Isolation branching: a branch and bound algorithm for the k-terminal cut problem

Mark Velednitsky & Dorit S. Hochbaum

ONLINE FIRST

Journal of Combinatorial Optimization

ISSN 1382-6905

J Comb Optim
DOI 10.1007/s10878-020-00534-y

Journal of Combinatorial Optimization
https://doi.org/10.1007/s10878-020-00534-y

Isolation branching: a branch and bound algorithm for the k-terminal cut problem

Mark Velednitsky[1] · Dorit S. Hochbaum[1]

Abstract
In the `k-terminal cut` problem, we are given a graph with edge weights and k distinct vertices called "terminals." The goal is to remove a minimum weight collection of edges from the graph such that there is no path between any pair of terminals. The `k-terminal cut` problem is NP-hard. The `k-terminal cut` problem has been extensively studied and a number of algorithms have been devised for it. Most are approximation algorithms. There are also fixed-parameter tractable algorithms, but none have been shown empirically practical. It is also possible to apply implicit enumeration using any integer programming formulation of the problem and solve it with a branch-and-bound algorithm. Here, we present a branch-and-bound algorithm for the `k-terminal cut` problem which does not rely on an integer programming formulation. Our algorithm employs "minimum isolating cuts" and, for this reason, we call our branch-and-bound algorithm *Isolation Branching*. In an empirical experiment, we compare the performance of Isolation Branching to that of a branch-and-bound applied to the strongest known integer programming formulation of `k-terminal cut`. The integer programming branch-and-bound procedure is implemented with Gurobi, a commercial mixed-integer programming solver. We compare the performance of the two approaches for real-world instances and simulated data. The results on real data indicate that Isolation Branching, coded in Python, runs an order of magnitude faster than Gurobi for problems of sizes of up to tens of thousands of vertices and hundreds of thousands of edges. Our results on simulated data also indicate that Isolation Branching scales more effectively. Though we primarily focus on creating a practical tool for `k-terminal cut`, as a byproduct of our algorithm we prove that the complexity of Isolation Branching is fixed-parameter tractable with respect to the size of the optimal solution, thus providing an alternative, constructive, and somewhat simpler, proof of this fact.

✉ Mark Velednitsky
marvel@berkeley.edu

Dorit S. Hochbaum
dhochbaum@berkeley.edu

[1] University of California, Berkeley, USA

Published online: 07 February 2020

Keywords k-terminal cut · Optimization · Branch-and-bound · Minimum isolating cut · Clustering

1 Introduction

In the `k-terminal cut` problem, we are given an graph with edge weights and k distinct vertices called "terminals." The goal is to remove a minimum weight collection of edges from the graph such that there is no path between any pair of terminals. The `k-terminal cut` problem is NP-hard (Dahlhaus et al. 1994). In the literature, the problem has also been referred to as the `multiterminal cut` problem or the `multiway cut` or `multicut` problem with k terminals.

The `k-terminal cut` problem has a number of applications. Specific application areas include distributing computational jobs in a parallel computing system (Goldberg et al. 1989), partitioning elements of a circuit into sub-circuits that will be put on different chips (Dahlhaus et al. 1994), scheduling tasks (Karypis and Kumar 1998), understanding transportation bottlenecks (Karypis and Kumar 1998), planning the "divide" step in divide-and-conquer algorithms (Hochbaum 1996), and even partitioning Markov Random Fields for computer vision (Boykov et al. 1998). More generally, graph cut problems, including `k-terminal cut`, have applications to graph clustering (Fern and Brodley 2004). Minimizing the weight of edges between clusters is equivalent to maximizing the weight within clusters. In a setting where the weights measure similarity between vertices, the result is a graph clustering procedure. Thus, `k-terminal cut` gives an explicit combinatorial objective function for supervised graph clustering.

A number of algorithms have been devised for the `k-terminal cut` problem. Most of the algorithms are approximation algorithms. There are also fixed-parameter tractable algorithms that solve the problem optimally in time that is polynomial when the value of the optimum is fixed, but none have been shown empirically practical.

The first approximation algorithm for `k-terminal cut` gave an approximation ratio of 2.0 (Dahlhaus et al. 1994). Improved approximation algorithms are based on the linear programming relaxation of the integer programming formulation of the problem known in the literature as the *geometric Integer Program* (Călinescu et al. 1998). A sequence of improved approximation algorithms delivered an approximation factor of 1.5 (Călinescu et al. 1998), followed by 1.3438 (Karger et al. 2004), followed by 1.32388 (Buchbinder et al. 2013). The best-to-date approximation factor is 1.2965 (Sharma and Vondrák 2014).

It is possible to apply implicit enumeration using any integer programming formulation of the problem and solve it with a branch-and-bound algorithm. We compare the performance of our algorithm to a branch-and-bound procedure based on the geometric Integer Programming formulation. The geometric Integer Program is the Integer Program that was proved to have the smallest integrality gap, assuming the Unique Games Conjecture (Manokaran et al. 2008).

There are several fixed-parameter tractable optimization algorithms for `k-terminal cut`. It was first proven in 2004 that `k-terminal cut` is fixed-parameter tractable with respect to the value of the optimal solution (Marx 2004).

That proof was not constructive. A constructive proof in the form of an algorithm was given in Chen et al. (2009) with running time $O(|OPT|4^{|OPT|}n^3)$, where $|OPT|$ is the weight of the optimal cut and n is the number of vertices in the graph. The algorithm of Chen et al. (2009) is of theoretical value and has never been implemented. A by-product of our algorithm is an alternative, constructive proof of the fixed-parameter tractability of `k-terminal cut`.

The algorithm devised here applies concepts used in the closely related `k-cut` problem. In the `k-cut` problem, there are no terminals. The goal in the `k-cut` problem is to remove a minimum weight collection of edges such that the resulting graph consists of k non-empty, disjoint connected components. The `k-cut` problem was proved to be easier than `k-terminal cut`: in Goldschmidt and Hochbaum (1994), the authors proved that `k-cut` is polynomial for fixed k, whereas Dahlhaus et al. (1994) showed that `k-terminal cut` is NP-hard even for $k = 3$.

