step we calculate r^2 sums, take the minimum, and around 2n steps.

Example:

Find multiway cut on tree, the set of terminals are all leaves and the edges and their weights are as follow:
w(mx)=6, w(xy)=2, w(xz)=6, w(yL_8) =5, w(yL_7) =5, w(zL_6) =7, w(zL_5) =2, w(uL_4) =1, w(uL_3) =4, w(uL_2) =5, w(vL_1) =2, w(vu) =1, w(mv) =1. (Figure 6.1).

Step1:

Determine **pen**(v) for every vertex in tree; first compete penalty for all terminals:

, $\textbf{pen}(L_1) = (0, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$ $\textbf{pen}(L_2) = (\infty, 0, \infty, \infty, \infty, \infty, \infty, \infty)$,

, $\textbf{pen}(L_3) = (\infty, \infty, 0, \infty, \infty, \infty, \infty, \infty)$ $\textbf{pen}(L_4) = (\infty, \infty, \infty, 0, \infty, \infty, \infty, \infty)$,

, $\textbf{pen}(L_5) = (\infty, \infty, \infty, \infty, 0, \infty, \infty, \infty)$ $\textbf{pen}(L_6) = (\infty, \infty, \infty, \infty, \infty, 0, \infty, \infty)$,

, $\textbf{pen}(L_7) = (\infty, \infty, \infty, \infty, \infty, \infty, 0, \infty)$ $\textbf{pen}(L_8) = (\infty, \infty, \infty, \infty, \infty, \infty, \infty, 0)$,

Second compute penalty from father leaves to the root:

, $\textbf{pen}(u) = (10,5,6,9,10,10,10,10)$ $\textbf{pen}(v) = (6,7,8,8,8,8,8,8)$,

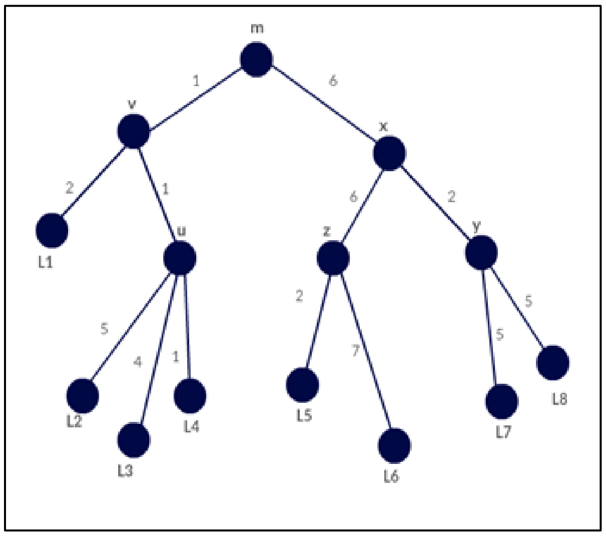
, $\textbf{pen}(z) = (9,9,9,9,7,2,9,9)$ $\textbf{pen}(y) = (10,10,10,10,10,10,5,5)$,

, $\textbf{pen}(x) = (15,15,15,15,15,9,13,13)$ $\textbf{pen}(m) = (21,22,22,22,22,16,20,20)$,

Step2:

Determine an optimal coloration $\tilde{\chi}$ of tree from root to leaves. (Figer 6.2).

$\tilde{\chi}(m) = 6$, $\tilde{\chi}(v) = 1$, $\tilde{\chi}(x) = 6$, $\tilde{\chi}(L_1) = 1$, $\tilde{\chi}(L_2) = 2$, $\tilde{\chi}(L_3) = 3$, $\tilde{\chi}(L_4) = 4$, $\tilde{\chi}(z) = 6$, $\tilde{\chi}(y) = 7$, $\tilde{\chi}(L_5) = 5$, $\tilde{\chi}(L_6) = 6$, $\tilde{\chi}(L_7) = 7$, $\tilde{\chi}(L_8) = 8$.



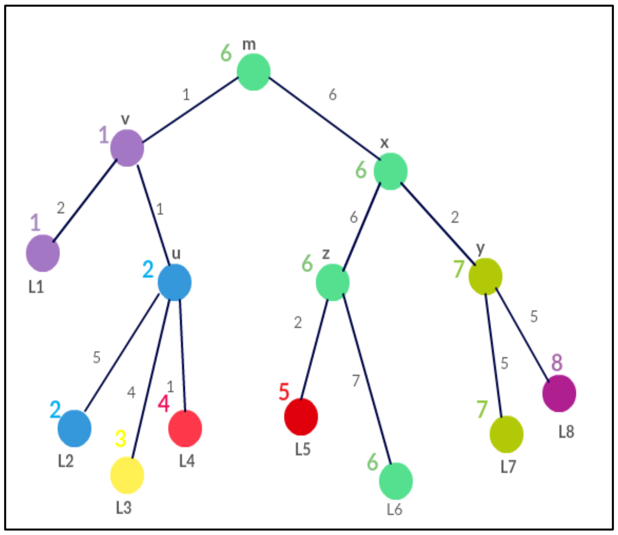


Figure (5-9): Example of a T tree with weight edge

Optimal coloration of T

5.3.3. Particular cases of multiway cut in trees

First case:

In put: Given a tree T = (V, E), color dependent weights function and a set of terminals S; where S $\subsetneq$V.

