Geometrical k-cut Problem and An Optimal Solution for Hypercubes

Sang-Young Cho
Hankuk University of Foreign Studies
Computer Science and Engineering Department
Wangsan Mohyeon Yongin Kyeonggi
Korea

Abstract: We introduce a generalization of the (s,t)-cut problem, called the Geometrical k-cut problem, with the concept of geometrical partitioning. A topology graph is employed to represent the geometrical structure of the partitioned nodes of a given k-terminal graph. This problem is NP-hard in general. We propose an optimal algorithm to solve the problem for hypercube topology graphs in polynomial time. The time complexity of the algorithm is $O(qn^3)$, where q is the dimension of a hypercube graph and n is the number of nodes in a k-terminal graph, with the Goldberg-Tarjan's network flow algorithm.

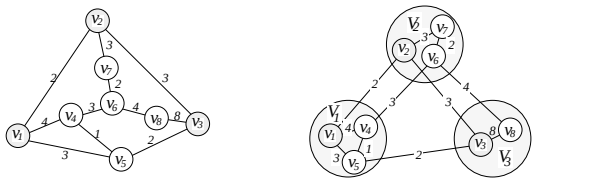
Key–Words: complexity, k-cut, hypercube, max-flow, min-cut, multi-terminal graph, partitioning, topology

1 Introduction

Given an undirected graph $G = (V, E)$ with positive edge weights and two terminals $s, t \in V$, an (s,t)-cut is a set of edges which, when deleted, partitions V into two subsets V_1 and V_2 such that $s \in V_1$ and $t \in V_2$. An (s,t)-cut is a *minimum* (s,t)-cut if the sum of the edge weights in the (s,t)-cut is minimum. For any $s, t \in V$, the well-known max-flow min-cut theorem [2] allows us to find a minimum (s,t)-cut in polynomial time by applying a maximum flow algorithm [1], [3].

An extension of the minimum (s,t)-cut problem is called the *specified k-cut problem*, which has numerous applications, particularly in clustering-related setups such as task assignment [8] and VLSI cell placement [9]. The problem considers an undirected k-terminal graph $G = (V, E)$ with positive edge weights and a set of prescribed $k(\leq n)$ nodes called *terminal* nodes, where $V = \{v_i | 1 \leq i \leq n\}$ and each edge of E is an unordered pair of nodes (v_i, v_j) in V. A k-cut is a set of edges which, when deleted, partitions V into k subsets $\{V_i | 1 \leq i \leq k\}$. The aim is to find a minimum weight k-cut so that each subset contains exactly one node of the given k. The minimum (s,t)-cut problem is therefore an instance of the specified k-cut problem where $k = 2$. The specified k-cut problem is NP-hard even for $k = 3$ [4].

In this paper, we introduce a generalization of the specified k-cut problem with the concept of *Geometrical Partitioning*. In the specified k-cut problem, the contribution of each edge in a k-cut to the cut weight is assumed to be just the edge weight. In our new problem we assume that each partitioned subset V_i resides in a distinct area and there exists a geometrical distance between two distinct areas. If an edge (v_i, v_j) is included in a k-cut as a result of a partitioning, then the contribution of (v_i, v_j) to the partition cost (cut weight) is not merely the weight of (v_i, v_j). Rather, its contribution is compensated by the geometrical distance between the two partitioned subsets which contain v_i and v_j respectively. In the remaining of this paper, the term *Geometrical Partitioning* refers to a partitioning of a k-terminal graph with a specified geometrical structure of k subsets so that the partitioning cost is compensated by the geometrical distances between the partitioned subsets.



(a) A three-terminal graph. (b) A non-optimal solution for the specified k-cut problem.

Figure 1: An example of a three-terminal graph and a possible three-cut.