The polynomial-time algorithm for `k-cut` introduced in Goldschmidt and Hochbaum (1994) relies on two building blocks which we will also use here: *seed sets* and *minimum isolating cuts*. A *seed set* is a set of vertices in the graph which we assume belong to the same component in an optimal solution. Given a set of seed sets, a *minimum isolating cut* is the minimum-weight (s, t)-cut which separates one seed set from the rest. For the `k-cut` problem, Goldschmidt and Hochbaum (1994) shows that if the "correct" set of $2k$ vertices are chosen as seeds, then the source set of the minimum isolating cut recovers one of the components in the optimal `k-cut`. In their algorithm, all $\sim \binom{n}{2k}$ possibilities are enumerated. Thus, the exponent of the running time polynomial depends quadratically on k.

Minimum isolating cuts have also been used in the 2-approximation algorithm for `k-terminal cut` presented in Dahlhaus et al. (1994). For each of the k terminals, consider the minimum isolating cut which separates the chosen terminal from the rest of the terminals. If we take the union of the edges which appear in all k of these minimum isolating cuts, then the result is a feasible k-terminal cut. It is shown in Dahlhaus et al. (1994) that the value of this solution is at most twice the value of the optimal k-terminal cut. Minimum isolating cuts have also been used to solve special instances of the `k-terminal cut` problem to optimality (Velednitsky 2019).

Our contributions in this paper are as follows:

1. We devise a branch-and-bound algorithm for the `k-terminal cut` problem, which does not rely on an integer programming formulation, and is demonstrated to be practical and scalable. Our algorithm employs minimum isolating cuts and, for this reason, we call our branch-and-bound algorithm *Isolation Branching*.
2. We conduct an empirical study of optimization procedures for `k-terminal cut`, in which the performance of Isolation Branching is compared to branch-and-bound on the geometric Integer Programming formulation of `k-terminal cut`. The Integer Programming branch-and-bound procedure is implemented with Gurobi, a commercial mixed-integer programming solver. The performance is evaluated for real-world instances and simulated data. The results on real data indicate that Isolation Branching, coded in Python, runs an order of magnitude faster than Gurobi on graphs with up to tens of thousands of vertices and hun-

dreds of thousands of edges. The results on simulated data indicate that Isolation Branching scales more effectively.

3. Though our primary motivation is developing a practical branch-and-bound algorithm for the `k-terminal cut` problem, we also prove that the running time of our algorithm is fixed-parameter tractable with respect to the size of the optimal solution. Thus, a byproduct of our algorithm is an alternative, constructive, and somewhat simpler proof of an already-known result: that the `k-terminal cut` problem is fixed-parameter tractable.

2 Preliminaries

The input to the `k-terminal cut` problem is an undirected graph $G = (V, E, w)$ with vertex set V, edge set E, weights w, and k terminals $s_1, \ldots, s_k \in V$. The graph G is assumed to have positive integer edge weights w_{ij} for $\{i, j\} \in E$. Throughout our algorithm, we maintain k disjoint sets, each of which contains one terminal, which we call the *seed* sets. The i^{th} seed set, S_i, is a subset of V which contains the terminal s_i and none of the other s_j $(j \neq i)$.

A collection of k subsets of V is a k-terminal cut if and only if the k sets partition the vertex set V: an edge is in the cut if and only if its endpoints are in two different sets. Our algorithm initializes with the minimal seed sets, containing only the respective terminals, $S_i = \{s_i\}$, and adds vertices as seeds until the seed sets form a partition. The tool used to add new vertices to the seed sets is the minimum isolating cut.

To introduce minimum isolating cuts, recall the minimum (s, t)-cut problem. Given a directed graph $\overrightarrow{G} = (V, A)$ and terminals s and t, a minimum (s, t)-cut is a partition of the set of vertices into a *source set* containing s and a *sink set* containing t such that the total weight of arcs from the source set to the sink set is minimized. It may be the case that there are several minimum cuts. In that case, a *maximal* source set is a source set not contained in the source set of any other minimum cut.

Given a maximum flow $\hat{f}$ in $\overrightarrow{G}$, the minimum (s, t)-cut with maximal source set can be identified in linear time. The residual graph with respect to $\hat{f}$ is defined as follows: if arc $(i, j) \in A$ has capacity c_{ij}, then the residual capacity from i to j is $c_{ij} - f_{ij}$ and the residual capacity from j to i is f_{ij}. Every arc that has positive residual capacity is a residual arc. The residual graph is $\overrightarrow{G_r} = (V, A_r)$, where A_r is the set of residual arcs. If we wanted to determine the minimum (s, t)-cut with *minimal* source set, we would identify the set of vertices reachable from s in the residual graph (using breadth-first search). These vertices would form the source set and the rest would form the sink set. To determine the minimum (s, t)-cut with *maximal* source set, we identify the set of vertices from which t can be reached in the residual graph (by reversing the arcs in the residual graph and using breadth-first search). These vertices form the sink set and the rest form the source set of the desired minimum cut with maximal source set.

For an undirected graph G, we may pose the minimum (s, t)-cut problem by transforming G into a directed graph: replace each edge $\{i, j\}$ with arcs (i, j) and (j, i), where the capacity of the new arcs equals the weight of the edge they replace.

Definition 1 *(Minimum Isolating Cut for S_i)* Given a collection of seed sets $\{S_1, \ldots, S_k\}$, a *minimum isolating cut* for S_i is a minimum cut with maximal source set which separates S_i from all the vertices in $\cup_{j \neq i} S_j$. The notation $\mathcal{I}(S_i)$ denotes the source set of this minimum isolating cut.

The problem of calculating a minimum isolating cut for S_i can be reduced to the problem of computing a minimum (s, t)-cut with maximal source set. This is accomplished by contracting all of the vertices in S_i into a single source vertex s and contracting all the vertices $\cup_{j \neq i} S_j$ into a single sink vertex t. Contracting S_i into s means replacing all the vertices in S_i with a single vertex s. Edges with one endpoint in S_i and another endpoint outside S_i are combined such that, for all $u \notin S_i$, $w(s, u) = \sum_{v \in S_i} w(v, u)$.