Question: Find multiway cuts.

In this case, we have a set of leaves U which do not belong to the set of terminals.

To solve this problem, we do the followings:

<u>Step1</u>: remove all leaves which are not terminals then find the optimal coloration for all vertices (V- U).

<u>Step2</u>: return the leaves U which we deleted and color this leaves with father color i.e. For every u $\in$U; $\tilde{\chi}$ (u) = $\tilde{\chi}$ (**Father**(u)).

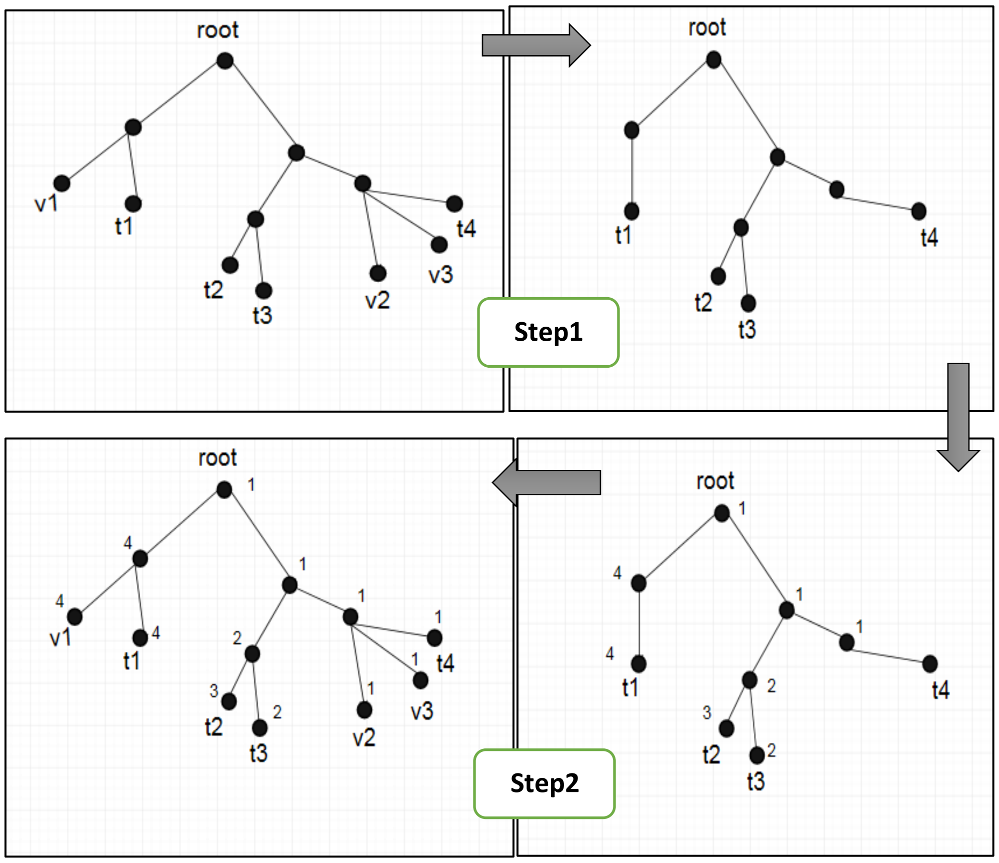


Figure (5-10) multiway cut on tree, first case

Second case:

In put: Given a tree T = (V, E), colour dependent weights function and a set of terminals S; such that S= L(T) ∪ $\{v_1, v_2, \dots, v_b\}$.(v_i is an internal vertex on the tree).

Question: Find multiway cut.

To solve this problem:

step1: partition the tree on the terminals $\{v_1, v_2, \dots, v_b\}$, which are not leaves then we will get b+1 components, and for each one, we will find the optimal coloration.

Step2: glue the components on the $\{v_1, v_2, \dots, v_b\}$.

