

A scalable clustering algorithm to approximate graph cuts

Leo Suchan^{ID}, Housen Li^{ID}, and Axel Munk^{ID} *†

April 10, 2024

Abstract

Due to their computational complexity, graph cuts for cluster detection and identification are used mostly in the form of convex relaxations. We propose to utilize the original graph cuts such as Ratio, Normalized or Cheeger Cut to detect clusters in weighted undirected graphs by restricting the graph cut minimization to *st*-MinCut partitions. Incorporating a vertex selection technique and restricting optimization to tightly connected clusters, we combine the efficient computability of *st*-MinCuts and the intrinsic properties of Gomory-Hu trees with the cut quality of the original graph cuts, leading to linear runtime in the number of vertices and quadratic in the number of edges. Already in simple scenarios, the resulting algorithm Xist is able to approximate graph cut values better empirically than spectral clustering or comparable algorithms, even for large network datasets. We showcase its applicability by segmenting images from cell biology and provide empirical studies of runtime and classification rate.

1 Introduction

The detection and identification of clusters in datasets is a fundamental task of data analysis, with applications including image segmentation (Senthilnath, Sindhu, and Omkar, 2014; Heuvel, Mandl, and Hulshoff Pol, 2008; Wang et al., 2020), machine learning (Chew and Cahill, 2015; Tang et al., 2016;

*The authors are with the Institute of Mathematical Stochastics, Georg August University of Göttingen, Göttingen, Germany

†H. Li and A. Munk are with the Cluster of Excellence “Multiscale Bioimaging: from Molecular Machines to Networks of Excitable Cells” (MBExC)

Malioutov and Barzilay, 2006), and parallel computing (Yu, Shen, and Z. Hu, 2016; Peng et al., 2014; Chen et al., 2011). The construction and partitioning of a graph is key to many popular methods of cluster detection. Prominent graph cuts such as Ratio Cut Hagen and Kahng, 1992, Normalized Cut J. Shi and Malik, 2000 or Cheeger Cut Cheeger, 1971 are designed to partition a graph in a “balanced” way while providing an intuitive geometric interpretation. Ratio Cut, for instance, attempts to balance cluster sizes, and Normalized Cut and Cheeger Cut aim for equal volumes of the resulting partitions. However, as the computation of the above mentioned cuts is an NP-hard problem (see e.g. Mohar, 1989; Bui and Jones, 1992; J. Shi and Malik, 2000; Šíma and Schaeffer, 2006), attention has shifted primarily to their various convex relaxations and regularizations. Most prominent are spectral clustering techniques, which can be viewed as a convex relaxation of the optimization problem underlying Normalized and Cheeger Cut Luxburg, 2007. They offer a quick and simple way to partition a graph, with a worst-case runtime cubic in the number of vertices Luxburg, 2007, and their practical success has been demonstrated in numerous applications. Consequently, they have been the subject of extensive theoretical analysis (Luxburg, Belkin, and Bousquet, 2008; Maier, Luxburg, and Hein, 2013; García Trillos and Slepčev, 2018) as well as the inspiration for a multitude of specialized algorithms (Ng, Jordan, and Weiss, 2001; Nascimento and Carvalho, 2011; Zhong and Pun, 2022). However, it is well known that spectral clustering does not always yield a qualitatively sensible partition, the most famous example being the so-called “cockroach graph” and its variants Guattery and Miller, 1998. More significantly, it has been shown that spectral clustering possesses fundamental flaws in detecting clusters of different scales Nadler and Galun, 2007.

To overcome these issues we propose a different approach which combines the versatility and quality of graph cuts with the most significant trait of spectral clustering, its quick and simple computability. While spectral clustering builds on Normalized Cut and Cheeger Cut alone, the suggested algorithm *Xist* is applicable to any balanced graph cut functional (also sometimes called sparsest cut), thus making it adaptable and scalable. In contrast to spectral clustering, our algorithm approximates graph cuts not by means of relaxation of the graph cut functional, but instead by restricting minimization onto a particularly designed subset of partitions. This subset is constituted by *st*-MinCuts, where *s* and *t* runs through a certain subset of vertices. These partitions can be computed very fast through max flows (Orlin, 2013) and the well-known duality of the max-flow and min-cut problems. Hence, our proposed algorithm *Xist* to a large extent preserves the combinatorial nature of the problem while retaining a worst computational complexity *quadratic* in the number of vertices and *linear* in the number of edges ([Theorem 3.2](#)).

The rest of the paper is organized as follows. [Section 2](#) introduces the basic notation. The proposed 2-way cut algorithm is detailed in [Section 3](#), together with its theoretical properties. In [Section 4](#), we study the performance of the proposed algorithm on simulated and real world datasets and introduce a multiway cut extension. [Section 5](#) provides a discussion of further extensions and concludes the paper. Technical proofs are given in the [Appendix](#).

2 Definitions and notation

We consider simple, undirected, and weighted graphs, denoted as $G = (V, E, \mathbf{W})$, with V the vertex set, E the set of edges and $\mathbf{W} = (w_{ij})_{i,j \in V}$ the weight matrix. Each entry w_{ij} equals the weight of edge $\{i, j\}$ if $\{i, j\} \in E$ and zero otherwise. Thus, \mathbf{W} is symmetric and has a zero diagonal.

Definition 1 (Graph cut). For a simple, undirected, weighted graph $G = (V, E, \mathbf{W})$, we define the (*balanced*) *graph cut* (or: *sparsest cut*) XC of G as:

$$\begin{aligned} \text{XC}(G) &:= \min_{S \subset V} \text{XC}_S(G) \\ \text{where } \text{XC}_S(G) &:= \sum_{i \in S, j \in S^c} \frac{w_{ij}}{\text{Bal}_G(S, S^c)} \end{aligned}$$

for $S \subset V$. Here, $S^c := V \setminus S$ is the complement of S , and XC and $\text{Bal}_G(S, S^c)$ serve as placeholders for the graph cut (see [Table 1](#)) and its corresponding balancing term, respectively.

[Table 1](#) lists the balancing terms for Minimum Cut (MinCut), Ratio Cut, Normalized Cut (NCut) and Cheeger Cut. In general, these balancing terms can depend on the underlying graph structure, the partition S as well as the weight matrix \mathbf{W} . For any partition $S \subset V$, define its *size* as $|S| := \#\{i \in S\}$ and its *volume* as $\text{vol}(S) := \sum_{i \in S} \deg(i)$, where $\deg(i) := \sum_{j \in V} w_{ij}$ is the *degree* of vertex i . Assume $n := |V| < \infty$ and $m := |E| < \infty$.

Table 1: Definitions of common (balanced) graph cuts.

Cut name	XC	$\text{Bal}_G(S, S^c)$	Reference
Minimum Cut	MC	1	E.g. Cook et al., 1998
Ratio Cut	RC	$ S S^c $	Hagen and Kahng, 1992
Normalized Cut	NC	$\text{vol}(S) \text{vol}(S^c)$	J. Shi and Malik, 2000
Cheeger Cut	CC	$\min\{\text{vol}(S), \text{vol}(S^c)\}$	Cheeger, 1971

The disadvantage of (balanced) graph cuts is their computational complexity that is rooted in their combinatorial nature. It has been shown that the problem of computing NCut is NP-complete, see J. Shi and Malik, 2000, Appendix A, Proposition 1, and the idea behind the proof can be adapted for the other balanced cuts in [Table 1](#), namely Ratio Cut and Cheeger Cut (see Šíma and Schaeffer, 2006 for an alternative proof for the latter). Several other balanced graph cuts are also known to be NP-complete to compute, for instance, the graph cut with the balancing term $\text{Bal}_G(S, S^c) = \min\{|S|, |S^c|\}$, see Mohar, 1989. It should be noted that, due to its lack of a balancing term, MinCut is computable in polynomial time (see e.g. Gawrychowski and Weimann, 2024 for a randomized algorithm with high probability in $\mathcal{O}(m \log^2 n)$ time). For applications, however, MinCut is of limited relevance since it tends to separate a single vertex from the remainder of the graph Luxburg, 2007. Thus, all practically relevant graph cuts become non-computable even for problems of moderate sizes. This also remains the case for many relaxed versions D. Wagner and F. Wagner, 1993 and approximations of graph cuts up to a constant factor Bui and Jones, 1992. Sherman (2009) proposed an algorithm to approximate Ratio Cut on an unweighted graph up to a $\mathcal{O}(\sqrt{\log n})$ factor in $\tilde{\mathcal{O}}(n \cdot \text{polylog}(n))$ time, building upon previous results by Khandekar, S. Rao, and Vazirani, 2009; Arora and Kale, 2016, and attain the lower bound on the approximation factor as shown by Orecchia et al., 2008. To the best of our knowledge, however, these results do not extend to other balancing terms or to weighted graphs. As mentioned in the [Introduction](#), spectral clustering represents a notable exception as it can be regarded as a relaxation of Normalized and Cheeger Cut while still being computable in $\mathcal{O}(n^3)$ time for weighted graphs.

3 The algorithms

We first introduce the basic algorithm and then present a refined and vastly accelerated variant.

3.1 A basic algorithm for imitating graph cuts through *st-min* cuts

To retain the qualitative aspects of the (balanced) graph cuts themselves as much as possible we suggest to restrict the combinatorial optimization to a certain collection of partitions. Then, if such a collection of partitions is well-chosen, the original graph cut (i.e. the minimizer over all partitions) can be imitated on a qualitative level. For this purposes, we consider some

collections of *st*-MinCut partitions, i.e. the cuts that separate the two nodes s and t in V for $s, t \in V$. More precisely, an *st*-MinCut partition S_{st} is defined as

$$S_{st} \in \arg \min_{S \in \mathcal{S}_{st}^*} \text{MC}_S(G),$$

where $\mathcal{S}_{st}^* := \{S \subset V : s \in S, t \notin S\}$.

The partition S_{st} might not be unique. However, the nonuniqueness represents a fringe case that is highly unlikely to occur on real-world data; see [Section 4](#). Also notice that, by definition, $s \in S_{st}$ and $t \in S_{st}^c$ for any $s, t \in V$, $s \neq t$. This property will be important later. The fastest algorithms for computing an *st*-MinCut partition take $\mathcal{O}(nm)$ time for general graphs; for instance, one could use the algorithm suggested in [Orlin \(2013\)](#) if $m = \mathcal{O}(n^{1.06})$, and [Orlin and Gong \(2021\)](#) otherwise, see also [Appendix A.3](#).