Next, we develop an understanding of how these seed sets relate to the optimal `k-terminal cut`:

Definition 2 *(Containment Property)* A collection of seed sets $(S_1, \ldots, S_k)$ is said to have the *containment property* if there exists an optimal `k-terminal cut` $(S_1^\star, \ldots, S_k^\star)$ such that $S_i \subseteq S_i^\star \quad \forall i$.

In Dahlhaus et al. (1994), prove the following lemma, which we have rephrased here:

Lemma 1 (Isolation Lemma) *Consider the collection of seed sets $S_i = \{s_i\}$. For any i,*

$$(\{s_1\}, \ldots, \{s_{i-1}\}, \mathcal{I}(\{s_i\}), \{s_{i+1}\}, \ldots, \{s_k\})$$

has the containment property in G.

As an example, consider Fig. 1. The unique optimal k-terminal cut has weight 8 (cutting the four edges that form the central square) while the four minimum isolating cuts for the terminals each have weight 3. The source sets of the four minimum isolating cuts are subsets of the source sets of the optimal k-terminal cut. The isolation lemma proves that this is always the case. In our analysis, we rely on a simple generalization:

Lemma 2 (Seed Set Isolation Lemma) *Consider a collection of seed sets $(S_1, \ldots, S_k)$ with the containment property in G. Then*

$$(S_1, \ldots, S_{i-1}, \mathcal{I}(S_i), S_{i+1}, \ldots, S_k)$$

has the containment property in G.

Proof Let $(S_1^*, S_2^*, \ldots, S_k^*)$ be an optimal `k-terminal cut` in G, with $S_i \subseteq S_i^*$ for each i, and let E_{OPT} be the edges of that cut. Merge the vertices of each S_i into their respective s_i to create the new graph G' with terminals s_i'. By the containment property, $S_i \subseteq S_i^*$, so none of the edges in E_{OPT} have both endpoints in the same S_i, so all of the edges in E_{OPT} still connect two distinct vertices in G'. Thus, E_{OPT} is still an optimal solution in G' to the `k-terminal cut` problem.

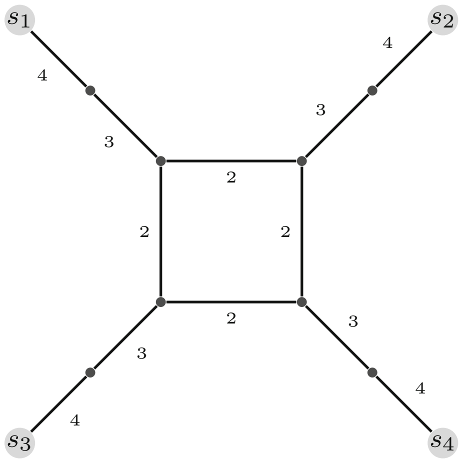

Fig. 1 Isolation lemma in a small graph

We apply the isolation Lemma (1) in G'. The minimum isolating cut $\mathcal{I}(s_i')$ in G' adds the same vertices as $\mathcal{I}(S_i)$ in G. That is,

$$\mathcal{I}(s_i') \setminus s_i' = \mathcal{I}(S_i) \setminus S_i.$$

From the isolation lemma, we have that $(s_1', \ldots, \mathcal{I}(s_i'), \ldots, s_k')$ has the containment property in G'.

$$\begin{aligned}&\mathcal{I}(s_i') \setminus s_i' \subseteq S_i^* \setminus S_i \\ &\quad \Longrightarrow \mathcal{I}(S_i) \setminus S_i \subseteq S_i^* \setminus S_i \\ &\quad \Longrightarrow \mathcal{I}(S_i) \subseteq S_i^*.\end{aligned}$$

We conclude that $(S_1, \ldots, \mathcal{I}(S_i), \ldots, S_k)$ has the containment property in G. □

3 Branch and bound

In our algorithm for `k-terminal cut`, we assign vertices to seed sets until all the vertices have been assigned. As long as the minimum isolating cuts provide non-trivial information, we use them to add unassigned vertices to seed sets. Otherwise, we "branch" by considering assigning a certain unassigned vertex to all possible seed sets. Following the branches where we make the "correct" assignment (where the vertex is assigned to the seed set to which it belongs in the optimal solution), we will reach the optimal solution. In branches where we make the "wrong" assignment, we will eventually arrive at a feasible solution that is not optimal. Using a bound we derive for this purpose, based on the 2-approximation from Dahlhaus et al. (1994), we can ignore many of these branches by proving that they will not lead to an optimal solution before expanding them.

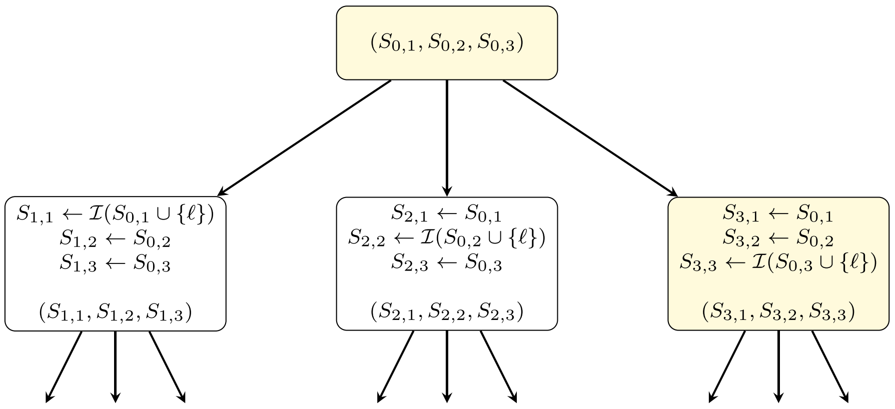
$(S_{0,1}, S_{0,2}, S_{0,3})$
$S_{1,1} \leftarrow \mathcal{I}(S_{0,1} \cup \{\ell\})$
$S_{1,2} \leftarrow S_{0,2}$
$S_{1,3} \leftarrow S_{0,3}$
$(S_{1,1}, S_{1,2}, S_{1,3})$
$S_{2,1} \leftarrow S_{0,1}$
$S_{2,2} \leftarrow \mathcal{I}(S_{0,2} \cup \{\ell\})$
$S_{2,3} \leftarrow S_{0,3}$
$(S_{2,1}, S_{2,2}, S_{2,3})$
$S_{3,1} \leftarrow S_{0,1}$
$S_{3,2} \leftarrow S_{0,2}$
$S_{3,3} \leftarrow \mathcal{I}(S_{0,3} \cup \{\ell\})$
$(S_{3,1}, S_{3,2}, S_{3,3})$