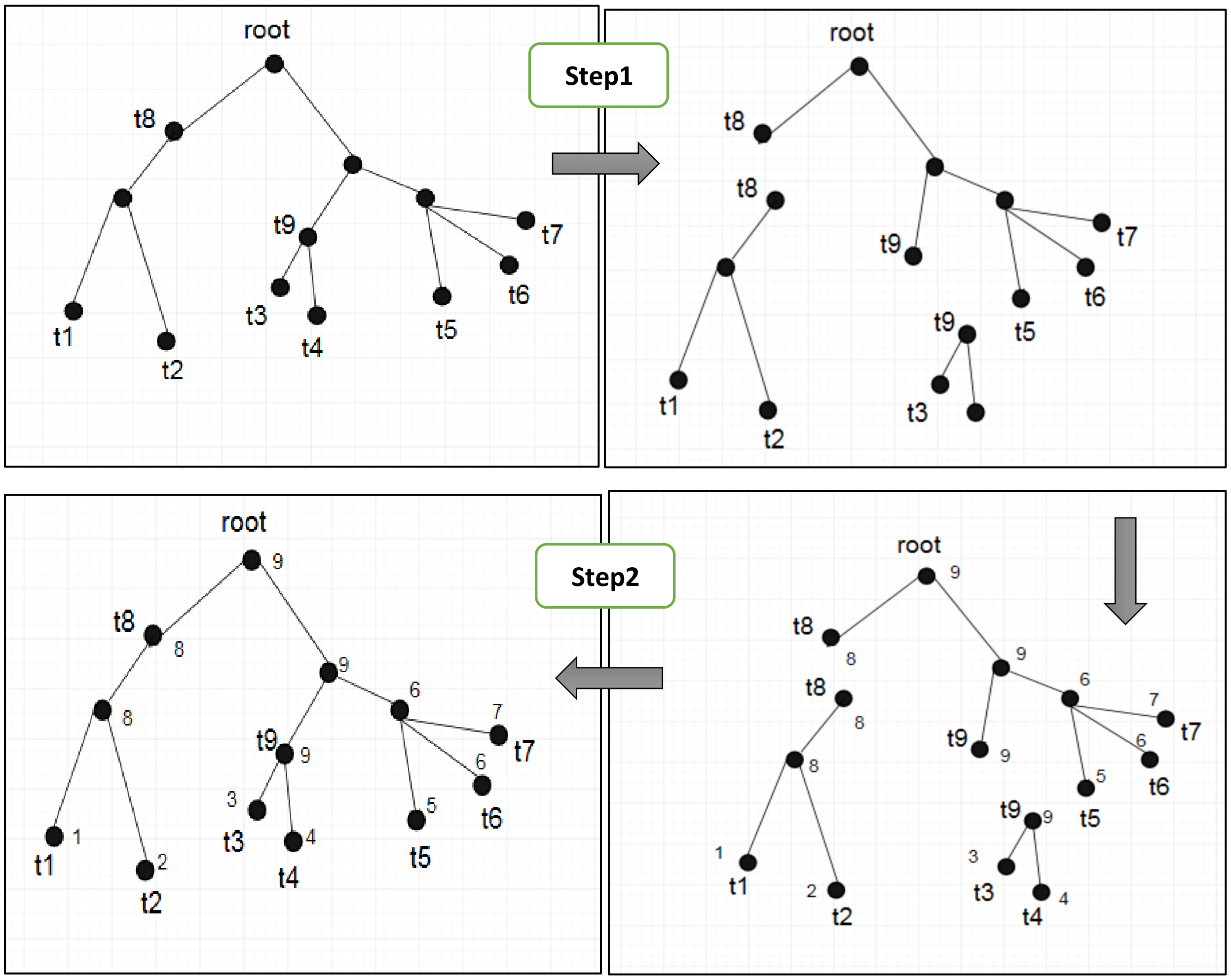


Figure (5-11) multiway cut on tree, second case

Third case:

In put: Given a tree T = (V, E), colour dependent weights function and a set of terminals $S \subseteq V$.

Question: Find multiway cut.

In this case, the set of terminals do not have any condition; it can contain some inner vertices and not necessary all leaves are terminals.

Step1: remove all leaves which are not terminals.

Step2: partition the tree on the terminals which are not leaves, and for each component find the optimal coloration.

Step3: joined the components on the terminals which are not leaves.

Step4: add the removed leaves and colour it with father’s colour.

Chapter 6: Approximation Algorithms

In this chapter, we describe two approximation algorithms for MKCP.

6.1. Approximation Algorithms for min k-cut

6.1.1. Greedy algorithm

The idea of this algorithm is to separate G into two components, and then separate one of the two components into two components (we get three components), and so on, until the process generates k components. In each step, we enumerate all possible cuts and select the one having the smallest weight. This algorithm is due to Saran and Vazirani (1995).

6.1.2. Algorithm EFFICIENT

Consider the cut C on a graph G, and let Comps(C) be the number of connected components in G induced by removing the cut C. If uv is an edge on G, we denote by $\mathrm{C_{uv}}$ be a min uv-cut separating the vertices u and v.