Our idea is to restrict the graph cut minimization to the *st*-MinCut partitions. The reason for choosing this particular subset of partitions is twofold: First, it is computable in polynomial time as outlined above, and second, an *st*-MinCut partition is forced to separate two clusters if s and t belong to different clusters that are more tightly connected than the edges connecting them. In this scenario, the MinCut value of cutting between the two clusters is lower than cutting off parts of any one of the clusters as the cut still needs to separate s and t . This requirement of the two clusters being more intra-connected than interconnected is precisely how one would define a cluster, so for proper choice of vertices s and t one expects the *st*-MinCut partition to be “reasonable” (i.e. in that it separates two clusters). A visualization of this can be seen in [Figure 1](#) where one could consider the orange vertices to be good choices for s and t – this is elaborated upon in [Section 3.2](#).

A first version of our algorithm can be stated as follows:

As the computation of *st*-MinCut in line 3 of our **basic Xvst algorithm** is in $\mathcal{O}(nm)$ time, the complexity of this algorithm is $\mathcal{O}(n^3m)$ ([Theorem 3.2](#)). While this is not particularly fast, the design of the algorithm guarantees that the resulting partition is a cut that is reasonable in the sense that it separates two vertices s and t through the *st*-MinCut partition S_{st} while also taking cluster size into account via the balancing term in the graph cut value $\text{XC}_{S_{st}}(G)$. This imitates the nature of (the NP-complete problem of) computing graph cuts from a qualitative perspective, whereas techniques such as spectral clustering approximate the corresponding functionals by convex relaxations. More precisely, the partition S_{\min} that the **basic Xvst algorithm** outputs is guaranteed to be an *st*-MinCut for some $s, t \in V$, whereas the partition returned by spectral clustering does not possess any inherent qualitative feature per se.

Basic Xvst algorithm: XC(G) via st -MinCuts

input : weighted graph $G = (V, E, \mathbf{W})$
output: XC value c_{\min} with associated partition S_{\min}

```
1 set  $c_{\min} \leftarrow \infty$  and  $S_{\min} \leftarrow \emptyset$ 
2 for  $\{s, t\} \subset V$  with  $s \neq t$  do
3   compute an  $st$ -MinCut partition  $S_{st}$  on  $G$ 
4   compute the graph cut value  $\text{XC}_{S_{st}^c}(G)$  of partition  $S_{st}$ 
5   if  $\text{XC}_{S_{st}}(G) < c_{\min}$  then
6      $c_{\min} \leftarrow \text{XC}_{S_{st}}(G)$ 
7      $S_{\min} \leftarrow S_{st}$ 
8   end
9 end
```

In the literature, the attention has mainly focused on the set \mathcal{S}_{st}^* of st -MinCut partitions for a fixed pair of $s, t \in V$, $s \neq t$. For instance, the cardinality of \mathcal{S}_{st}^* has been used as a structure characterization on the crossing minimization problem in graph planarizations Chimani, Gutwenger, and Mutzel, 2007. Andersen and Lang (2008) proposed an algorithm subroutine to improve existing partitions that is based on st -MinCuts; their method, however, introduces artificial vertices s and t to act as penalization for changing the existing partition, in difference to our algorithm which considers only “real” vertices $s, t \in V^{\text{loc}}$. It should be noted that Bonsma (2010) showed that the problem of finding the most balanced partition in \mathcal{S}_{st}^* is NP-hard, so that we consider only one st -MinCut partition S_{st} for a given pair of $s, t \in V$, and employ instead the collection of such partitions for all pairs of $s, t \in V$, namely, $\{S_{st}: s, t \in V, s \neq t\}$. In this way, we preserve the intrinsic structure of the graph to a large extend while gaining efficient computation in polynomial time.

3.2 The proposed Xist algorithm

We improve the **basic Xvst algorithm** by two techniques, see Figure 1 for an illustration.

First, we further restrict the minimization of the graph cut functional by only considering certain vertices s and t to compute the st -MinCuts over. In practice, st -MinCut partitions only become viable if one forces s and t to belong to tightly connected clusters to avoid a partition that cuts out only one vertex (this may happen as MinCut does not have a balancing term to counteract this). If s and t are connected to their respective neighbouring

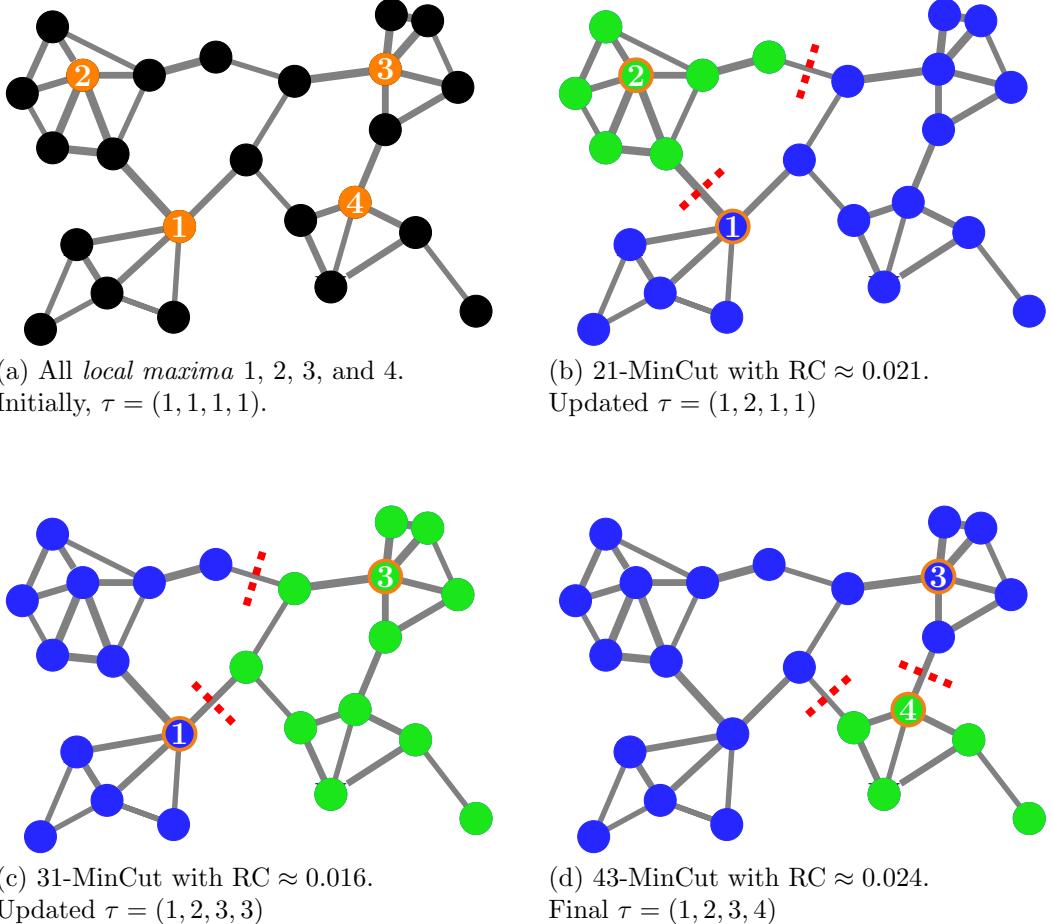


Figure 1: Illustration of the **Xist algorithm** for the Ratio Cut functional on a weighted toy graph, where edge thickness is proportional to edge weight. The vector τ in the **Xist algorithm** determines the vertices s and t for the next st -MinCut. (a) depicts the set of local maxima $V^{\text{loc}} = \{1, 2, 3, 4\}$. **Xist** computes first the 21-MinCut and updates τ in (b), then the 31-MinCut with another update to τ in (c). Finally, the 43-MinCut is computed, and τ is not updated further. Since the 31-MinCut in (c) gives the best Ratio Cut value among all three partitions, **Xist** outputs this partition and value.

nodes through high-weighted edges, the st -MinCut cannot simply cut out one or the other and is therefore forced to find a different, more balanced way to separate both vertices. Formally, we call a vertex $u \in V$ a *local maximum* if $\deg(u) \geq \deg(v)$ for all $v \in V$ with $\{u, v\} \in E$. We denote the set of local

maxima as

$$V^{\text{loc}} := \{u \in V \mid \deg(u) \geq \deg(v) \text{ for all } v \in V \text{ with } \{u, v\} \in E\},$$

with its cardinality $|V^{\text{loc}}| =: N$. For a visualization of V^{loc} see [Figure 1](#) (a). By definition of V^{loc} , we can ensure the scenario described above by requiring both s and t to be local maxima.

Second, it is not necessary to iterate over all pairs $\{s, t\} \subset V^{\text{loc}}$ to obtain all st -MinCuts of vertices in V^{loc} . It is known that in a graph of n vertices, there are at most $n - 1$ distinct st -MinCuts, and that these can be computed through the construction of the so-called *Gomory-Hu tree* that was introduced by Gomory and T. C. Hu, [1961](#). The Gomory-Hu tree is a tree built on V where the edge weights are st -MinCut values, $s, t \in V$. Gomory and T. C. Hu ([1961](#)) showed that this tree can be constructed through vertex contraction and only $n - 1$ st -MinCut computations, and that it encapsulates all st -MinCut values. Consequently, there are only $n - 1$ st -MinCuts, meaning that it is possible to improve the complexity of the [basic Xvst algorithm](#) by $\mathcal{O}(n)$. Additionally, their proofs can be adapted for the case that only those st -MinCuts are of interest where $s, t \in A$ for any subset $A \subseteq V$. We present this in [Appendix A.2](#).

Expanding upon this classical result, Gusfield ([1990](#)) showed that alternatively to the Gomory-Hu method of vertex contraction and tree construction, it is possible to compute all st -MinCuts directly on the (uncontracted) graph G . Consequently, Gusfield ([1990](#), Section 3.4) presented an adaptation of the Gomory-Hu algorithm that is simpler to implement and runs on the original graph only.

We incorporate the two techniques (restriction to local maxima and Gomory-Hu tree vertex selection) into the [basic Xvst algorithm](#) to obtain the final [Xist algorithm](#) (short for XC imitation through st-MinCuts; pronounced like “exist”).