Fig. 2 Branch and bound tree for $k = 3$

3.1 Branching

Throughout, T will denote the incumbent branch-and-bound tree. At each node of the tree, $d \in T$, we will store a collection of pairwise disjoint seed sets $S_{d,i} \subset V$, $i \in \{1, \ldots, k\}$. For convenience, we will use *nodes* when referring to the branch-and-bound tree T and *vertices* when referring to V in the original graph G.

We will say that a vertex $\ell \in V$ is *unassigned* in $d \in T$ if it is not in any of the $S_{d,i}$. In our branching step, we choose an unassigned vertex ℓ in d and create k children of d in T by assigning ℓ to each of the $S_{d,i}$ in turn and computing the new minimum isolating cuts (with maximal source sets). If d does not have any unassigned vertices then it provides us with a feasible solution to the `k-terminal cut` instance. Algorithm 1 is the pseudo-code of the Isolation Branching algorithm. Figure 2 provides an illustration.

The following lemma shows that the containment property propagates down the tree:

Lemma 3 (Inheritance) *If $d \in T$ has the containment property, then at least one child of d has it.*

Proof Assume node d has the containment property. For all i, $S_{d,i} \subseteq S_i^*$. Let ℓ be the unassigned vertex chosen for branching. Assume $\ell \in S_j^*$. Then $S_{d,j} \cup \{\ell\} \subseteq S_j^*$, so the collection of sets

$$(S_{d,1}, \ldots, S_{d,j} \cup \{\ell\}, \ldots, S_{d,k})$$

has the containment property. By the seed set isolation lemma (Lemma 2), the collection of sets

$$(S_{d,1}, \ldots, \mathcal{I}(S_{d,j} \cup \{\ell\}), \ldots, S_{d,k})$$

also has the containment property. These are the seed sets for the j^{th} child of d. □

Algorithm 1 Isolation Branching

```
# initialization
$T \leftarrow \emptyset$.
add node 0 to $T$ (root node).
$d \leftarrow 0$.
for $i = 1 \dots k$ do
    $S_{0,i} \leftarrow \mathcal{I}(\{s_i\})$.
end for
# main loop
while node $d$ has unassigned vertices do
    vertex selection: choose vertex $\ell$ unassigned in $d$ (see Sect. 4.1).
    $D \leftarrow |T|$.
    for $i = 1 \dots k$ do
        add node $D + i$ to $T$ (child of node $d$).
        $S_{D+i,i} \leftarrow \mathcal{I}(S_{d,i} \cup \{\ell\})$.
        for $j \neq i$, $S_{D+i,j} \leftarrow S_{d,j}$.
    end for
    node selection: choose node $d$ unexplored in $T$ (see Sect. 4.1).
end while
# output
Return cut $(S_{d,1}, \dots, S_{d,k})$.
```

3.2 Bounding

We would prefer to only expand nodes which have a chance of leading to an optimal solution. At each node $d \in T$, we will consider the values of the functions

$$A(d) = |V| - \sum_{i=1}^{k} |S_{d,i}|.$$

$$L(d) = \frac{1}{2} \sum_{i=1}^{k} w(S_{d,i}, V \setminus S_{d,i}).$$

In words, $A(d)$ is the number of unassigned vertices at node d and $L(d)$ serves as a lower bound. To be more precise, $L(d)$ is half the sum of the weights of the minimum isolating cuts. If the collection of seed sets at d has the containment property, then the sum of minimum isolating cuts is known to be a 2-approximation to the optimal solution, so $L(d)$ must be less than the value of the optimal cut. The following two lemmas show that $L(d)$ can be used to cull branches even when the collection of seed sets does *not* have the containment property.

Lemma 4 *If $d \in T$ and $A(d) = 0$, then $L(d)$ is the value of the feasible* `k-terminal cut` *at node d.*

Proof Each edge in the feasible `k-terminal cut` is exactly double-counted inside the sum $L(d)$ since it appears in exactly two minimum isolating cuts. Multiplying by one-half returns the weight of the feasible `k-terminal cut`. □

Lemma 5 (Strict Monotonicity) *If $d_2 \in T$ is a descendant of $d_1 \in T$, then*

$$L(d_2) > L(d_1).$$

Proof It is sufficient to prove the inequality when d_2 is a child of d_1. Recall that $S_{d_1,i}$ is required to be a *maximal* source set for all i. Assume that from d_1 to d_2 we add our unassigned vertex ℓ to $S_{d_1,j}$ (and then take the minimum isolating cut). The size of the new minimum isolating cut must *strictly* increase, otherwise it contradicts the maximality of the previous source set. Formally,

$$w(S_{d_2,j}, V \setminus S_{d_2,j}) > w(S_{d_1,j}, V \setminus S_{d_1,j}).$$

For the rest ($i \neq j$),

$$w(S_{d_2,i}, V \setminus S_{d_2,i}) = w(S_{d_1,i}, V \setminus S_{d_1,i}).$$

In total, the value of the sum $L(d_2)$ strictly increases from $L(d_1)$. □

Together, these two lemmas give us the desired restriction. If we know that the weight of the optimal `k-terminal cut` is bounded above by B (for example, from finding a feasible solution), then we need not expand nodes where $L(d) \geq B$ since any nodes which descend from these nodes cannot be optimal.