Phase 1. Find the cut $\mathrm{C_{uv}}$= min uv-cut for every edge uv in G.

Phase 2. Order the cuts on phase 1 as sequence $(C_1, C_2, \dots, C_m)$, by the weight function such that (w $(\mathrm{C_1}) \leq \mathrm{w}(\mathrm{C_2}) \leq \cdots \leq \mathrm{w}(\mathrm{C}_m)$).

Phase 3. Create a k-cut C by selecting cuts from sequence $(\mathrm{C_1}, \mathrm{C_2}, \dots, \mathrm{C_m})$ such that $\mathrm{C} = \bigcup_{\mathrm{i=1}}^{\mathrm{b}} \mathrm{C_i}$, (2 ≤b≤ m), and then isolate any cut $\mathrm{C_j}$ from $\{\mathrm{C_1}, \mathrm{C_2}, \dots, \mathrm{C_b}\}$ that satisfies $\mathrm{C_j} \subseteq \bigcup_{\mathrm{i=1}}^{\mathrm{j-1}} \mathrm{C_i}$.

Note:

- In phase 1, to find the cut $\mathrm{C_{uv}}$= min uv-cut, we use the max -flow min -cut theorem, and to find the set $\{\mathrm{C_1}, \mathrm{C_2}, \dots, \mathrm{C_m}\}$ we used m times this theorem.
- Let C be a k- cut producer from this algorithm generating more than k components,

we can remove some edges from C until creating a k cut.

- Now every edge uv in G is contained in the cut $\mathrm{C_{uv}}$, then $\bigcup_{i=1}^{m} \mathrm{C_i} = \mathrm{E}$, and it creates n components.

Lemma 6.1:

If $\{\mathrm{C_1}, \mathrm{C_2}, \ldots, \mathrm{C_d}\}$ is a k- cut producer from this algorithm, then for any s; $1 \leq s < d$, we have: Comps $(\bigcup_{i=1}^{s} C_i)$ < Comps $(\bigcup_{i=1}^{s+1} C_i)$

Proof:

There exists at least one edge uv in the set $\mathrm{C_{i+1}} - \bigcup_{i=1}^{s} \mathrm{C_i}$ since $\mathrm{C_{i+1}} \not\subseteq \bigcup_{i=1}^{s} \mathrm{C_i}$, and since the edge uv does not belong to the cut $\bigcup_{i=1}^{s} \mathrm{C_i}$ then it is a self-edge in the graph induced by the cut $\bigcup_{i=1}^{s} \mathrm{C_i}$, the end points u,v belong to the same connected component, since uv belongs to the cut $\mathrm{C_{i+1}}$ then it is a cross-edge in the graph induced by the cut $\bigcup_{i=1}^{s+1} C_i$, the end points u, v belong to the different component.

Then the graph induced by the cut $\bigcup_{i=1}^{s+1} C_i$ have more connected components. □

We can develop the algorithm EFFICIENT by using Gomory-Hu trees. For any graph, G = (V, E), there exists a set of $|V|$-1 cuts, that contain a min cut between every pair of vertices in G, then to find the cuts $\{\mathrm{C_1}, \mathrm{C_2}, \ldots, \mathrm{C_m}\}$ we need only (n-1) max flows.

6.1.3. Algorithm EFFICIENT with Gomory-Hu trees

Phase 1. Compute a Gomory-Hu tree in G.

Phase 2. Order the cuts on T as a sequence $(h_1, h_2, \ldots, h_{n-1})$, by the weight function (w $(\mathrm{h_1}) \leq \mathrm{w}(\mathrm{h_2}) \leq \cdots \leq \mathrm{w}(\mathrm{h}_{n-1})$).

Phase 3. Create a k-cut C by selecting cuts from the sequence $(h_1, h_2, \ldots, h_{n-1})$, such that b is the minimum number satisfying $\mathrm{Comps}(\bigcup_{i=1}^{b} h_i) \geq k$.

6.1.4. Approximation factor for the k-cut problem

Union property:

Let G be a graph and the sequence $\mathrm{C} = (\mathrm{C_1}, \mathrm{C_2}, \ldots, \mathrm{C_r})$ be all possible cuts in G, and for every i and j, $1 \leq i < j \leq r$, $\mathrm{w(C_i) \leq w(C_j)}$. And let $\mathrm{H = (h_1, h_2, \dots, h_l)}$ be any subsequence cuts of C. The cut H satisfies union property if for every k, $1 \leq k \leq$ $\mathrm{index(h_l)}$, if $\mathrm{h_1 \cup h_2 \cup \dots \cup h_q = C_1 \cup C_2 \cup \dots \cup C_k}$, such that $\mathrm{h_q}$ is the last cut in the sequence H with $\mathrm{index(h_q)} \leq k$.