Note that the set V^{loc} could be substituted by any subset of vertices $A \subseteq V$, and [Xist](#) would still output the best XCut among st -MinCuts for all pairs of vertices $s, t \in A$, $s \neq t$. There are several reasons for choosing the set V^{loc} specifically:

- (i) By only considering local maxima the st -MinCut is forced to separate s and t and therefore has to cut through the presumed “valley” (i.e. set of vertices with low degree) that lies between s and t .
- (ii) Vertices that are no local maxima are not likely to benefit from st -MinCuts. This is due to the lack of a balancing term as previously discussed; MinCut tends to cut out only one vertex, e.g. the vertex of a

Xist algorithm: XCut imitation through st -MinCuts on local maxima via implicit Gomory-Hu trees

```

input : weighted graph  $G = (V, E, \mathbf{W})$ 
output: XC value  $c_{\min}$  with associated partition  $S_{\min}$ 

1 set  $c_{\min} \leftarrow \infty$  and  $S_{\min} \leftarrow \emptyset$ 
2 determine the set of local maxima  $V^{\text{loc}} \subseteq V$ 
3 if  $|V^{\text{loc}}| = 1$  then terminate
4 set  $\tau \leftarrow (1, \dots, 1) \in \mathbb{R}^N$ 
5 for  $i \in \{2, \dots, N\}$  do
6   let  $s$  denote the  $i$ -th, and  $t$  the  $\tau_i$ -th vertex in  $V^{\text{loc}}$ 
7   compute an  $st$ -MinCut partition  $S_{st}$  on  $G$ 
8   // Note that by definition  $s \in S_{st}$  and  $t \in S_{st}^c$ 
9   compute the XCut value  $\text{XC}_{S_{st}}(G)$  of partition  $S_{st}$ 
10  if  $\text{XC}_{S_{st}}(G) < c_{\min}$  then
11     $c_{\min} \leftarrow \text{XC}_{S_{st}}(G)$ 
12     $S_{\min} \leftarrow S_{st}$ 
13  end
14  for  $j \in \{i, \dots, N\}$  do
15    let  $v_j$  be the  $j$ -th vertex in  $V^{\text{loc}}$ 
16    if  $v_j \in S_{st}$  and  $\tau_j = \tau_i$  then  $\tau_j \leftarrow i$ 
17  end
18 end

```

low degree compared to its neighbours. This will, however, not happen as often with vertices of high degree as the latter punishes one-vertex cuts, by yielding a (comparatively) large cut value. See [Figure 1](#), for example.

- (iii) As a subset of V , the set V^{loc} greatly reduces the number of st -MinCuts to compute. The precise extend of its influence is difficult to quantify and heavily depends on the graph itself. In practice, only considering local maxima can lead to a vastly improved runtime (cf. [Figure 3](#) later).

The number N of local maxima in a graph G depends heavily on the vertex degrees and edge weights. Clearly, N can be bounded from above by the independence number $\alpha(G)$ of the graph, several upper bounds of which are available in the literature ([Willis, 2011](#)). One example is the following upper bound O, Y. Shi, and Taoqiu, [2021](#), Theorem 3.2:

$$1 \leq N \leq \alpha(G) \leq \frac{\Delta(G)n}{\Delta(G) + \delta(G)},$$

where $\Delta(G)$ and $\delta(G)$ are the maximum and minimum number of neighbours of a vertex in G , respectively. This is not sharp in general, and is dominated by more sophisticated bounds, which, however, are more difficult to compute; in fact, the graph independence number $\alpha(G)$ is NP-hard to compute itself (Garey and Johnson, 1979).

3.3 Theoretical properties

One important advantage of incorporating the Gomory-Hu method is that **Xist** is guaranteed to optimize the cut value over $N - 1$ *distinct* partitions, thus removing redundant computations. This reduces runtime significantly (cf. Figure 3 later).

Assumption 1. There exist $s, t \in V$, $s \neq t$, with a unique st -MinCut partition S_{st} such that

$$\text{XC}_{S_{st}}(G) = \min_{u,v,S_{u,v}} \text{XC}_{S_{uv}}(G),$$

where the minimum is taken over all $u, v \in V$, $u \neq v$, and *all* partitions S_{uv} that attain the respective uv -MinCuts.

Assumption 1 is the technical condition necessary to guarantee that the **Xvst** algorithm and in particular **Xist** yield a consistent output regardless of the procedure chosen to compute the st -MinCut. The main issue is uniqueness of the underlying st -MinCuts: There could be two distinct partitions that both attain the st -MinCut, but yield a different XCut value. Even if we were not to rely on an oracle to compute the st -MinCut partition, problems could still arise as no polynomial algorithm can compute all st -MinCut partitions for fixed $s, t \in V$, $s \neq t$ (Bonsma, 2010), so it is not possible to efficiently determine the XCut minimizing among all st -MinCut attaining partitions, and it is often not clear what partition a given st -MinCut algorithm will output, given the existence of two partitions with the same MinCut value, but different XCut values.

Consequently, from a technical standpoint, a version of **Assumption 1** is necessary for any algorithm that uses st -MinCut partitions and not just the cut value itself. The condition itself is fairly weak, especially in practice. For instance, if all st -MinCut partitions are unique (i.e. for any $s, t \in V$, $s \neq t$), **Assumption 1** is satisfied, so global uniqueness would be a stronger restriction. In practice, this condition will almost always be satisfied, especially for image data such as the examples in Section 4, or more generally for graphs with “suitably different” weights.

Using **Assumption 1**, we can more clearly characterize the output of **Xist**.

Theorem 3.1. *Xist* outputs $\min_{s,t \in V^{\text{loc}}} \text{XC}_{S_{st}}(G)$. Further, if **Assumption 1** holds with $s, t \in V^{\text{loc}}$, *Xist* and the **basic Xvst algorithm** yield the same output. If additionally the optimal XCut partition constitutes the st -MinCut for some $s, t \in V^{\text{loc}}$, *Xist* outputs the optimal XCut partition.

Proof. First, by design of *Xist algorithm*, it computes the minimal XCut among *some* st -MinCut partitions for *all* pairs $s, t \in V^{\text{loc}}, s \neq t$, the algorithm iterates over. This fact is shown in the [Appendix A.2](#), specifically [Theorem A.5](#). This, however, already shows the first claim as *Xist* considers st -MinCut partitions S_{st} for all $s, t \in V^{\text{loc}}$ and, by design, selects the one with the best XCut value among them.

As for the second part of the statement, it is clear that the **basic Xvst algorithm** computes $\min_{s,t \in V} \text{XC}_{S_{st}}(G)$. Here, it is important to stress that the partition S_{st} might not be the unique st -MinCut attaining partition. Indeed, as we treat each st -MinCut computation as an oracle call, it is not even clear whether in each algorithm, computing the st -MinCut consistently returns the same partition. Under **Assumption 1**, however, there exist $s, t \in V$ (even $s, t \in V^{\text{loc}}$ by assumption from the theorem statement) such that the st -MinCut partition S_{st} is unique and it attains the best possible XCut among *all possible* uv -MinCut partitions, for all $u, v \in V$. [Theorem A.5](#) guarantees that this st -MinCut is considered by *Xist* (and obviously also by the **basic Xvst algorithm**) and, because the minimum S_{st} is unique and attains the best XCut value, both algorithms yield S_{st} as their output.

The last assertion of the theorem now follows immediately. \square

[Theorem 3.1](#) shows that the **basic Xvst algorithm** and *Xist* are equivalent up to restriction to the subset V^{loc} , and in particular that it suffices to consider $N - 1$ pairs of st -MinCut partitions to obtain all possible such cuts between pairs $s, t \in V^{\text{loc}}, s \neq t$. In particular, this improves the worst-case complexity of *Xist* (compared to the **basic Xvst algorithm**) by one order of magnitude. We further obtain an approximation guarantee, i.e. that under the assumption that the optimal XCut partition (i.e. the partition attaining the minimum $\min_{S \subseteq V} \text{XC}(S)$) is an st -MinCut for some $s, t \in V^{\text{loc}}$, then *Xist* outputs this partition.

The following [Theorem 3.2](#) shows that use of both the restriction to local maxima and the use of the Gomory-Hu method improves the runtime of *Xist* significantly when compared to the **basic Xvst algorithm**.

Theorem 3.2. Assume that for any fixed partition S , the evaluation of $\text{XC}_S(G)$ takes $\mathcal{O}(\kappa)$ time, with $\kappa := \kappa(m, n)$ depending on m and n . Then, the computational complexity of the **basic Xvst algorithm** is $\mathcal{O}(n^2 \max\{nm, \kappa\})$, and that of *Xist* is $\mathcal{O}(N \max\{nm, \kappa\})$.

Note that evaluation of the functionals of popular cuts such as MinCut, Ratio Cut, NCut or Cheeger Cut are computed using only edge weights, meaning that this step is actually only $\mathcal{O}(m)$, except for Ratio Cut, which also requires determining $|S_{st}|$, thus yielding $\mathcal{O}(m + n)$ instead. Thus, for such cuts, the **basic Xvst algorithm** admits a computational complexity of $\mathcal{O}(n^3m)$, while **Xist** is $\mathcal{O}(Nnm)$ for general graphs. Recall that spectral clustering has a worst-case complexity of $\mathcal{O}(n^3)$. Thus, if $Nm = \mathcal{O}(n^2)$, the **Xist algorithm** is at least as fast as spectral clustering. If further $m = \mathcal{O}(n)$ and $N \ll n$, **Xist** can be much faster; this is often the case for graphs of bounded degrees, e.g. for an image, where its pixels constitute a regular grid. Note, however, that in the least favorable case of $N \asymp n$ and $m \asymp n^2$, **Xist** can be one order slower than spectral clustering.

3.4 Software implementation

We have implemented the **basic Xvst algorithm**, the **Xist algorithm** and **Multi-Xist** in Python (also in R, relatively slower). Our implementations can be found on GitHub (Suchan, 2023), to allow for the results in the following Section 4 to be reproduced. We are currently in the process of implementing our algorithms in C++, from which we expect a significant increase in computational efficiency.

4 Simulations and applications

The following simulations were performed on a laptop with Windows 11 operating system, a 2.70 GHz Intel® Core™ i5-12600H processor and 16 GB of RAM. The code necessary to reproduce the following (data) analysis can be found on GitHub (Suchan, 2023).