3.3 Running time

The number of children of each tree node is k, because we consider adding the selected unassigned vertex to each of the k possible terminals. Furthermore, the value of

$$L(d) = \frac{1}{2} \sum_i w(S_{d,i}, V \setminus S_{d,i})$$

is at least $\frac{|OPT|}{2}$ at the root node ($d = 0$) and is strictly increasing with depth (Lemma 5). When the edge weights are integer, the increase must be at least $\frac{1}{2}$. $L(d)$ is exactly $|OPT|$ at a node with an optimal solution. Thus, $|OPT|$ is a bound on the depth of the tree. If we sum over the number of possible nodes at depths $1, 2, \ldots, |OPT|$, we see that the number of nodes considered is at most

$$1 + k + k^2 + \cdots + k^{|OPT|} < 2k^{|OPT|}.$$

Let $T(n, m)$ be the complexity of evaluating a minimum s, t-cut on a graph with n nodes and m edges. The complexity of our algorithm is thus $O(2k^{|OPT|}T(n, m))$. From this, we have fixed-parameter tractability.

Corollary 1 *When we can bound $|OPT|$ by a factor that does not depend on n (for example, graphs with terminals of bounded weighted degree), then the algorithm Isolation Branching runs in polynomial time.*

4 Empirical study

4.1 Isolation branching implementation

Our implementation is available online at https://github.com/marvel2010/k-terminal-cut and works as a Python package (ktcut). It represents graphs using NetworkX (Hagberg et al. 2008). We chose Python for ease of implementation and portability, even though it is not the fastest language in terms of its practical running time (Stein and Geyer-Schulz 2013).

There are two hyper-parameters which affect the performance of our branch-and-bound algorithm: the vertex-selection strategy and the node-selection strategy.

Branching Vertex Selection At each tree node, how do we decide which unassigned graph vertex to branch on to create the children nodes?
Branching Node Selection After expanding a node in the tree, how do we decide which node to expand next?

Branching Vertex: For choosing the branching vertex, we considered a few options. The options included choosing a vertex randomly, choosing the vertex farthest from an existing source set, or choosing the vertex of largest degree. Initial experiments suggested that the last strategy was best (largest degree), so our results use that strategy. The largest degree strategy makes some sense. If a vertex is forced to be in a particular source set, then its neighbors must either join the source set or the edge between them is cut. This means that, when a high-degree node is added to a source set, either the source set grows significantly in the next minimum isolating cut or the weight of the cut grows significantly. Either outcome is good, since it means that either the source sets grow quickly or we create tree nodes that do not need to be expanded (large values of $L(d)$). In our implementation, we contract source sets into a single terminal vertex at each node in the branch-and-bound tree. This allows subsequent minimum cuts to be evaluated on smaller graphs.

Branching Node: For the branching node, we chose the "least bound" strategy: that is, we always chose the node with the smallest lower bound, $L(d)$, with ties broken by fewer unassigned vertices, $A(d)$. We considered two other heuristics: choosing the node with smallest $A(d)$ and choosing the node with smallest $L(d) + A(d)$. We found that the "least bound" heuristic performed best, in practice.

4.2 Comparison to integer programming formulation branch-and-bound

To compare our Isolating Branching algorithm to integer programming branch-and-bound, we used Gurobi, a popular commercial software package for integer programming. The formulation to which we applied integer programming branch-and-bound is below. It is referred to in the literature as the *geometric* Integer Programming formulation (Călinescu et al. 1998). Assuming the Unique Games Conjecture, it has been proven that no other formulation can have a smaller integrality gap than this one (Manokaran et al. 2008).

The variable x_i^t is a binary variable: it is 1 if vertex i is assigned to terminal t and 0 otherwise.

$$\begin{aligned}
&\mathit{minimize} \sum_{\{i,j\}\in E} \sum_{t=1}^{k} \frac{1}{2} w_{ij} z_{ij}^t && \\
&\quad s.t. \quad z_{ij}^t \geq x_i^t - x_j^t && \forall \{i,j\} \in E, 1 \leq t \leq k \\
&\qquad\qquad z_{ij}^t \geq x_j^t - x_i^t && \forall \{i,j\} \in E, 1 \leq t \leq k \\
&\qquad\qquad \sum_{t=1}^{k} x_i^t = 1 && \forall i \in V \\
&\qquad\qquad x_i^t \in \{0,1\} && i \in V, 1 \leq t \leq k \\
&\qquad\qquad z_{ij}^t \in \{0,1\} && \{i,j\} \in E, 1 \leq t \leq k \\
&\qquad\qquad x_t^t = 1 && \forall 1 \leq t \leq k
\end{aligned} \tag{IP}$$

Strictly speaking, the constraint $z_{ij}^t \in \{0, 1\}$ does not need to be specified, since it is implied. However, we found that specifying it explicitly helped Gurobi solve instances faster. We suspect this is because the z_{ij}^t variables are good variables to branch on. All of Gurobi's hyper-parameters were set to their default values.

4.3 Hardware

The experiments were run in a single thread on a laptop with an Intel $i7 - 3520M$ processor and 16GB of RAM.

4.4 Data sets

Real Data Sets: The twenty-four real-world data sets we used for experimentation were gathered from two sources. The first source was the DIMACS Implementation Challenge. According to the website, "These real-world graphs are often used as benchmarks in the graph clustering and community detection communities." These data sets are available online at https://www.cc.gatech.edu/dimacs10/. The second source was KONECT, an online repository of popular graph datasets. These data sets are available online at http://konect.uni-koblenz.de/.

In most of the data sets, the graphs are already connected. In the rest, we only considered the largest connected component, otherwise the `k-terminal cut` problem decomposes into smaller problems on each component.