Lemma 6.2:

If the cuts $\mathrm{h_1, h_2, \dots, h_l}$ are induced by the algorithm EFFICIENT, then these cuts satisfy union property.

Proof:

The proof is by contradiction, let $\mathrm{(C_1, C_2, \dots, C_r)}$ be all possible cuts in G, assume the cuts $\mathrm{(h_1, h_2, \dots, h_l)}$ be subsequence from$\mathrm{(C_1, C_2, \dots, C_r)}$, and it induced by Algorithm EFFICIENT, assume it do not satisfies union property, then there exist k , $1 \leq k \leq$ $\mathrm{index(h_l)}$, such that $\mathrm{h_1 \cup h_2 \cup \dots \cup h_q \neq C_1 \cup C_2 \cup \dots \cup C_k}$ (6-1)

and $\mathrm{h_q}$ is the last cut with $\mathrm{index(h_q)} \leq k$.

Let index $\mathrm{((h_{q+1}) = j}$, and since $\mathrm{w(h_q) < w(h_{q+1})}$, then $\mathrm{w(C_k) < w(C_j)}$, and from equivalents (6-1), $\mathrm{h_q \neq C_k}$, and

$\mathrm{h_1 \cup h_2 \cup \dots \cup h_q = C_1 \cup C_2 \cup \dots \cup C_{k-1}}$. (6-2)

Suppose the edge e belong to the cut $\mathrm{C_k - [C_1 \cup C_2 \cup \dots \cup C_{k-1}]}$, then $e \notin \mathrm{C_1 \cup C_2 \cup \dots \cup C_{k-1}]}$, and by (6-2), $\mathrm{e \notin h_1 \cup h_2 \cup \dots \cup h_q}$.

Let $\mathrm{C_e}$ be a min cut separate the endpoint of e, and since $e \in \mathrm{C_k}$, then $\mathrm{w(C_e) \leq w(C_k)}$, and $\mathrm{w(C_k) < w(h_{q+1})}$, then $\mathrm{w(C_e) < w(h_{q+1})}$, then $\mathrm{e \in h_1 \cup h_2 \cup \dots \cup h_q}$, it is contradiction. □

Theorem 6.1:

Algorithm EFFICIENT finds a k-cut having weight within a factor of $\left(2 - \frac{2}{k}\right)$of the optimal. (Saran and Vazirani (1995).

Proof:

Consider an undirected graph G = (V, E), and let C be an optimal min k-cut in G that partitions G into k components $\{V_1, V_2, \dots, V_k\}, \mathrm{C_i} \subseteq \mathrm{C}$ be the cut separating V_i from $\bar{V}_i$ ($1 \leq i \leq$ k). Hence $C = \bigcup_{i=1}^{k} C_i$, and sort the cut C_i by weight function, $\mathrm{w(C_1)} \leq \mathrm{w(C_2)} \leq \cdots \leq \mathrm{w}(C_k)$, and let H be the k -cut induced by Algorithm EFFICIENT, and $\mathrm{H} = \bigcup_{\mathrm{i=1}}^{\mathrm{l}} \mathrm{h_i}$ ($\mathrm{l} \leq \mathrm{k} - 1$).

By proposition (5-1): $\mathrm{w(C)} = \mathrm{w}(\bigcup_{\mathrm{i=1}}^{\mathrm{k}} \mathrm{C_i}) = \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{k}} \mathrm{w(C_i)}$, then:

$2\mathrm{w(C)} = \sum_{\mathrm{i=1}}^{\mathrm{k}} \mathrm{w(C_i)}$, (6-3)

since $\mathrm{w(C_i)} \leq \mathrm{w(C_k)}$ for every i then: $2\mathrm{w(C)} = \mathrm{w(C_1)} + \mathrm{w(C_2)} + .. + \mathrm{w(C_k)} \leq \mathrm{k\,w(C_k)}$,

then: $\frac{2}{k}\mathrm{w(C)} \leq \mathrm{w(C_k)}$. (6-4)

Since $l \leq k - 1$, and by lemma (6-2) the cuts $(h_1, .., h_l)$ satisfies union property,

$\sum_{i=1}^{l} w(h_i) \leq \sum_{i=1}^{k-1} w(C_i)$, then

$w(\mathrm{H}) = \mathrm{w}(\bigcup_{\mathrm{i=1}}^{\mathrm{l}} \mathrm{h_i}) \leq \sum_{i=1}^{l} w(h_i) \leq \sum_{i=1}^{k-1} w(C_i)$,