Fig. 1(a) shows an example of a three-terminal graph where v_1, v_2, and v_3 are terminal nodes. Fig. 1(b) shows a possible 3-cut of the graph, namely $\{(v_1, v_2), (v_4, v_6), (v_2, v_3), (v_6, v_8), (v_3, v_5)\}$. The nodes of the graph are partitioned into three subsets:

V_1, V_2, and V_3, and each subset has exactly one terminal node. In the specified k-cut problem, the weight of a k-cut is the summation of the weights of the edges in the k-cut. Therefore the weight of the 3-cut shown in Fig. 1(b) is 14. To explain geometrical partitioning, let us assume that the distance between V_1 and V_2 is 1, that between V_2 and V_3 is 1, and that between V_3 and V_1 is 2. Accordingly, for example, the contribution of (v_3, v_5) is defined to be the weight of the edge times the distance between V_1 and V_3, i.e., $2 \times 2 = 4$, assuming that the cost of an edge in the cut is proportional to its distance. Therefore, in our new problem, the weight of the k-cut is 16, i.e., $2\times1 + 3\times1 + 3\times1 + 4\times1 + 2\times2$. The new partitioning problem, called the *Geometrical k-cut Problem* (GKP), is NP-hard because the problem is equivalent to the specified k-cut problem [4] if the distance between any two subsets is one.

The geometrical distances between partitioned subsets can be represented by a connected graph, called *Topology Graph*. For example, a three-node linear array can represent the distances between the nodes of the graph shown in Fig. 1(b) if they are arranged in the order of V_1, V_2, and V_3. We propose an algorithm to solve the problem when the topology graph is a hypercube graph in polynomial time.

This paper is organized as follows. In Section 2, we give a formal definition of GKP. Section 3 suggests our approach for the problem in case of hypercube topology graphs. The paper is concluded in Section 4.

2 Problem Description

Consider an undirected graph G with k terminal nodes and positive edge weights, where $G = (V, E)$, V is a set of n nodes, $\{v_i | 1 \le i \le n\}$, and E is a set of edges where each edge (v_i, v_j) is an unordered pair of nodes of V. The weight of an edge (v_i, v_j) is denoted as $w_{i,j} = w_{j,i}$. Without loss of generality, let the set of k terminal nodes be $K = \{v_i | 1 \le i \le k \le n\}$.

The geometrical distances between k partitioned subsets can be modeled by a *topology graph* $G_T = (P, L)$, where P is a set of k points, $\{p_i | 1 \le i \le k\}$, and L is a set of lines where each line corresponds to an unordered pair of points in P. We use the terms *point* and *line* instead of node and edge to distinguish topology graphs from terminal graphs, where a point corresponds to a location at which one of the partitioned subsets should be located. If two points are geometrically adjacent to each other, a line is created to connect them. Non-adjacent points are connected via other points along a path between them. We define the *distance* $d_{i,j}$ between any two points p_i and p_j to be the minimum number of lines connecting them. We assume that the distance between any two adjacent points is one. Obviously $d_{i,i} = 0$.

A partition of V into k subsets can be represented by a mapping function. A mapping M of the nodes V to the points P of G_T is a function $M : v_i \rightarrow p_{M(i)}$, i.e., $M(i)$ is the index of the point onto which v_i is mapped. We define $M^{-1}(i) = \{v_j | M(j) = i\}$ for $1 \le i \le k$.

We assume that each terminal node has a fixed mapping and, without loss of generality, $M(i) = i$ for $1 \le i \le k$, i.e., v_i should be mapped into p_i. A mapping M_f is said to be *feasible* if each $M^{-1}(i)$ contains v_i of K.

We define the cost of an edge (v_i, v_j) under a mapping M to be $w_{i,j} \cdot d_{M(i),M(j)}$, i.e., the cost of an edge is proportional to the distance between its two mapped points. The cost of a mapping $Cost(M)$ is defined to be the sum of the cost contributions made by all the edges of G:

$$Cost(M) = \sum_{i<j} w_{i,j} \cdot d_{M(i),M(j)} \cdot$$

Note that the cost of an edge (v_i, v_j) is zero if v_i and v_j are mapped into the same point.

The *Geometrical k-Cut Problem (GKP)* can now be formulated as follows: Given a k-terminal graph $G = (V, E)$ and a topology graph $G_T = (P, L)$ as described earlier, find a feasible mapping M_f of the nodes of V onto the points of P such that $Cost(M_f)$ is minimized.