Simulated Data Sets: To systematically study the running time scaling of our Isolation Branching algorithm, we used simulated graphs. It has been observed that many real-world graphs, from social networks to computer networks to metabolic networks, exhibit both a power-law degree distribution and high clustering (Holme and Kim 2002). The Power Cluster model, introduced in Holme and Kim (2002), generates random graphs which exhibit both of these properties. NetworkX includes a tool for randomly generating graphs according to the Powerlaw Cluster model with three

Table 1 Real Data, 5 Terminals, Running Times

Alias	Instance properties			Integer programming	Isolation Branching	Improvement ratio
	Category	#Vertices	#Edges	CPU seconds	CPU seconds	
pdzbase	Metabolic	161	209	0.33	0.034	9.71
adjnoun	Lexical	112	425	0.52	0.09	5.78
polbooks	Misc	105	441	0.5	0.14	3.57
ChicagoRegional	Infrastructure	823	822	1.38	0.185	7.46
netscience	Coauthorship	379	914	1.14	0.192	5.94
euroroad	Infrastructure	1039	1305	2.35	0.942	2.49
propro	Metabolic	1458	1948	3.78	0.175	21.60
celegans_metabolic	Metabolic	453	2025	2.54	0.089	28.54
jazz	HumanSocial	198	2742	3.46	0.104	33.27
ego-facebook	Social	2888	2981	8.98	0.344	26.10
email	Communication	1133	5451	9.23	4.76	1.94
vidal	Metabolic	2883	6007	9.88	0.585	16.89
powergrid	Infrastructure	4941	6594	24.2	25.5	0.95
petster-friendships	Social	1788	12476	19.7	0.711	27.71
as20000102	Computer	6474	12572	21.4	1.87	11.44
hep-th	Coauthorship	5835	13815	17.7	0.686	25.80
petster-hamster	Social	2000	16098	29	1.08	26.85
polblogs	Hyperlink	1222	16714	20.5	0.611	33.55
arenas-pgp	OnlineContact	10680	24316	45	2.24	20.09
as-22july06	Computer	22963	48436	74.3	5.78	12.85
as-caida20071105	Computer	26475	53381	76.2	5.45	13.98
astro-ph	Coauthorship	14845	119652	160	5.22	30.65
reactome	Metabolic	5973	145778	179	8.1	22.10
ca-AstroPh	Coauthorship	17903	196972	240	6.81	35.24

The running time, in CPU seconds, of Isolation Branching versus Integer Programming with Gurobi in instances with five terminals. Both algorithms were run to optimality on all instances. The instances are sorted by the number of edges. The "improvement ratio" is the running time of Integer Programming divided by the running time of Isolation Branching

parameters: the number of vertices, the number of random edges to add for each vertex, and the probability of creating a triangle. In our scaling experiment, we vary the first parameter (the number of vertices) while leaving the latter two fixed at 10 and 0.1, respectively.

Terminals: In the data sets, terminals are not specified. In order to find suggested terminals, do the following: first, we perform spectral clustering on the graph to get an approximate clustering (by performing k-means clustering on the spectral embedding of the graph). Next, we choose the largest-degree vertex in each approximate cluster and set those vertices to be our k terminals.

Table 2 Real Data, 10 Terminals, Running Times

Alias	Instance Properties			Integer Programming CPU Seconds	Isolation Branching CPU Seconds	Improvement Ratio
	Category	#Vertices	#Edges			
pdzbase	Metabolic	161	209	0.66	0.07	9.43
adjnoun	Lexical	112	425	0.99	1.32	0.75
polbooks	Misc	105	441	1.01	15	0.07
ChicagoRegional	Infrastructure	823	822	2.28	0.188	12.13
netscience	Coauthorship	379	914	3.27	0.251	13.03
euroroad	Infrastructure	1039	1305	4.43	7.59	0.58
propro	Metabolic	1458	1948	5.64	0.182	30.99
celegans_metabolic	Metabolic	453	2025	5.73	0.165	34.73
jazz	HumanSocial	198	2742	6.61	1.05	6.30
ego-facebook	Social	2888	2981	7.76	61.5	0.13
email	Communication	1133	5451	15	208	0.07
vidal	Metabolic	2883	6007	16.9	0.385	43.90
powergrid	Infrastructure	4941	6594	440	392	1.12
petster-friendships	Social	1788	12476	34.6	0.479	72.23
as20000102	Computer	6474	12572	41.3	1.98	20.86
hep-th	Coauthorship	5835	13815	40.9	0.87	47.01
petster-hamster	Social	2000	16098	38.4	0.522	73.56
polblogs	Hyperlink	1222	16714	47	0.618	76.05
arenas-pgp	OnlineContact	10680	24316	64.3	1.71	37.60
as-22july06	Computer	22963	48436	253	8.55	29.59
as-caida20071105	Computer	26475	53381	209	9.75	21.44
astro-ph	Coauthorship	14845	119652	338	7.41	45.61
reactome	Metabolic	5973	145778	552	6.98	79.08
ca-AstroPh	Coauthorship	17903	196972	1120	7.2	155.56

The running time, in CPU seconds, of Isolation Branching versus Integer Programming with Gurobi in instances with ten terminals. Both algorithms were run to optimality on all instances. The instances are sorted by the number of edges. The "improvement ratio" is the running time of Integer Programming divided by the running time of Isolation Branching

4.5 Results

Before presenting the main comparison, we break down the performance of Isolation Branching. On the twenty-four real data sets, we report the number of minimum isolating cuts calculated, as well as the value of the optimal objective. The analysis is repeated in real graphs with five terminals (Appendix, Table 4) and with ten terminals (Appendix, Table 5). The graphs and method for choosing the terminals were described in Sect. 4.4.