$w(\mathrm{H}) \leq \sum_{i=1}^{k-1} w(C_i) = \sum_{i=1}^{k-1} w(C_i) + w(C_k) - w(C_k) = \sum_{i=1}^{k} w(C_i) - w(C_k)$,

$w(\mathrm{H}) \leq \sum_{i=1}^{k} w(C_i) - w(C_k)$, from (6-3) and (6-4)

$w(\mathrm{H}) \leq 2w(C) - \frac{2}{k}w(C)$,

$w(\mathrm{H}) \leq 2\left(1 - \frac{1}{k}\right)w(C)$ □

Running time complexity:

Let G = (V, E) be an undirected graph, If we assume that the running time for max flow is $\mathrm{T(n, m)}$ such that $|\mathrm{V}| = \mathrm{n}$, and $|\mathrm{E}| = \mathrm{m}$. (The best running time for min cut is $\mathrm{O}(mn)$, it is due to Orlin. (2013).

The running time of the algorithm EFFICIENT with Gomory-Hu tree is O((n-1) $\mathrm{T(n, m)}$) and if $\mathrm{T(n, m)} = \mathrm{O}(mn)$ then our algorithm has running time $\mathrm{O}(mn^2)$.

6.1.5. Algorithm SPLIT

Phase 1: Pick the smallest cut split the graph in to two components, and remove the edges of this cut from graph.

Phase 2: In the present graph, again pick the smallest cut split one of the component in to two components, and remove the edges of this cut from graph.

Repeat this process until the present graph has k connected components**.**

The different between SPLIT and EFFICIEN algorithms that SPLIT algorithm picks smallest cut in the present graph and EFFICIEN algorithm picks smallest cut in the initial graph.

Running time complexity:

If the graph G has order n, then to generate a new component, we need (n-1) max flows, and to create k components we need (k-1) split, then the running time of SPLIT algorithm is O (k n (T(n,,m)).

Algorithm SPLIT finds a k-cut having weight within a factor of $\left(2-\frac{2}{k}\right)$ of the optimal. (Saran and Vazirani (1995).

6.1.6. Lower bound

Theorem (6-2)

The min k cut found by EFFICIENT and SPLIT algorithms lies in the range

$$(1-\varepsilon)\left(2-\frac{2}{\mathrm{k}}\right)\mathrm{optC} \leq \mathrm{optB} \leq \left(2-\frac{2}{\mathrm{k}}\right)\mathrm{optC}$$

Such that $0 \leq \varepsilon \leq 1$, and C is the optimal k-cut and B is the k-cut found by EFFICIEN or SPLIT algorithms

Proof:

Consider the graph G = (V, E), of order =(2k-1), let $V = \{x_1, x_2, \dots, x_k, y_{k+1}, \dots, y_{2k-1}\}$, and let the weight function as figure (6-1).

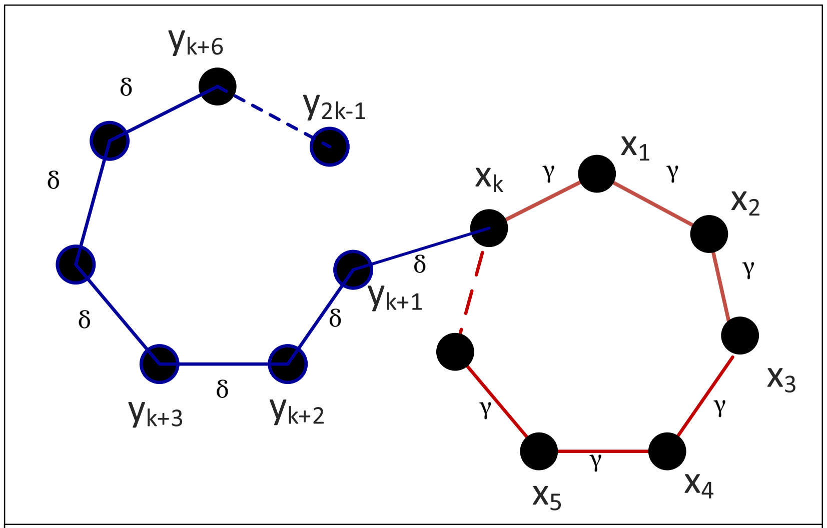


Figure (6-1) : A tight example of EFFICIEN and SPLIT algorithms

Note that $\delta = 2\gamma(1-\varepsilon)$, and $\varepsilon > 0$.

The min k-cut, C, is the cut of all edge of weight γ, then $w(C) = k\gamma$.

The min k-cut, B, is the cut of all edge of weight δ, then $w(B) = \delta(k-1)$.