In general, GKP is NP-hard because the problem can be reduced to the specified k-cut problem (which is NP-hard even for $k = 3$ [4]) if the topology graph is assumed to be a fully-connected graph; in this case the distance between any two points is one and the cost function can be represented as follows:

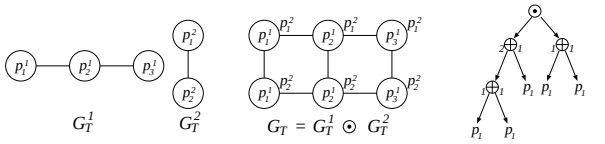
$$Cost(M) = \sum_{i<j, M(i) \neq M(j)} w_{i,j} \ ,$$

which is the cut weight in the specified k-cut problem.

3 Hypercubes and Solution

In this section, the product graph operator is introduced to recursively construct a type of hypercube graphs starting from trivial one-point graphs. Using these graph operators, a graph expression can express a hypercube topology in a hierarchical form. Our solution to GKP is presented in a recursive form based on the hierarchical representation of a hypercube graph.

Definition 1 (product operator) *Given two topology graphs $G_T^1 = (P^1, L^1)$ and $G_T^2 = (P^2, L^2)$, the* product operator $\odot$ *produces a product of G_T^1 and G_T^2, denoted as G_T, where $G_T = (P, L) = G_T^1 \odot G_T^2$ such that $P = P^1 \times P^2 = \{(p_x^2, p_y^1) | \ p_x^2 \in P^2, \ p_y^1 \in P^1\}$ and any two points $p_i = (p_w^2, p_x^1)$ and $p_j = (p_y^2, p_z^1)$ in P are adjacent in G_T if (1) $p_w^2 = p_y^2$ and p_x^1 is adjacent to p_z^1 in G_T^1, or (2) $p_x^1 = p_z^1$ and p_w^2 is adjacent to p_y^2 in G_T^2.*



(a) Two topology graphs. (b) The result of the product operator. (c) The hierarchy tree.

Figure 2: An example of the product operator.

Fig. 2 shows an example use of the product operator. The generated topology graph G_T contains $|P^1| \cdot |P^2|$ points and $|P^1| \cdot |L^2| + |P^2| \cdot |L^1|$ lines. To address the points of the generated G_T, we associate each product operator with an index function $A_\odot$. Let a point of G_T be generated with p_i^2 and p_j^1. The index pair (i, j) denotes the generated point. Let all the points of G_T be lexicographically sorted by index pairs. Assuming the point with index pair (i, j) is the w-th element in the sorted list, the point is addressed as p_w by the index function $A_\odot$, i.e., $A_\odot(p_i^2, p_j^1) = p_w$. In Fig. 2 (b), $A_\odot(p_1^2, p_1^1) = p_1$, $A_\odot(p_1^2, p_2^1) = p_2$, $A_\odot(p_1^2, p_3^1) = p_3$, $A_\odot(p_2^2, p_1^1) = p_4$, $A_\odot(p_2^2, p_2^1) = p_5$, and $A_\odot(p_2^2, p_3^1) = p_6$. The hierarchy tree shown in Fig. 2 (c) is built for the G_T shown in Fig. 2 (b); $\oplus$ means connecting two graphs with a line.

Definition 2 (hypercube topology graph) *A binary n-cube or an n-dimensional hypercube, Q_n, is defined recursively as follows:*

1. *Q_0 is a trivial graph with one point, and*
2. *$Q_q = K_2 \odot Q_{q-1}$, where $K_2 = Q_0 \oplus Q_0$ [5].*

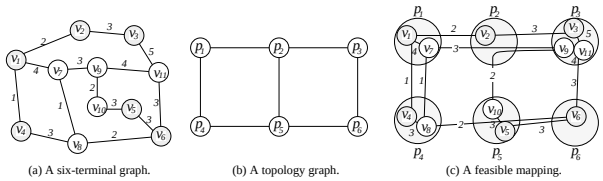
Consider a GKP instance with a k-terminal graph $G = (V, E)$ and a topology graph $G_T = (P, L)$. If G_T is a hypercube graph, the GKP instance is called a Geometrically Cuttable k-cut Problem (GCKP) and denoted by the pair $[G, Q_q]$. We assume that Q_q is associated with a corresponding hierarchy tree H and each operator in the tree is associated with an index function. Note that we assumed each terminal node v_i, $1 \leq i \leq k$, is, without loss of generality, mapped into the point p_i. The term *specified* is used to reflect this static mapping constraint for the terminal nodes.