One property of the objective function of `k-terminal cut` is that it can occasionally lead to solutions where most of the graph is assigned to one component of

Table 3 Simulated Data, 5 Terminals, Running Times

Instance Properties #Vertices	Integer programming		Isolation Branching		Improvement ratio
	Avg CPU time	Deviation	Avg CPU time	Deviation	
1000	13	0.3	17	4.3	0.74
2000	50	15.0	56	12.3	0.90
3000	192	86.6	65	17.6	2.94
4000	198	75.4	102	24.7	1.94
5000	585	161.0	234	49.6	2.50
6000	756	168.9	171	41.7	4.42
7000	1791	396.2	345	64.5	5.19
8000	2999	1322.6	479	82.0	6.26
9000	6573	1625.3	466	80.3	14.11

The average and standard deviation running time of Isolation Branching versus Integer Programming, measured in CPU seconds, on simulated data sets with five terminals, generated according to the Powerlaw Cluster model with 10 new edges per vertex and probability 0.1 of creating a triangle. For each size graph, 10 random instances were generated. Both algorithms were run to optimality on all instances. The "improvement ratio" is the running time of Integer Programming divided by the running time of Isolation Branching

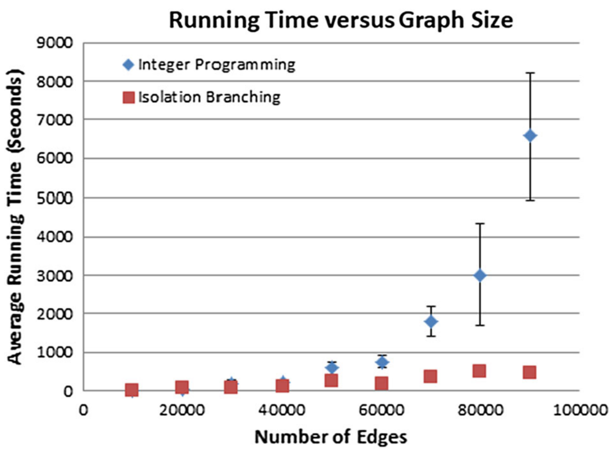

Fig. 3 The average running time of our Isolation Branching versus gurobi integer programming on ten random instances of `k-terminal cut` generated using the PowerlawCluster model

the partition. Since this is of interest, in addition to reporting the value of the optimal solution we also report the fraction of the graph which is in the largest partition of the optimal solution.

We find that the running time of Isolation Branching is correlated with the size of the graph and the number of minimum isolating cuts performed. It does not appear to be correlated with the properties of the optimal solution we considered.

Next, we compare Isolation Branching to Integer Programming with Gurobi. We compare on the twenty-four real-world test graphs, first with five terminals (Table 1) and then with ten terminals (Table 2). Isolation Branching is faster than Integer Programming on twenty-three of the twenty-four instances with five terminals and nineteen of the twenty-four instances with ten terminals. Isolation Branching provides a median speedup of $18\times$ in instances with 5 terminals and a median speedup of $26\times$ in instances with 10 terminals.

To systematically investigate the scaling of Isolation Branching versus Integer Programming, we consider random instances (as described in Sect. 4.4). Properties of the optimal solution can be found in Table 6. Running time comparisons can be found in Table 3 and Fig. 3. The running time is the average running time of each algorithm across ten randomly generated `k-terminal cut` instances. The error bars in the figure reflect the standard deviation of running time across those instances.

As in the real data sets, Isolation Branching scales better than Integer Programming to large instances. In instances with $30,000$ edges, Isolation Branching provides a factor of $3\times$ speedup over Gurobi. In instances with $90,000$ edges, the speedup is a factor of $14\times$. We believe that this behavior is largely explained by the slow growth in the number of minimum isolating cuts performed by Isolating Branching. In instances with $\sim 30,000$ edges, Isolation Branching performs 70 minimum isolating cuts on average. In instances with $\sim 90,000$ edges, Isolation Branching performs only 130 minimum isolating cuts on average.

5 Conclusions

In this paper, we introduce Isolation Branching, a new branch-and-bound algorithm devised for solving the `k-terminal cut` problem. In the empirical study, we demonstrate that Isolation Branching offers improvements of an order of magnitude over solving the Integer Program with Gurobi, especially on large graphs. Using simulated data, we demonstrate that the Isolation Branching algorithm scales better from small to large instances. Our code is available online at https://github.com/marvel2010/k-terminal-cut.

An advantage of our algorithm is that it uses only minimum (s, t)-cuts, avoiding the use of linear programming. As a byproduct of our analysis of the running time of Isolation Branching, we offer an alternative proof that `k-terminal cut` is fixed-parameter tractable with respect to the size of the optimal solution.

Acknowledgements Funding was provided by the National Physical Science Consortium.

Appendix

A Tables

See appendix Tables 4, 5 and 6.

Table 4 Real Data, 5 Terminals, Properties of Isolation Branching

Alias	Instance properties			Isolation branching		Optimal solution	
	Category	#Vertices	#Edges	CPU seconds	#Isolating cuts	Objective	Max component
pdzbase	Metabolic	161	209	0.034	5	8	0.652
adjnoun	Lexical	112	425	0.09	10	53	0.884
polbooks	Misc	105	441	0.14	75	50	0.514
ChicagoRegional	Infrastructure	823	822	0.185	10	4	0.417
netscience	Coauthorship	379	914	0.192	10	13	0.335
euroroad	Infrastructure	1039	1305	0.942	50	14	0.739
propro	Metabolic	1458	1948	0.175	5	6	0.974
celegans_metabolic	Metabolic	453	2025	0.089	5	84	0.956
jazz	HumanSocial	198	2742	0.104	5	67	0.960
ego-facebook	Social	2888	2981	0.344	5	7	0.631
email	Communication	1133	5451	4.76	35	136	0.994
vidal	Metabolic	2883	6007	0.585	5	6	0.957
powergrid	Infrastructure	4941	6594	25.5	75	19	0.789
petster-friendships	Social	1788	12476	0.711	5	4	0.987
as20000102	Computer	6474	12572	1.87	5	82	0.961
hep-th	Coauthorship	5835	13815	0.686	5	7	0.988
petster-hamster	Social	2000	16098	1.08	5	17	0.973
polblogs	Hyperlink	1222	16714	0.611	5	309	0.992
arenas-pgp	OnlineContact	10680	24316	2.24	5	17	0.965
as-22july06	Computer	22963	48436	5.78	5	187	0.923

Table 4 continued

Alias	Instance properties			Isolation branching		Optimal solution	
	Category	#Vertices	#Edges	CPU seconds	#Isolating cuts	Objective	Max component
as-caida20071105	Computer	26475	53381	5.45	5	138	0.938
astro-ph	Coauthorship	14845	119652	5.22	5	38	0.988
reactome	Metabolic	5973	145778	8.1	5	63	0.992
ca-AstroPh	Coauthorship	17903	196972	6.81	5	82	0.996