$w(B) = \delta(k-1) = 2\gamma(1-\varepsilon)(k-1)$,

$w(B) = \gamma k(1-\varepsilon)\frac{2}{k}(k-1) = w(C)(1-\varepsilon)\left(2-\frac{2}{k}\right)$. □

6.2. Approximation Algorithms for the variants of the min k-cut

6.2.1. Multiway Cut Problem

Dahlhaus and Johnson (1994) give a $2\left(1-\frac{1}{k}\right)$ approximation algorithm for the multiway cut problem; the best approximation algorithm is $O(\log|k|)$given by Garg et al. (1996).

6.2.2. The Minimum Steiner k-Cut Problem

Chekuri et al. (2004) give two approximation algorithms for the problem: a greedy one with a $2\left(1-\frac{1}{k}\right)$-approximation based on Gomory-Hu trees, and a $2\left(1-\frac{1}{|T|}\right)$-approximation based on LP rounding.

6.2.3. Capacitated Max k-Cut Problem

- For un equal capacities: Feige et al. (2001) give an approximation algorithm with a lower bound of $\frac{1}{2} + \varepsilon$ when k = 2, and ε is a universal constant.
- For equal capacities: Andrson (1999) give an algorithm that obtains a $1 - \frac{1}{k} + \Omega\left(\frac{1}{k^3}\right)$ performance guarantee.
- For general (equal or un equal capacities): Ageev et al. (2001) give a $\frac{1}{2}$ approximation algorithm.

6.2.4. Multiway Un Cut Problem

Langberg et al. (1970) give a 0.8535-approximation algorithm for multiway uncut use linear programming relaxation. (**linear programming relaxation** of a 0-1 integer program is the problem that arises by replacing the constraint that each variable must be 0 or 1 by a weaker constraint, that each variable belong to the interval [0,1]).

6.2.5. k-route cut problem

Chuzhoy et al. (2011) give a $O(klogr)^{\frac{3}{2}}$-approximation algorithm for k-route cut.

6.2.6. Directed multiway cut problem

The best approximation for this problem is $\mathrm{O}(\min(\sqrt{\mathrm{n}}, \mathrm{opt}))$ by Gupta (2003).

Kortsarts et al. (2005) gives an $O(n^{\frac{2}{3}})$- approximation algorithm with running time O (nm^2).

Chapter 7 : Conclusion

Dahlhaus et al. (1992) proved that the Minimum k-cut optimization problem is NP-hard. Saran and Vazirani (1995) find a $2\left(1-\frac{1}{k}\right)-$approximation algorithm, with running time $\mathrm{O}(\mathrm{n}-1)$ max flow, based on Gomory- Hu trees.

For a fixed k, Xiao et al. (2011) give a polynomial algorithm with a running time $O(n^{4k})$ max flow.

Future investigations can be studying the minimum k-cut problem in graphs other than trees or forests, as outerplanar graphs or series parallel graphs. If these studies keep the NP-hardness of the problem, approximations algorithms will be a motivated approach for this problem.

References

[1] A. Ageev, R. Hassin, and M. Sviridenko (2001), *A 0.5-approximation algorithm for max dicut with given sizes of parts*, SIAM Journal on Discrete Mathematics 14 (2), pp. 246-255.

[2] W. Ben Ameur, A. R. Mahjoub and J. Neto (2010); *The Maximum cut problem*, in: *Paradigms of Combinatorial Optimization Problems and New Approaches*, V. T. Paschos (editor), 2nd edition, Wiley.

[3] G. Andersson (1999), *An approximation algorithm for max p-section*, Lecture Notes in Computer Science (Proceedings of STACS'99) 1563, pp. 237–247.

[4] F. Barahona (1981), *Balancing signed of fixed genus in polynomial time,* Technical Report, Departement of Mathematics, University of Chile.

[5] F. Barahona (1983), *The max-cut problem on graphs not contractible to K_5*, Operations Research Letters 2, pp. 107-111.

[6] F. Barahona and A. R. Mahjoub (1986), *On the cute polytope,* Mathematical Programming 36, pp. 157-173.

[7] F. Barahona (2000), *On the k-cut problem,* Operations Research Letters 26, pp. 99-105.

[8] Y.Y. Boykov, M.P.Jolly (2011), *Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images*, Proceedings. Eighth IEEE International Conference on.

[9] J. A. Bondy and U. S. R. Murty (2008); *Graph Theory;* 1st edition, Springer Verlag.

[10] M. Bruglieri, F. Maffioli and M. Ehrgott (2004), *Cardinality constrained minimum cut problems: Complexity and Algorithms*, Discrete Applied Mathematics 137, pp. 311-341.

[11] C. Chekuri, S. Guha, and J. (Seffi) Naor (2004), *The Steiner k-Cut Problem,* SIAM J. Discrete Math 20, pp. 261–271.

[12] S. R. Choudhury (2008), *Approximation algorithms for a graph-cut problem with applications to a clustering problem in bioinformatics*, MSc thesis, Department of Mathematics and Computer Science, University of Lethbridge.