Given a GCKP $[G, Q_q]$, we can transform the original problem into several subproblems such that each subproblem is an (s, t)-cut problem, which can be solved optimally using an existing network-flow algorithm. By combining the solutions for the subproblems we can obtain the solution for the original problem whose cost is the sum of the costs of the subproblems. Our strategy for solving GCKP follows the divide-and-conquer methodology which allows solutions of GCKP to be found efficiently.

Given a GCKP $[G, Q_q]$ with a corresponding hierarchy tree H, let us traverse H from the top level to the bottom level. Whenever a product operator is visited, two independent subproblems can be created so that the optimal solutions for the subproblems can determine an optimal solution for the original problem. The two subproblems are created with the following rule, Rule 3:

Rule 3 *Assume that $G_T = G_T^1 \odot G_T^2$, where $G_T^1 = (P^1, L^1)$, $G_T^2 = (P^2, L^2)$, and $G_T = (P, L)$, and an index function $A_\odot$ is associated with the product operator.*

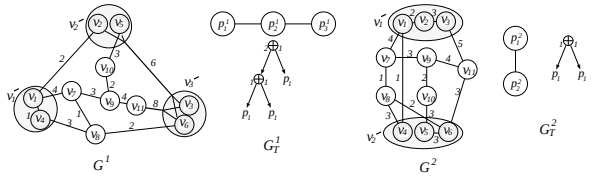
1. *For each terminal node v_w of G, declare it to be specified into a point p_j^1, resp. p_i^2, if $p_x = A_\odot(p_i^2, p_j^1)$ and v_w is specified into p_x.*
2. *For all points of subtopology graphs, combine the nodes specified into p_j^1, resp. p_i^2, into a single super node and call the node v'_j, resp. v'_i. Declare that v'_j, resp. v'_i, is specified into p_j^1, resp. p_i^2. A super node maintains a list of its component nodes.*
3. *With G_T^1, resp. G_T^2, and the generated $|P^1|$-terminal, resp. $|P^2|$-terminal, graph G^1, resp. G^2, make a new subproblem $[G^1, G_T^1]$, resp. $[G^2, G_T^2]$.*



(a) A six-terminal graph. (b) A topology graph. (c) A feasible mapping.

Figure 3: Example of a topology graph with product operator.

Fig. 4 shows the two subproblems generated from the example shown in Fig. 3 (a) and (b) with the hierarchy tree shown in Fig. 2 (b) and (c). In the first step of Rule 3, v_1 is specified into p_1^1 of G_T^1 and p_1^2 of G_T^2, v_2 into p_2^1 and p_1^2, v_3 into p_3^1 and p_1^2, v_4 into p_1^1



(a) A subproblem with a three-terminal graph. (b) A subproblem with a two-terminal graph.

Figure 4: Two-subproblem example in the product operator case.

and p_2^2, v_5 into p_2^1 and p_2^2, and v_6 into p_3^1 and p_2^2. Then the nodes specified into the same point are combined into a single node. For the subproblem shown in Fig. 4 (a), v_1 and v_4 become v_1' which is specified into p_1^1, v_2 and v_5 become v_2' which is specified into p_2^1, and v_3 and v_6 become v_3' which is specified into p_3^1. Similarly, for the subproblem shown in Fig. 4 (b), v_1, v_2, and v_3 become v_1' which is specified into p_1^2, and v_4, v_5, and v_6 become v_2' which is specified into p_2^2. Finally, with G_T^1 and G_T^2, and the generated 3-terminal graph G^1 and the 2-terminal graph G^2, we make two subproblem $[G^1, G_T^1]$ and $[G^2, G_T^2]$.

If $M_f^{1\prime}$ and $M_f^{2\prime}$ are feasible mappings for the generated subproblems $[G^1, G_T^1]$ and $[G^2, G_T^2]$, let M_f^1 and M_f^2 be the mappings obtained from $M_f^{1\prime}$ and $M_f^{2\prime}$ by translating each mapping from a super node to a point into mappings from the associated component nodes to the point. We determine a mapping M for the original problem with the following rule, Rule 4:

Rule 4 *$M(x) = w$ if $p_w = A_\odot(p_i^2, p_j^1)$, $M_f^1(x) = j$, and $M_f^2(x) = i$.*

It is not difficult to see that the mapping M obtained from Rule 4 is a feasible mapping.