4.1 Approximation of multiway cuts

We start with an extension of **Xist** to compute multiway cuts, and examine its performance on a real-world dataset. Given a graph G and a number of desired partitions k , this extension **Multi-Xist** is obtained through a greedy approach: First, apply **Xist** to G , receiving partitions $T_{1,1}$ and $T_{1,2} := V \setminus T_{1,1}$, then restrict G to $T_{1,1}$ (and $T_{1,2}$) to obtain $G_{1,1}$ (and $G_{1,2}$). Then cut both restricted graphs again using **Xist**, and select the partition ($T_{1,1}$, say) that yields the lowest (“normalized”; see below) **Xist** value. Defining $T_{2,1}$ and $T_{2,2} := T_{1,1} \setminus T_{2,1}$, at this point G is considered to be divided into three partitions: $T_{1,2}$, $T_{2,1}$ and $T_{2,2}$. The pattern continues: The graph is further

restricted to $T_{2,1}$ and $T_{2,2}$, and again cut using **Xist**, whereupon the lowest cut value among those and $T_{1,2}$ is selected. This iterative cutting, restricting and selecting is continued until k partitions have been computed.

It should be noted that it is necessary to “normalize” the computed **Xist** cut values in order to ensure comparability across graphs of (possibly) vastly different sizes (in terms of n , m and \mathbf{W}). More specifically, cutting a graph whose weights have been scaled up by a constant factor should not change its normalized XCut. Hence, the balanced graph cuts are normalized by multiplying with an additional factor of $\sum_{i,j} w_{ij} / \text{Bal}(V, V)$:

$$\begin{aligned}\overline{\text{XC}}_S(G) &:= \sum_{i \in S, j \in S^c} \frac{w_{ij}}{\text{Bal}(S, S^c)} \frac{\text{Bal}(V, V)}{\sum_{i,j \in V} w_{ij}}, \\ \text{s.t. } \overline{\text{XC}}(G) &:= \min_{S \subset V} \overline{\text{XC}}_S(G).\end{aligned}$$

Clearly, a partition S minimizes $\overline{\text{XC}}(G)$ if and only if it minimizes $\text{XC}(G)$ (regardless of whether this minimization is done over all partitions or only over a subset), so this normalization does not impact the output of **Xist** (up to the aforementioned normalizing factor of $\sum_{i,j} w_{ij} / \text{Bal}(V, V)$).

One could also modify the **Xist** algorithm to approximate multiway graph cut kXC directly by computing $\{s_1, \dots, s_k\}$ -MinCuts for given nodes $s_1, \dots, s_k \in V$. This problem is more complex than computing kMC(G) of the entire graph G , even being NP-complete for general graphs, although polynomial algorithms exist if G is planar and k is fixed, see Dahlhaus et al., 1994. In contrast, our choice of iterative cutting has the advantage that it is computable in polynomial time for any weighted graph, more precisely, in $\mathcal{O}(kNnm)$ time. Moreover, **Multi-Xist** has a built-in “optimality guarantee” in that in each iteration the best current XC imitation is selected. This, of course, comes with all advantages and disadvantages that such a greedy approach entails.

We now demonstrate the application of **Multi-Xist** to a real-world example of detecting cell clusters. The data in question consists of images of microtubules in PFA-fixed NIH 3T3 mouse embryonic fibroblasts (DSMZ: ACC59) labeled with a mouse anti-alpha-tubulin monoclonal IgG1 antibody (Thermofisher A11126, primary antibody) and visualized by a blue-fluorescent Alexa Fluor® 405 goat anti-mouse IgG antibody (Thermofisher A-31553, secondary antibody). Acquisition of the images was performed using a confocal microscope (Olympus IX81). The images were kindly provided by Ulrike Rölleke and Sarah Köster (University of Göttingen), and are available on GitHub (Rölleke and Köster, 2020). We take on the task of identifying the $k = 8$ main clusters of cells. To reduce computation time, we construct the

Multi-Xist algorithm: Multiway Xist with XCut-minimizing cluster selection

Input : weighted graph $G = (V, E, \mathbf{W})$, number $k \in \mathbb{N}_{\geq 2}$ of desired partitions

Output: k -way partition R

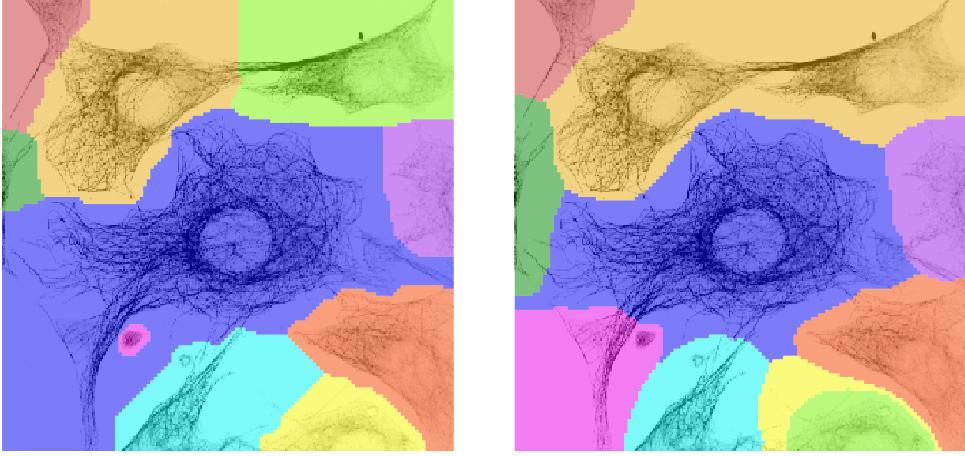
```

1  $S \leftarrow ((\emptyset, V), (\emptyset, \emptyset), \dots, (\emptyset, \emptyset)) \in \{(T_1, T_2) \mid T_1, T_2 \subseteq V\}^k$ 
2  $c \leftarrow (0, \infty, \dots, \infty) \in (-\infty, \infty]^k$ 
3 for  $i = 2, \dots, k$  do
4   set  $j_{\min} \leftarrow \arg \min_{\ell=1, \dots, k} c_j$  and  $(T_{i,1}, T_{i,2}) \leftarrow S_{j_{\min}}$ 
5   for  $\ell = 1, 2$  do
6     if  $|T_{i,\ell}| \leq 1$  then
7       set  $r_{i,\ell} \leftarrow \infty$  and  $R_{i,\ell} \leftarrow \emptyset$ 
8     else
9       set  $G_{i,\ell} \leftarrow (T_{i,\ell}, E \cap (T_{i,\ell} \times T_{i,\ell}), \mathbf{W}|_{T_{i,\ell}})$ , where  $\mathbf{W}|_{T_{i,\ell}}$  is
          the weight matrix of  $G$  restricted to the vertices in  $T_{i,\ell}$ 
          only
10      compute  $r_{i,\ell} \leftarrow \text{Xist}(G_{i,\ell})$  with corresponding partition
           $R_{i,\ell}$  using Xist
11       $r'_{i,\ell} \leftarrow r_{i,\ell} \cdot \text{Bal}_{G_{i,\ell}}(T_{i,\ell}, T_{i,\ell}) / (\sum_{i,j \in V} w_{ij})$ 
12    end
13  end
14  set  $c_{j_{\min}} \leftarrow r'_{i,1}$  and  $c_i \leftarrow r'_{i,2}$ 
15  set  $S_{j_{\min}} \leftarrow (R_{i,1}, T_{i,1} \setminus R_{i,1})$  and  $S_i \leftarrow (R_{i,2}, T_{i,2} \setminus R_{i,2})$ 
16 end
17  $R \leftarrow \{S_{i,1} \cup S_{i,2}\}_{i=1, \dots, k}$ 

```

graph by down-sampling the original cell image on a coarse regular grid of size $r \times r$, here for $r = 128$. While the partition computed using this slight discretization does not have the same resolution of the original image (which is 512×512), the grid size parameter r can be chosen as large or as small as required. Edges were assigned by connecting each grid point to its eight direct neighbours, with weights defined as the product of the grey-color intensity values of connected vertices. In contrast to the usual **Xist algorithm**, instead of line 2 we defined the set of local maxima to be pixels whose image (grey) value (instead of their degree) is larger than that of all neighbouring pixels. We make this slight modification because in images, this method better encompasses and expresses the concept of *local maxima*. The results are displayed in [Figure 2](#).

We see that **Multi-Xist** does indeed yield a sensible partition, and it is



(a) **Multi-Xist**

(b) Spectral clustering

Figure 2: Image segmentation of microtubules in NIH 3T3 cells. The colors visualize the resulting $k = 10$ partitions via **Multi-Xist** and spectral clustering, respectively. The underlying cell clusters are visible in black, where a darker color marks denser microtubule network.

also able to separate neighbouring cell clusters of different scales, correctly separating smaller, more disconnected cell clusters from the rest. This is in contrast to spectral clustering which cuts right through the main cell cluster, separating it into a red and blue part. This example exemplifies also the scalability issues spectral clustering faces when dealing with clusters of different scales Nadler and Galun, 2007. A visualization of the full **Multi-Xist** process on this image, in particular the selection of which cluster to cut further, is shown in Figure 2, where we let $k = 2, \dots, 9$. This example demonstrates even more clearly the multiscale advantage **Xist** has over spectral clustering can be found in Figure 5 in Appendix A.1.

4.2 Empirical runtime comparison

In the following sections we compare **Xist** to five state-of-the-art graph partitioning algorithms, both in terms of partition quality and runtime. The algorithms and implementation being compared are the following:

- Our **Xist** algorithm as implemented in Python (see Suchan, 2023).
- The Leiden algorithm introduced by V. A. Traag, Waltman, and Eck, 2019 and implemented in the `leidenalg` Python package (V. Traag et al., 2023).

- The Python-support of the KaHIP (Karlsruhe High Quality Partitioning) algorithm package (Sanders and Schulz, 2013), where we used the “strong social” mode of their main algorithm to achieve the highest partitioning quality.
- The main algorithm of the METIS graph partitioning software (Karypis and Kumar, 1998) as wrapped in Python via the PyMetis package (Kloeckner et al., 2022).
- The Chaco algorithm (Hendrickson and Leland, 1995) as a standalone software package obtained from GitHub (Burgess and Devine, 2023), called through a shell script executed from within Python in order to use and record its output.
- The classical spectral clustering approach originally proposed by J. Shi and Malik, 2000 and realized through the scikit-learn Python package (Pedregosa et al., 2011), where the eigenvector entries are clustered again using k -means clustering, see Luxburg, 2007, Section 4.