[13] V. Chvátal (1973), *Tough Graphs and Hamiltonian Circuits*, Discrete Mathematics 5, pp. 215-228.

[14] S. A. Cook (1971), *The complexity of theorem-proving procedures*, Proceedings of the third annual ACM symposium on Theory of computing, pp. 151-158.

[15] W. H. Cunningham (1985), *Optimal Attack and Reinforcement of a Network*, J. Assoc. Comput. Mach. 32, pp. 549- 561.

[16] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis (1994), *The complexity of multiterminal cuts*, SIAM Journal on Computing, pp. 864-894.

[17] P. Erdős and L. A. Székely (1994), *On weighted multiway cuts in trees*, Mathematical Programming 65, pp. 93-105.

[18] U. Feige and M. Langberg (2001), *Approximation algorithms for maximization problems arising in graph partitioning,* J. Algorithms 41, pp. 174–211.

[19] M. R. Garey and D. S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness,* 1st edition, W. H. Freeman and Co, New York.

[20] N. Garg, V. Vazirani, and M. Yannakakis (1997), *Primal-dual approximation algorithms for integral flow and multi-cut in trees*, Algorithmica 18, pp. 3-20.

[21] D. R. Gaur, R. Krishnamurti, and R. Kohli, (2008), *The capacitated max k-cut problem,* Mathematical Programming 115, pp. 65-72.

[22] M. X. Goemans and D. P. Williamson (1995), *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming,* J. Assoc. Comput. Mach. 42, pp. 1115-1145.

[23] M. X. Goemans and D. P. Williamson, (2004), *Approximation algorithms for max 3-cut and other problems via complex semidefinite programming*, J. Comput. Sys. Sci. 68, pp. 442-470.

[24] O. Goldschmidt and D. S. Hochbaum (1994), *A polynomial algorithm for the k-cut problem for fixed k,* Mathematics of Operation Research 19, pp. 24-37.

[25] R. E. Gomory and T. C. Hu (1991), *Multi-terminal network flows*, J. Soc. Indust. Apple. Math. 9, 551-570.

[26] A. Gupta (2003), *Improved approximation algorithm for directed multi cut*, Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 12-14.

[27] F. Hadlock (1975), *Finding a maximum cut of a planar graph in polynomial time*, SIAM Journal on Computing 4, pp. 221-225.

[28] V. Kann, S. Khanna, J. Lagergren, and Panconesi (1997), *On the hardness of approximating max-k-cut and its dual*, Chicago Journal of Theoretical Computer Science.

[29] R. M. Karp (1972), *Reducibility among combinatorial problems*, in: R. E. Miller and J. W. Thatcher (editors), Plenum, New York, pp. 85-103.

[30] V. King, S. Rao, and R. E. Tarjan (1994), *A faster deterministic maximum flow algorithm*, In selected papers from the Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, Orlando, FL, USA. Academic Press, Inc, pp. 447- 474,

[31] Y. Kortsarts, G. Kortsarz and Z. Nutov (2005), *Greedy Approximation Algorithms for Directed Multicuts*, Networks and International Journal 45, pp. 214-217.

[32] M. Langberg, Y. Rabani and C. Chaitanya (1970), *Approximation Algorithms for Graph Homomorphism Problems*, Lecture Notes in Computer Science 4110, pp. 176-187, in: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques.

[33] L. A. Levin (1973), *Universal search problems*, Problemy Perdaci Informacii 9, pp. 265-266.

[34] S. T. McCormick, M. R. Rao, and G. Rinaldi (2003), *Easy and difficult objective functions for max cut*, Mathematical Programming 94, pp. 459-466.

[35] J. B. Orlin (2013), *Max flows in O(nm) time, or better*, TOC '13 Proceedings of the forty-fifth annual ACM symposium on Theory of computing, pp 765-774.

[36] H. Saran and V. V. Vazirani (1995), *Finding k cuts within twice the optimal*, SIAM Journal on Computing 24, pp. 101-108.

[37] M. Xiao, L. Cai and A.C-C.Yao (2011), *Tight Approximation Ratio of a General Greedy Splitting Algorithm for the Minimum k-Way Cut Problem*, Algorithmica 59, pp. 510-520.

[38] Wikipedia (2011), https://commons.wikimedia.org/wiki/File:Spin_glass_by_Zureks.svg.

[39] Wikipedia (2007), https://en.wikipedia.org/wiki/NP_(complexity)#/media/File:P_np_np-complete_np-hard.svg.

Terminology Index

Strong polynomial time algorithm	12
Sub graph	4
Terminal set	36
Tree	5
Value of a cut	5
Value of a flow	7
Weighted graph	3