Lemma 5 *Given $[G, G_T]$ with $G_T = G_T^1 \odot G_T^2$, assume that $[G^1, G_T^1]$ and $[G^2, G_T^2]$ are the subproblems of $[G, G_T]$ obtained by applying Rule 3. Also assume that M_f is a feasible mapping of $[G, G_T]$ by applying Rule 4 with M_f^1 and M_f^2 which are feasible mappings of $[G^1, G_T^1]$ and $[G^2, G_T^2]$. The cost of the mapping M_f can be computed as follows:*

$$Cost(M_f) = Cost(M_f^1) + Cost(M_f^2). \quad (1)$$

Proof: The contribution of an edge (v_x, v_y) to $Cost(M_f)$ is $w_{x,y} \cdot d_{M_f(x),M_f(y)}$. By Rule 4 $p_{M_f(x)} = A_\odot(p^2_{M_f^2(x)}, p^1_{M_f^1(x)})$ and $p_{M_f(y)} = A_\odot(p^2_{M_f^2(y)}, p^1_{M_f^1(y)})$. The distance $d_{M_f(x),M_f(y)}$ is the sum of two distances: $d_{M_f^1(x),M_f^1(y)}$ in G_T^1 and $d_{M_f^2(x),M_f^2(y)}$ in G_T^2. Consequently the cost of the mapping M_f, $Cost(M_f)$, is

$$\begin{aligned}
& \sum_{x<y} w_{x,y} \cdot d_{M_f(x),M_f(y)} \\
= & \sum_{x<y} w_{x,y} \cdot \left(d_{M_f^1(x)^1,M_f^1(y)} + d_{M_f^2(x),M_f^2(y)} \right) \\
= & \sum_{x<y} w_{x,y} \cdot d_{M_f^1(x),M_f^1(y)} + \sum_{x<y} w_{x,y} \cdot d_{M_f^2(x),M_f^2(y)} \\
= & \; Cost(M_f^1) + Cost(M_f^2).
\end{aligned}$$

This proves the lemma. □

Lemma 5 shows that we can find an optimal solution for the original problem $[G, G_T]$ if the subproblems $[G^1, G_T^1]$ and $[G^2, G_T^2]$ can be solved optimally. Therefore a GCKP instance with $G_T = G_T^1 \odot G_T^2$ can be divided into two independent GCKP instances which are associated with G_T^1 and G_T^2 respectively. In case of hypercube topology graphs, G_T^1 is K_2. That means the subproblem is two-terminal graph and it can be solved optimally using existing max-flow min-cut algorithms.

The overall algorithm for GCKP can be described in the following:

Algorithm : HYPER
Input : $G = (V, E)$ and $Q_q = (P, L)$
with a hierarchy tree H
Output: a mapping function $M(\cdot)$ //M: $v_i \rightarrow p_{M(i)}$//
if the top level operator is $\oplus$ **then**
// the topology of subproblem is K_2 //
find a minimum cut $C = (V_1, V_2)$ in the generated two-terminal graph;
make two subproblems $[G^1, G_T^1]$ and $[G^2, G_T^2]$ with C;
for $v_x \in V$
determine $M_f(x)$ with the cut result;
endfor
return with $M_f(\cdot)$;
else // $G_T = G_T^1 \odot G_T^2$ //
// apply Rule 3 //
make two subproblems $[G^1, G_T^1]$ and $[G^2, G_T^2]$ as explained in Rule 3;
$M_f^{1\prime}(\cdot)$ = **HYPER**(G_1, K_2);
$M_f^{2\prime}(\cdot)$ = **HYPER**(G_2, Q_{n-1});
Get M_f^1 and M_f^2 by extending each super node into its component nodes;
for $v_x \in V$
determine $M_f(x)$ by Rule 4;
endfor
return with $M_f(\cdot)$;

endif
end. // exit with the mapping $M_f(\cdot)$ //

Given $[G,]$, the algorithm HYPER recursively explores the hierarchy tree corresponding to Q_q. Whenever a topology graph is K_2, the algorithm finds a cut which can be used to determine a partial mapping for the problem. This step is the most time-consuming step because each of the other steps can be implemented with linear complexity.