All of the above algorithms were used with their default parameters to ensure comparability, with the exception of the Leiden algorithm because it requires the additional input of a “resolution parameter” with no given default option. Hence, we considered the Leiden algorithm as an oracle (denoted as “Leiden (oracle)”), meaning so that the tuning parameter is chosen as to optimize the quality measure in question, i.e. to minimize the NCut value or maximize the classification rate (see Figure 4). Such choices of parameters require the knowledge of the true cluster assignments, thus referred to as *oracle parameter* choices. Only when the cut quality is not evaluated, i.e. for runtime comparisons (see Figure 3), we use the Leiden algorithm in its usual form (denoted as “Leiden”) with an arbitrary choice of resolution parameter.

Further note that the Leiden algorithm returns a partition of the graph into k sets of vertices, where k is heavily dependent upon the “resolution parameter”, so that, to evaluate the quality of the Leiden algorithm fairly, it is necessary to generalize the XCut term XC to a k -fold partition of G . This generalization is well-established in the literature, and it is given by

$$\text{XC}_T(G) := \frac{1}{2} \sum_{i=1}^k \text{XC}_{T_i}(G) \quad \text{for } T = \{T_1, \dots, T_k\} \subset V^k \text{ with } \sum_{i=1}^k T_i = V \tag{4.1}$$

(and all T_i are pairwise disjoint). Compared to the original definition of $\text{XC}_S(G)$, where $S \subset V$, it is evident that $\text{XC}_S(G) = \text{XC}_{\{S, S^c\}}(G)$.

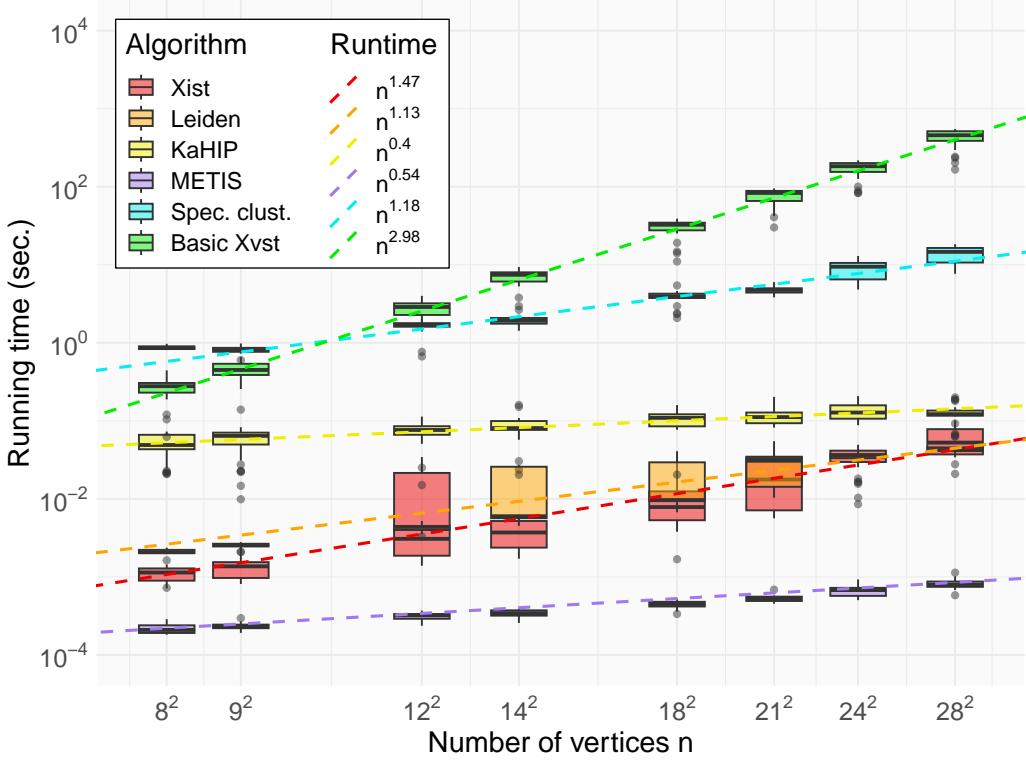


Figure 3: Comparison of the empirical runtimes (in seconds) of **Xist** (red), Leiden in its non-oracle form (orange), KaHIP (yellow), METIS (violet), spectral clustering (cyan), and the **basic Xvst algorithm** (green) on a log-log scale against the number $n = r^2$ of vertices, with dashed lines indicating the empirical complexity of the respective algorithms. The boxplots were obtained by applying each algorithm to a dataset of 21 NIH3T3 cell cluster images of size 504×504 pixels each after discretizing it onto a $r \times r$ grid, for $r \in \{8, 9, 12, 14, 18, 21, 24, 28\}$.

We compare the runtime of the basic algorithm and the proposed Xist algorithm with that of the above algorithms on the same cell image dataset as in Section 4.1. We crop the images to 504×504 and vary the “resolution” of the grid (i.e. the grid size parameter r) and examine the rates at which the runtime increases, see Figure 3. Note that we did not include the Chaco algorithm in this runtime comparison as it does not have a Python implementation (or a Python wrapper), so it had to be run as a standalone program, rendering time comparisons meaningless. To still give an intuition in terms of absolute time, Chaco’s runtime was slightly above the times of KaHIP in Figure 3.

As is shown, the proposed **Xist** algorithm is empirically roughly 1.5 orders of magnitude faster than the **basic Xvst algorithm**. Moreover, in terms of absolute time it is much faster than even very efficient algorithms such as KaHIP or spectral clustering which are implemented in C, even though **Xist** is only implemented in Python and the fact that for the computation of the *st*-MinCuts, a theoretically suboptimal algorithm had to be used as – to the best of our knowledge – there does not yet exist any implementation of the current fastest ones, namely Orlin, 2013 and Orlin and Gong, 2021. We are currently in the process of implementing **Xist** efficiently (i.e. in C++). Finally, note that the near-constant time of the Chaco algorithm in [Figure 3](#) should be considered with an appropriate amount of scepticism since it is the only algorithm without a Python implementation and thus needs to be called via a shell script as described above.

4.3 Qualitative assessment of **Xist**

Since the aim of our algorithm is to imitate graph cuts (while still being computable in polynomial time), we evaluate the quality of our algorithm in a partitioning exercise where the task is to determine which points from a two-cluster sample stem from which cluster. To this end let $\delta > 0$ and consider a random mixture of Gaussians:

$$\mathbf{X} := (X_1, \dots, X_n) \sim BZ_0 + (1 - B)Z_\delta, \quad (4.2)$$

where $B \sim \text{Ber}(1/2)$ a Bernoulli random variable, and $Z_\delta \sim \mathcal{N}_2(\boldsymbol{\delta}, \mathbf{I}_2)$ the two-dimensional standard Gaussian around $\boldsymbol{\delta} = (\delta, \delta) \in \mathbb{R}^2$. For details on the graph construction see [Figure 4](#).

We compare the partition quality of **Xist** to that of the other algorithms (see [Section 4.2](#)). Note that Löffler, Zhang, and Zhou (2021) have shown that in this scenario (i.e. random mixture of Gaussians), spectral clustering is asymptotically minimax optimal in terms of the classification rate (i.e. the ratio of observations X_i correctly classified as belonging to their respective Gaussian in the mixture), so this scenario is highly favourable towards spectral clustering. Despite this, **Xist** is able to achieve a similar accuracy to the oracle version of the Leiden algorithm, being only slightly worse than spectral clustering and much better than the other state-of-the-art algorithms as [Figure 4](#) (a) shows, where the classification rate is plotted over the intercluster distance δ . Note that, as stated before, Leiden was realized as an oracle, i.e. it outputs the highest classification rate in (a) and lowest NCut value in (b) over a range of its so-called “resolution parameter”. Likely due to this realization as an oracle it performs almost as good as spectral clustering,

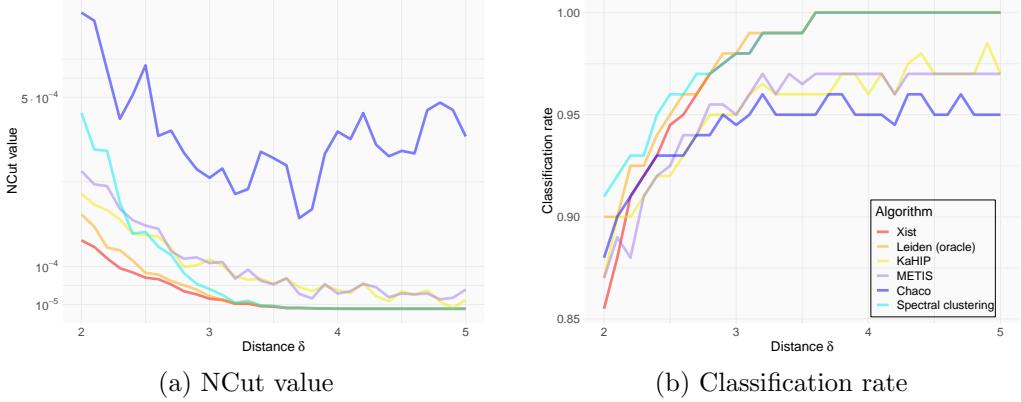


Figure 4: Comparison of the classification rate and NCut value of **Xist** (red), the oracle version of the Leiden algorithm (orange), KaHIP (yellow), METIS (violet), Chaco (blue), spectral clustering (cyan), and the **basic Xvst algorithm** (green) over the intercluster distance δ . The graph being partitioned is built from $n = 100$ samples from the Gaussian mixture distribution in (4.2) using a combination of 5-nearest and 0.2-neighbourhood, i.e. by defining $\{u, v\} \in E$ if and only if $u \in \text{NN}_5(v)$ or $v \in \text{NN}_5(u)$ or $\|u - v\| \leq 0.2$, where $\text{NN}_5(u)$ denotes the set of five nearest (in terms of the euclidean norm $\|\cdot\|$ on \mathbb{R}^2) neighbours in V of $u \in V$. The weights are defined as $w_{uv} := \exp(-\|u - v\|/0.2)$. These choices are made to ensure a connected graph (5-nearest neighbours) where clusters are more easily detectable (0.2-neighbourhood) and the spatial structure is pronounced (exponential weights). The curves depicted were obtained by taking the mean classification rate and mean NCut value over 100 iterations of the above procedure, re-generating \mathbf{X} each time.

but only marginally better than **Xist**, and only for small δ . **Xist** consistently outperforms the other state-of-the-art algorithms. The difference is most striking in Figure 4 (b), where **Xist** yields a consistently better NCut value than all the other algorithms, thus achieving its set goal of approximating graph cuts better than spectral clustering in particular.