Properties of Isolation Branching in instances with five terminals: the number of minimum isolating cuts performed, the value of the optimal solution, and the fraction of the graph in the largest component of the optimal partition. The instances are sorted by the number of edges

Table 5 Real Data, 10 Terminals, Properties of Isolation Branching

Alias	Instance Properties			Isolation Branching		Optimal Solution	
	Category	#Vertices	#Edges	CPU Seconds	#Isolating Cuts	Objective	Max Component
pdzbase	Metabolic	161	209	0.07	30	19	0.261
adjnoun	Lexical	112	425	1.32	120	84	0.804
polbooks	Misc	105	441	15	1530	93	0.438
ChicagoRegional	Infrastructure	823	822	0.188	20	9	0.142
netscience	Coauthorship	379	914	0.251	10	27	0.235
euroroad	Infrastructure	1039	1305	7.59	300	25	0.612
propro	Metabolic	1458	1948	0.182	10	18	0.892
celegans_metabolic	Metabolic	453	2025	0.165	10	125	0.934
jazz	HumanSocial	198	2742	1.05	20	182	0.934
ego-facebook	Social	2888	2981	61.5	12320	103	0.262
email	Communication	1133	5451	208	1140	248	0.976
vidal	Metabolic	2883	6007	0.385	10	12	0.948
powergrid	Infrastructure	4941	6594	392	1810	35	0.293
petster-friendships	Social	1788	12476	0.479	10	153	0.978
as20000102	Computer	6474	12572	1.98	10	296	0.942
hep-th	Coauthorship	5835	13815	0.87	10	21	0.968
petster-hamster	Social	2000	16098	0.522	10	28	0.960
polblogs	Hyperlink	1222	16714	0.618	10	27	0.981
arenas-pgp	OnlineContact	10680	24316	1.71	10	26	0.952
as-22july06	Computer	22963	48436	8.55	10	508	0.911

Table 5 continued

Alias	Instance Properties			Isolation Branching		Optimal Solution	
	Category	#Vertices	#Edges	CPU Seconds	#Isolating Cuts	Objective	Max Component
as-caida20071105	Computer	26475	53381	9.75	10	537	0.887
astro-ph	Coauthorship	14845	119652	7.41	10	54	0.982
reactome	Metabolic	5973	145778	6.98	20	75	0.986
ca-AstroPh	Coauthorship	17903	196972	7.2	10	96	0.993

Properties of Isolation Branching in instances with ten terminals: the number of minimum isolating cuts performed, the value of the optimal solution, and the fraction of the graph in the largest component of the optimal partition. The instances are sorted by the number of edges

Table 6 Simulated Data, 5 Terminals, Properties of Isolation Branching

Instance properties #Vertices	Isolation branching		
	Avg CPU time	Deviation	Avg #Isolating cuts
1000	17	4.3	67
2000	56	12.3	99
3000	65	17.6	69
4000	102	24.7	81
5000	234	49.6	138
6000	171	41.7	81
7000	345	64.5	135
8000	479	82.0	126
9000	466	80.3	131

Properties of Isolation Branching on simulated data sets with five terminals, generated according to the Powerlaw Cluster model, with 10 new edges per vertex and probability 0.1 of creating a triangle

References

Boykov Y, Veksler O, Zabih R (1998) Markov random fields with efficient approximations. In: 1998 Proceedings. 1998 IEEE computer society conference on Computer vision and pattern recognition. pp. 648–655

Buchbinder N, Naor JS, Schwartz R (2013) Simplex partitioning via exponential clocks and the multiway cut problem. In: Proceedings of the forty-fifth annual ACM symposium on theory of computing. pp. 535–544

Călinescu G, Karloff H, Rabani Y (1998) An improved approximation algorithm for multiway cut. In: Proceedings of the thirtieth annual ACM symposium on theory of computing. pp. 48–52

Chen J, Liu Y, Lu S (2009) An improved parameterized algorithm for the minimum node multiway cut problem. Algorithmica 55(1):1–13

Dahlhaus E, Johnson DS, Papadimitriou CH, Seymour PD, Yannakakis M (1994) The complexity of multiterminal cuts. SIAM J Comput 23(4):864–894

Fern XZ, Brodley CE (2004) Solving cluster ensemble problems by bipartite graph partitioning. In: Proceedings of the twenty-first international conference on machine learning. p. 36

Goldberg AV, Tardos É, Tarjan RE (1989) Network flow algorithms. Technical report, Cornell University Operations Research and Industrial Engineering

Goldschmidt O, Hochbaum DS (1994) A polynomial algorithm for the k-cut problem for fixed k. Math Oper Res 19(1):24–37

Hagberg A, Swart P, S Chult D (2008) Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States)

Hochbaum DS (1996) Approximation algorithms for NP-hard problems. PWS Publishing Co, Gretna

Holme P, Kim BJ (2002) Growing scale-free networks with tunable clustering. Phys Rev E 65(2):026107

Karger DR, Klein P, Stein C, Thorup M, Young NE (2004) Rounding algorithms for a geometric embedding of minimum multiway cut. Math Oper Res 29(3):436–461

Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 20(1):359–392

Manokaran R, Naor JS, Raghavendra P, Schwartz R (2008) Sdp gaps and ugc hardness for multiway cut, 0-extension, and metric labeling. In: Proceedings of the fortieth annual ACM symposium on theory of computing. pp. 11–20

Marx D (2004) Parameterized graph separation problems. In: International workshop on parameterized and exact computation. pp. 71–82. Springer

Sharma A, Vondrák J (2014) Multiway cut, pairwise realizable distributions, and descending thresholds. In: Proceedings of the forty-sixth annual ACM symposium on theory of computing. pp. 724–733

Stein M, Geyer-Schulz A (2013) A comparison of five programming languages in a graph clustering scenario. J Univers Comput Sci 19(3):428–456

Velednitsky M (2019) Solving $(k-1)$-stable instances of k-terminal cut with isolating cuts. In: International conference on combinatorial optimization and applications. springer