A minimum (s,t)-cut can be found in polynomial time by applying any existing maximum flow algorithm based on the max-flow min-cut theorem [2]. The Goldberg-Tarjan's algorithm [3] is the best known algorithm with time complexity $O(EV\log(V^2/E))$, where E and V are the number of edges and the number of nodes of a two-terminal graph, respectively. In the worst case, the complexity is $O(V^3)$ because E can be up to $V(V-1)/2$. Each two-terminal graph generated during algorithm execution has at most n nodes. Thus the time to find a minimum cut is not worse than $O(n^3)$.

The algorithm HYPER should apply the maximum flow algorithm q times because the number of K_2's is q in the hierarchy tree of Q_q. Therefore, GCKP can be solved in time not worse than $O(qn^3)$ with the Goldberg-Tarjan's network flow algorithm. There is much room to reduce the execution time of the algorithm HYPER. In fact, two subproblems corresponding to a bridge operator have less than n nodes. Ideally, the numbers of nodes, $|V_1|$ and $|V_2|$, are $n/2$. This can reduce a lot of running time. Assume that we generated several subproblems; and we should find a minimum cut in each subproblem. In this case we can find two minimum cuts for the subproblems at the same time by combining the source, resp. terminal, nodes into one source, resp. terminal, node and by making one two-terminal graph [6].

4 Conclusions

We have proposed a new k-cut problem, called the Geometrical k-cut problem, with the concept of geometrical partitioning. The problem is an extension of the existing (s,t)-cut problem and the specified k-cut problem. This problem is NP-hard and thus intractable for an arbitrary topology graph with a large number of points and a large number of nodes.

We have proposed an optimal algorithm for hypercube topologies. The hypercube topology is recursively defined with the graph operator *product* $\odot$ and K_2. Given k terminals and n nodes, we construct a set of two-terminal graphs using the topological properties of the given hypercube topology graph. The problem is solved by applying the well-known maximum flow algorithm to each of the two-terminal graphs. The cuts, found by the algorithm HYPER, of the two-terminal graphs determine the point into which each node is mapped and the sum of the weights of the cuts defines the total cost for the corresponding mapping. Consequently, the Geometrical k-cut problem with hypercube topologies can be solved in time no worse than $O(qn^3)$, where q is the dimension of hypercube graph, according to the Goldberg-Tarjan's maximum flow algorithm [3].

Acknowledgements: The research was supported by MIC(Ministry of Information and Communication) Korea under the 2007 ITRC(Information Technology Research Center)support program supervised by the IITA(Institute of Information Technology Assessment).

References:

[1] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow algorithms," *J. ACM*, vol. 19, no. 2, Apr. 1972, pp. 248–264.

[2] L. R. Ford and D. R. Fulkerson, *Flows in Networks,* Princeton, NJ: Princeton Univ. Press, 1962.

[3] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum flow problem," *J. of ACM*, vol. 35, no. 4, Oct. 1988, pp. 921–940.

[4] O. Goldschmidt and D. S. Hochbaum, "Polynomial algorithm for the k-cut problem," in *Proc. 29th Annu. Symp. on Found. of Comput. Sci.*, 1988, pp. 444–451.

[5] F. Harary, J. P. Hayes, and H. J. Wu, "A survey of the theory of hypercube graphs," *Comput. Math. Applic.*, vol. 15, July 1988, pp. 277–289.

[6] T. C. Hu, *Combinatorial Algorithms*, Addison-Wesley Publishing Company, 1982.

[7] K. F. Korth and A. Silberschatz, *Database System Concepts*, 2nd Ed., McGraw-Hill Inc., 1991.

[8] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.* **C-37**, no. 11, Nov. 1988, pp. 1384–1397.

[9] G. Vijayan, "Generalization of Min-Cut Partitioning to Tree Structures and Its Applications," *IEEE Trans. Comput.* **C-40**, no. 3, Mar. 1991, pp. 307–314.