This simulation study additionally refutes the notion that graph-based methods (such as KaHIP, METIS or Chaco) can not achieve a good performance in the case of a Gaussian mixture *by design* because they introduce an additional abstraction (the graph). The success of **Xist** in this scenario shows that the poor performance of other graph-based methods is not inherent to the graph structure, but likely the result of computational artifacts. Indeed, as Figure 4 demonstrates, if one is able to compute graph cuts highly

accurately, as good as or even better results than spectral clustering can be achieved.

4.4 Clustering large network datasets

To demonstrate the applicability and versatility of [Xist](#), even in its current Python implementation, we take on the task of clustering large network datasets. We consider the “Stanford Large Dataset Collection” (SNAP, see [Leskovec and Krevl, 2014](#)) which contains many real-world network graphs of varying types and sizes. Of those graphs we consider several undirected graphs, all unweighted, in particular

- the arXiv High-Energy Physics – Phenomenology (HEP-PH) dataset ([Leskovec, Kleinberg, and Faloutsos, 2005](#); [Gehrke, Ginsparg, and Kleinberg, 2003](#)) where authors represent nodes and edges papers co-authored between them;
- parts of the Multi-Scale Attributed Node Embedding (MUSAE) dataset ([Rozemberczki, Allen, and Sarkar, 2021](#)), namely
 - the MUSAE Facebook subset, where nodes are verified Facebook pages and edges mutual links between them;
 - the MUSAE Squirrel subset, where nodes are squirrel-related Wikipedia pages and edges mutual links between them;
- the Artist subset of the GEMSEC Facebook dataset ([Rozemberczki, Davies, et al., 2019](#)), consisting of nodes representing verified Facebook pages categorized as “artist” and edges mutual likes between them;
- and the Enron email dataset ([Klimt and Yang, 2004](#)), containing Enron email addresses as nodes and edges between nodes if an email was sent from one node to the other;

In all datasets, node features were ignored as the algorithms in question are not designed to account for such information. Also note that the selection of subsets of these datasets was essentially arbitrary; similar results hold for other subsets (e.g. the MUSAE crocodile subset).

On the above datasets, we compare the performance of [Xist](#) and the KaHIP, METIS and Chaco algorithms, both in terms of the Normalized Cut value of the partition as well as the algorithm runtime. The result can be found in [Table 2](#). Note that spectral clustering cannot be reasonably applied here since it requires computation of the eigenvalues of the graph Laplacian,

an $n \times n$ matrix, a task that is very memory-intensive. We also did not apply the Leiden algorithm here since the search for an NCut-minimizing “resolution parameter” would be very time-consuming for datasets as large as the above, and even then reporting the time accurately would be difficult since the runtime (and output) of any such oracle heavily depends on the range the “resolution parameter” is chosen from.

Table 2: Normalized Cut value and empirical runtime of **Xist** as well as KaHIP, METIS and Chaco for various datasets. Here, n and m are the number of vertices and edges of only the largest connected component of the respective dataset.

Dataset	n	m	NCut value ($\cdot 10^{-8}$)			
			Xist	KaHIP	METIS	Chaco
MUSAE Squirrel	5201	198353	26.08	45.76	67.33	43.98
arXiv HepPh	11204	117619	1.01	45.73	51.13	55.74
MUSAE Facebook	22470	170823	3.01	17.23	18.66	20.91
Enron Email	33696	180811	1.25	67.70	53.06	66.91
GEMSEC Artist	50515	819090	4.70	13.56	13.87	14.98

Dataset	n	m	Time (seconds)			
			Xist	KaHIP	METIS	Chaco
MUSAE Squirrel	5201	198353	0.38	1.26	0.04	0.09
arXiv HepPh	11204	117619	0.91	0.76	0.03	0.04
MUSAE Facebook	22470	170823	8.95	1.10	0.05	0.07
Enron Email	33696	180811	4.67	3.86	0.06	0.36
GEMSEC Artist	50515	819090	10.71	20.44	0.27	0.32

Note that we restricted the original datasets to only their largest respective components since, as previously mentioned, **Xist** would immediately separate unconnected components and return an XCut value of 0 (for any XCut balancing term). It is evident from [Table 2](#) that **Xist** is able to handle large datasets, taking a moderate amount of time to cluster all the datasets without being implemented in a lower-level programming language such as C or C++ – unlike METIS, KaHIP or Chaco which have efficient implementations already (see [Section 4.2](#)). Despite this, **Xist** has a better runtime scaling than KaHIP, and we expect a C++ implementation of it to be competitive in terms of runtime.

Qualitatively, one can see that **Xist** decisively outperforms all three state-

of-the-art algorithms, showing that it achieves its goal of approximating balanced graph cuts (here: normalized cut) better than the its competitors.

5 Conclusion and discussion

In this article we have proposed a novel algorithm $Xist$ for balanced graph cuts, which preserves the qualitative (multiscale) feature of the original graph cut and meanwhile allows fast computation even for large scale datasets, in particular for sparse graphs. This is achieved by combining the combinatorial nature of st -MinCuts with the acceleration techniques based on restriction to vertices of locally maximal degrees and vertex merging.

We have demonstrated the applicability and versatility of our algorithm on (cell) image segmentation, simulated data as well as large network datasets. The desirable performance of the proposed $Xist$, also seen from our simulations, benefits greatly from the structure of V^{loc} , induced by locally maximizing the degree, and its ordering, which respects the intrinsic geometric structure of the data, to a large extent. We stress, however, that the theoretical findings of [Theorem 3.1](#), in particular its guarantee to consider $|V^{\text{loc}}| - 1$ distinct partitions, and [Theorem 3.2](#) are applicable to any subset of V . This means that $Xist$ can be easily adapted to other interesting subsets of vertices, making the algorithm even more flexible.

Acknowledgments

LS is supported by the DFG (German Research Foundation) under project GRK 2088: “Discovering Structure in Complex Data”, subproject A1. HL is funded and AM is supported by the DFG under Germany’s Excellence Strategy, project EXC 2067: “Multiscale Bioimaging: from Molecular Machines to Networks of Excitable Cells” (MBExC). AM and HL are supported by DFG CRC 1456 “Mathematics of Experiment”, and AM is supported by DFG RU 5381 “Mathematical Statistics in the Information Age – Statistical Efficiency and Computational Tractability”, subproject “Sublinear time methods with statistical guarantees”. The authors would like to especially thank Max Wardetzky (University of Göttingen) for helpful discussions, Ulrike Rölleke and Sarah Köster (University of Göttingen) for providing the cell image dataset, and Florin Manea (University of Göttingen) for pointing out some references.

References

- Andersen, Reid and Kevin J. Lang (2008). “An algorithm for improving graph partitions”. In: *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’08. San Francisco, California: Society for Industrial and Applied Mathematics, pp. 651–660.
- Arora, Sanjeev and Satyen Kale (May 2016). “A Combinatorial, Primal-Dual Approach to Semidefinite Programs”. In: *Journal of the ACM* 63.2. ISSN: 0004-5411. DOI: [10.1145/2837020](https://doi.org/10.1145/2837020).
- Bonsma, Paul (2010). “Most balanced minimum cuts”. In: *Discrete Applied Mathematics. The Journal of Combinatorial Algorithms, Informatics and Computational Sciences* 158.4, pp. 261–276. DOI: [10.1016/j.dam.2009.09.010](https://doi.org/10.1016/j.dam.2009.09.010).
- Bui, Thang Nguyen and Curt Jones (1992). “Finding good approximate vertex and edge partitions is NP-hard”. In: *Information Processing Letters* 42.3, pp. 153–159. ISSN: 0020-0190. DOI: [10.1016/0020-0190\(92\)90140-Q](https://doi.org/10.1016/0020-0190(92)90140-Q).
- Burgess, Wade and K Devine (2023). *Chaco*. Sandia National Laboratories. URL: <https://github.com/sandialabs/Chaco>.
- Cheeger, Jeff (1971). “A lower bound for the smallest eigenvalue of the laplacian”. In: *Problems in Analysis*. Princeton: Princeton University Press, pp. 195–200. ISBN: 9781400869312. DOI: [doi:10.1515/9781400869312-013](https://doi.org/10.1515/9781400869312-013).
- Chen, Wen-Yen et al. (2011). “Parallel Spectral Clustering in Distributed Systems”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.3, pp. 568–586. DOI: [10.1109/TPAMI.2010.88](https://doi.org/10.1109/TPAMI.2010.88).
- Chew, Selene E. and Nathan D. Cahill (2015). “Semi-Supervised Normalized Cuts for Image Segmentation”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1716–1723. DOI: [10.1109/ICCV.2015.200](https://doi.org/10.1109/ICCV.2015.200).
- Chimani, Markus, Carsten Gutwenger, and Petra Mutzel (2007). “On the minimum cut of planarizations”. In: *6th Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications*. Vol. 28. Electron. Notes Discrete Math. Elsevier Sci. B. V., Amsterdam, pp. 177–184. DOI: [10.1016/j.endm.2007.01.036](https://doi.org/10.1016/j.endm.2007.01.036).
- Cook, William et al. (1998). “Minimum cuts in undirected graphs”. In: *Combinatorial Optimization*. John Wiley & Sons. Chap. 3.5, pp. 71–84. ISBN: 9781118033142. DOI: [10.1002/9781118033142.ch3](https://doi.org/10.1002/9781118033142.ch3).
- Dahlhaus, Elias et al. (1994). “The complexity of multiterminal cuts”. In: *SIAM Journal on Computing* 23.4, pp. 864–894. DOI: [10.1137/S0097539792225297](https://doi.org/10.1137/S0097539792225297).

- García Trillos, Nicolás and Dejan Slepčev (2018). “A variational approach to the consistency of spectral clustering”. In: *Applied and Computational Harmonic Analysis* 45.2, pp. 239–281. ISSN: 1063-5203. DOI: [10.1016/j.acha.2016.09.003](https://doi.org/10.1016/j.acha.2016.09.003).
- Garey, Michael R. and David S. Johnson (1979). *Computers and Intractability. A guide to the theory of NP-completeness*. A Series of Books in the Mathematical Sciences. W.H. Freeman and Company.
- Gawrychowski Paweł and Mozes, Shay and Oren Weimann (2024). “Minimum Cut in $O(m \log^2 n)$ Time”. In: *Theory of Computing Systems* 68.4, pp. 814–834. ISSN: 1432-4350. DOI: [10.1007/s00224-024-10179-7](https://doi.org/10.1007/s00224-024-10179-7).
- Gehrke, Johannes, Paul Ginsparg, and Jon Kleinberg (Dec. 2003). “Overview of the 2003 KDD Cup”. In: *ACM SIGKDD Explorations Newsletter* 5.2, pp. 149–151. ISSN: 1931-0145. DOI: [10.1145/980972.980992](https://doi.org/10.1145/980972.980992).
- Gomory, R. E. and T. C. Hu (1961). “Multi-terminal network flows”. In: *Journal of the Society for Industrial and Applied Mathematics* 9, pp. 551–570. ISSN: 0368-4245.
- Guattery, Stephen and Gary L. Miller (July 1998). “On the quality of spectral separators”. In: *SIAM Journal on Matrix Analysis and Applications* 19.3, pp. 701–719.
- Gusfield, Dan (1990). “Very simple methods for all pairs network flow analysis”. In: *SIAM Journal on Computing* 19.1, pp. 143–155. ISSN: 0097-5397. DOI: [10.1137/0219009](https://doi.org/10.1137/0219009).
- Hagen, Lars and Andrew B. Kahng (1992). “A new approach to effective circuit clustering”. In: *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design*. ICCAD ’92. IEEE Computer Society Press, pp. 422–427. ISBN: 0897915402.
- Hendrickson, Bruce and Robert Leland (1995). “A multilevel algorithm for partitioning graphs”. In: *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*. Supercomputing ’95. San Diego, California, USA: Association for Computing Machinery, 28–es. ISBN: 0897918169. DOI: [10.1145/224170.224228](https://doi.org/10.1145/224170.224228).
- Heuvel, Martijn van den, Rene Mandl, and Hilleke Hulshoff Pol (Apr. 2008). “Normalized cut group clustering of resting-state FMRI data”. In: *PloS One* 3.4, e2001.
- Karypis, George and Vipin Kumar (1998). “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. In: *SIAM Journal on Scientific Computing* 20.1, pp. 359–392. DOI: [10.1137/S1064827595287997](https://doi.org/10.1137/S1064827595287997).
- Khandekar, Rohit, Satish Rao, and Umesh Vazirani (July 2009). “Graph partitioning using single commodity flows”. In: *Journal of the ACM* 56.4. ISSN: 0004-5411. DOI: [10.1145/1538902.1538903](https://doi.org/10.1145/1538902.1538903).

- King, Valerie, Satish B. Rao, and Robert Endre Tarjan (1994). “A faster deterministic maximum flow algorithm”. In: *Journal of Algorithms. Cognition, Informatics and Logic* 17.3. Third Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL, 1992), pp. 447–474. ISSN: 0196-6774. DOI: [10.1006/jagm.1994.1044](https://doi.org/10.1006/jagm.1994.1044).
- Klimt, Bryan and Yiming Yang (2004). *Introducing the Enron Corpus*. URL: <https://www.ceas.cc/papers-2004/168.pdf>.
- Kloeckner, Andreas et al. (July 2022). *PyMetis*. Version 2022.1. DOI: [10.5281/zenodo.6892213](https://doi.org/10.5281/zenodo.6892213). URL: <https://github.com/inducer/pymetis>.
- Leskovec, Jure, Jon Kleinberg, and Christos Faloutsos (2005). “Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations”. In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. KDD ’05. Chicago, Illinois, USA: Association for Computing Machinery, pp. 177–187. ISBN: 159593135X. DOI: [10.1145/1081870.1081893](https://doi.org/10.1145/1081870.1081893).
- Leskovec, Jure and Andrej Krevl (June 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. URL: <https://snap.stanford.edu/data>.
- Löffler, Matthias, Anderson Y. Zhang, and Harrison H. Zhou (2021). “Optimality of spectral clustering in the Gaussian mixture model”. In: *The Annals of Statistics* 49.5, pp. 2506–2530. ISSN: 0090-5364. DOI: [10.1214/20-aos2044](https://doi.org/10.1214/20-aos2044).
- Luxburg, Ulrike von (Aug. 2007). “A tutorial on spectral clustering”. In: *Statistics and Computing* 17, pp. 395–416. DOI: [10.1007/s11222-007-9033-z](https://doi.org/10.1007/s11222-007-9033-z).
- Luxburg, Ulrike von, Mikhail Belkin, and Olivier Bousquet (Apr. 2008). “Consistency of spectral clustering”. In: *The Annals of Statistics* 36.2, pp. 555–586. DOI: [10.1214/009053607000000640](https://doi.org/10.1214/009053607000000640).
- Maier, Markus, Ulrike von Luxburg, and Matthias Hein (2013). “How the result of graph clustering methods depends on the construction of the graph”. In: *ESAIM: Probability and Statistics* 17, pp. 370–418. DOI: [10.1051/ps/2012001](https://doi.org/10.1051/ps/2012001).
- Maloutov, Igor and Regina Barzilay (July 2006). “Minimum Cut Model for Spoken Lecture Segmentation”. In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Sydney, Australia: Association for Computational Linguistics, pp. 25–32. DOI: [10.3115/1220175.1220179](https://doi.org/10.3115/1220175.1220179).
- Mohar, Bojan (1989). “Isoperimetric numbers of graphs”. In: *Journal of Combinatorial Theory, Series B* 47.3, pp. 274–291. ISSN: 0095-8956. DOI: [10.1016/0095-8956\(89\)90029-4](https://doi.org/10.1016/0095-8956(89)90029-4).

- Nadler, Boaz and Meirav Galun (Sept. 2007). “Fundamental limitations of spectral clustering”. In: *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. The MIT Press, pp. 1017–1024. ISBN: 9780262256919. DOI: [10.7551/mitpress/7503.003.0132](https://doi.org/10.7551/mitpress/7503.003.0132).
- Nascimento, Mariá C.V. and André C.P.L.F. de Carvalho (June 2011). “Spectral methods for graph clustering - A survey”. In: *European Journal of Operational Research* 211.2, pp. 221–231.
- Ng, Andrew Y., Michael I. Jordan, and Yair Weiss (2001). “On spectral clustering: analysis and an algorithm”. In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS’01. Vancouver, British Columbia, Canada: MIT Press, pp. 849–856.
- O, Suil, Yongtang Shi, and Zhenyu Taoqiu (2021). “Sharp upper bounds on the k -independence number in graphs with given minimum and maximum degree”. In: *Graphs and Combinatorics* 37.2, pp. 393–408. ISSN: 0911-0119. DOI: [10.1007/s00373-020-02244-y](https://doi.org/10.1007/s00373-020-02244-y).
- Orecchia, Lorenzo et al. (2008). “On partitioning graphs via single commodity flows”. In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC ’08. Victoria, British Columbia, Canada: Association for Computing Machinery, pp. 461–470. ISBN: 9781605580470. DOI: [10.1145/1374376.1374442](https://doi.org/10.1145/1374376.1374442).
- Orlin, James B. (2013). “Max flows in $O(nm)$ time, or better”. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. STOC ’13. New York, NY, USA: Association for Computing Machinery, pp. 765–774. ISBN: 9781450320290. DOI: [10.1145/2488608.2488705](https://doi.org/10.1145/2488608.2488705).
- Orlin, James B. and Xiao-yue Gong (2021). “A fast maximum flow algorithm”. In: *Networks. An International Journal* 77.2, pp. 287–321. DOI: [10.1002/net.22001](https://doi.org/10.1002/net.22001).
- Pedregosa, Fabian et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85, pp. 2825–2830. URL: <http://jmlr.org/papers/v12/pedregosa11a.html>.
- Peng, Yi et al. (Dec. 2014). “JF-Cut: A Parallel Graph Cut Approach for Large Scale Image and Video”. In: *IEEE Transactions on Image Processing* 24.
- Rölleke, Ulrike and Sarah Köster (2020). *NIH 3T3 microtubule cell dataset*. URL: <https://github.com/leosuchan/Xist>.
- Rozemberczki, Benedek, Carl Allen, and Rik Sarkar (May 2021). “Multi-Scale attributed node embedding”. In: *Journal of Complex Networks* 9.2, cnab014. ISSN: 2051-1329. DOI: [10.1093/comnet/cnab014](https://doi.org/10.1093/comnet/cnab014).
- Rozemberczki, Benedek, Ryan Davies, et al. (2019). “GEMSEC: Graph Embedding with Self Clustering”. In: *Proceedings of the 2019 IEEE/ACM*

- International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM, pp. 65–72.
- Sanders, Peter and Christian Schulz (2013). “Think Locally, Act Globally: Highly Balanced Graph Partitioning”. In: *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA ’13)*. Vol. 7933. LNCS. Springer, pp. 164–175.
- Senthilnath, J., S. Sindhu, and S. N. Omkar (Dec. 2014). “GPU-based normalized cuts for road extraction using satellite imagery”. In: *Journal of Earth System Science* 123.8, pp. 1759–1769. ISSN: 0973-774X. DOI: [10.1007/s12040-014-0513-1](https://doi.org/10.1007/s12040-014-0513-1).
- Sherman, Jonah (Oct. 2009). “Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$ -approximations to sparsest cut”. In: *IEEE 50th Annual Symposium on Foundations of Computer Science (FOCS 2009)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 363–372. DOI: [10.1109/FOCS.2009.66](https://doi.org/10.1109/FOCS.2009.66).
- Shi, Jianbo and Jitendra Malik (Aug. 2000). “Normalized cuts and image segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8, pp. 888–905. ISSN: 1939-3539. DOI: [10.1109/34.868688](https://doi.org/10.1109/34.868688).
- Šíma, Jiří and Satu Elisa Schaeffer (2006). “On the NP-completeness of some graph cluster measures”. In: *Proceedings of the 32nd Conference on Current Trends in Theory and Practice of Computer Science. SOFSEM’06*. Měřín, Czech Republic, pp. 530–537. ISBN: 354031198X. DOI: [10.1007/11611257_51](https://doi.org/10.1007/11611257_51).
- Suchan, Leo (2023). *R and Python implementations of the Xist algorithm*. <https://github.com/leosuchan/Xist>.
- Tang, Meng et al. (2016). “Normalized Cut Meets MRF”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, pp. 748–765. ISBN: 978-3-319-46475-6.
- Traag, Vincent et al. (July 2023). *vtraag/leidenalg*: 0.10.0. Version 0.10.0. DOI: [10.5281/zenodo.8147844](https://doi.org/10.5281/zenodo.8147844).
- Traag, Vincent A., Ludo Waltman, and Nees Jan van Eck (Mar. 2019). “From Louvain to Leiden: guaranteeing well-connected communities”. In: *Scientific Reports* 9.1, p. 5233. ISSN: 2045-2322. DOI: [10.1038/s41598-019-41695-z](https://doi.org/10.1038/s41598-019-41695-z).
- Wagner, Dorothea and Frank Wagner (1993). “Between min cut and graph bisection”. In: *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science. MFCS ’93*. Berlin, Heidelberg: Springer-Verlag, pp. 744–750. ISBN: 3540571825.
- Wang, Faqiang et al. (2020). “A Variational Image Segmentation Model Based on Normalized Cut with Adaptive Similarity and Spatial Regu-

- larization”. In: *SIAM Journal on Imaging Sciences* 13.2, pp. 651–684. DOI: [10.1137/18M1192366](https://doi.org/10.1137/18M1192366).
- Willis, William (2011). “Bounds for the independence number of a graph”. MA thesis. USA: Virginia Commonwealth University. DOI: [10.25772/B95B-C733](https://doi.org/10.25772/B95B-C733).
- Yu, Miao, Shuhan Shen, and Zhanyi Hu (2016). “Dynamic parallel and distributed graph cuts”. In: *IEEE Transactions on Image Processing* 25.12, pp. 5511–5525. ISSN: 1057-7149. DOI: [10.1109/TIP.2016.2609819](https://doi.org/10.1109/TIP.2016.2609819).
- Zhong, Guo and Chi-Man Pun (Dec. 2022). “Improved Normalized Cut for Multi-View Clustering”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12, pp. 10244–10251. ISSN: 1939-3539. DOI: [10.1109/TPAMI.2021.3136965](https://doi.org/10.1109/TPAMI.2021.3136965).

A Appendix

A.1 Further comparison study

Figure 5 is an extension of Figure 2, comparing the **Multi-Xist** algorithm and spectral clustering on the same cell image example, for different $k \in \{2, \dots, 9\}$. In particular, this exemplifies the selection process of **Multi-Xist**, namely which subgraph to cut.

A.2 Correctness of **Xist**

In the following we present a proof for the correctness of our algorithms as previously claimed in Theorem 3.1. As seen in the proof of this theorem, **Xist** and the **basic Xvst algorithm** yield the same output under Assumption 1. Here, we show the remaining claim, namely that **Xist** outputs $\min_{s,t \in V^{\text{loc}}} \text{XC}_{S_{st}}(G)$. Clearly, by definition of **Xist** (specifically lines 10-13), we are only left to show that through the selection method via τ , st -MinCuts for all pairs $s, t \in A := V^{\text{loc}}$ are considered.

We start with some preliminary results. The first of these statements and their proofs are due to Gomory and T. C. Hu, 1961, and the overall structure of this section as well as the remaining claims and the proof of Theorem A.5 are adapted from Gusfield, 1990, with changes necessary to account for the specific design of **Xist** and the generalization to only consider vertices from an arbitrary (but fixed) subset $A \subset V$.

Lemma A.1. *For a weighted graph $G = (V, E, \mathbf{W})$ and any $v_1, \dots, v_k \in V$, $k \in \mathbb{N}_{\geq 2}$, with $v_i \neq v_{i+1}$ for $i \in \{1, \dots, k-1\}$, denote $c_{ij} := \text{MC}_{S_{v_i v_j}}(G)$.*

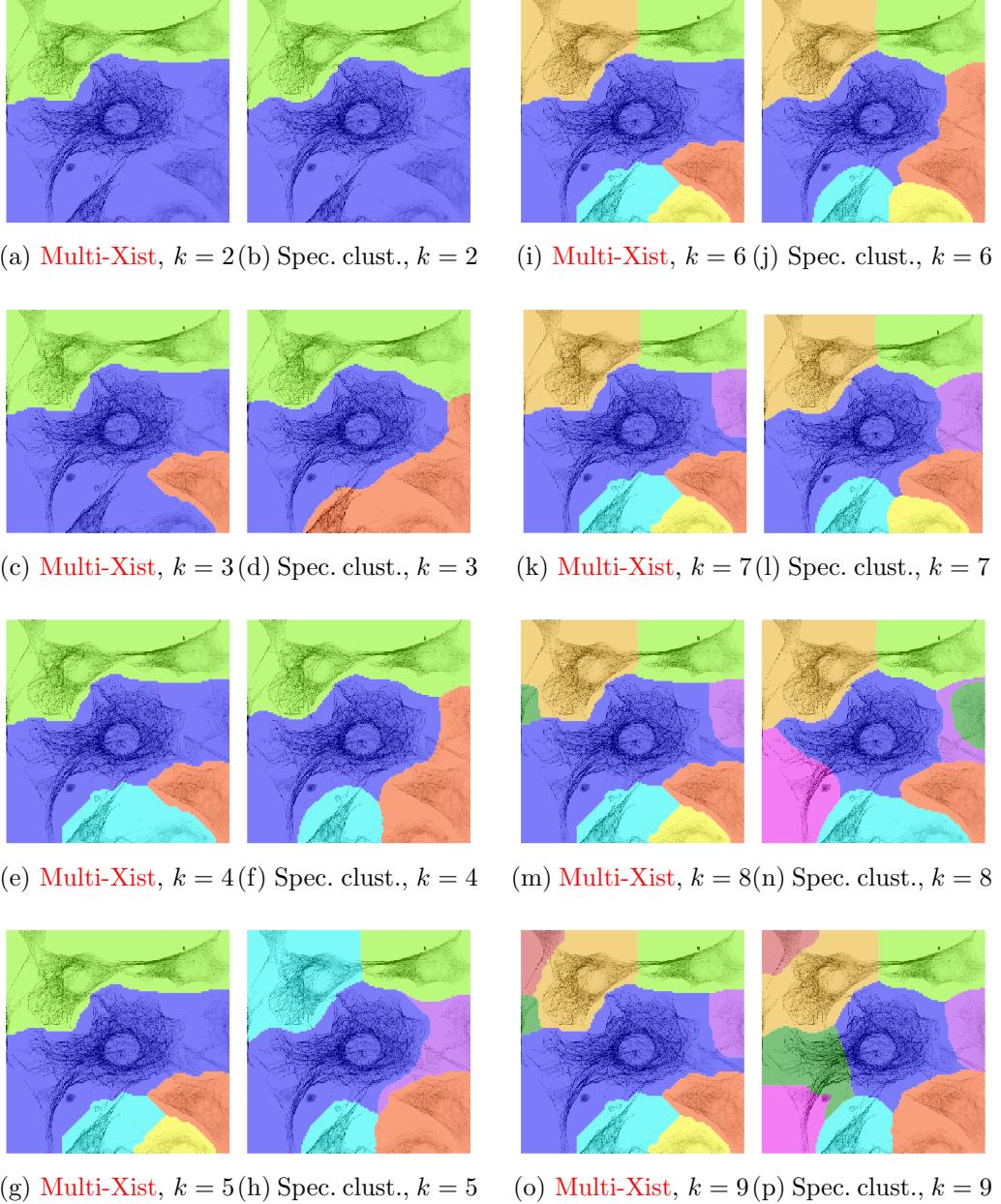


Figure 5: Extension of [Figure 2](#) on image segmentation of microtubules in NIH 3T3 cells. The colors visualize the resulting $k \in \{2, \dots, 9\}$ partitions via [Multi-Xist](#) and spectral clustering, and the underlying cell clusters are visible in black. The case $k = 10$ is depicted in [Figure 2](#).

Then

$$c_{1,k} \geq \min\{c_{1,2}, c_{2,3}, \dots, c_{k-1,k}\}. \quad (\text{A.1})$$

Proof. Suppose the claim does not hold, i.e. $c_{1,k} < c_{i,i+1}$ for all $i = 1, \dots, k-1$. Let $S \subset V$ denote the partition attaining the v_1v_k -MinCut $c_{1,k}$. Then, there exists an $i \in \{1, \dots, k-1\}$ such that v_i and v_{i+1} are on different sides of S (i.e. either $v_i \in S$ and $v_{i+1} \in S^c$ or vice versa) because $v_1 \in S$ and $v_k \in S^c$. Then, for this i , the partition attaining $c_{i,i+1}$ is also a valid v_1v_k -cut, and thus, as $c_{1,k}$ is the v_1v_k -MinCut value, $c_{1,k} \geq c_{i,i+1}$, contradicting the supposition. \square

Corollary A.2. *For a weighted graph $G = (V, E, \mathbf{W})$ and pairwise distinct $s, t, v \in V$, $\min\{\text{MC}_{S_{st}}(G), \text{MC}_{S_{sv}}(G), \text{MC}_{S_{vt}}(G)\}$ is not uniquely attained.*

The following [Lemma A.3](#) shows that for any two vertices u, v on the same side of an st -MinCut there is a uv -MinCut that preserves the st -MinCut partition by remaining on one side of the cut.

Lemma A.3. *Let $G = (V, E, \mathbf{W})$ be a weighted graph, let $s, t \in V$, $s \neq t$, and $u, v \in S_{st}$, $u \neq v$. Then, if $t \in S_{uv}^c$, $S_{st} \cap S_{uv}$ is a uv -MinCut, and if $t \in S_{uv}$, $S_{st} \cap S_{uv}^c$ is a uv -MinCut.*

Proof. For the uv -MinCut partition S_{uv} in G , define

$$A_1 := S_{st} \cap S_{uv}, \quad A_2 := S_{st} \cap S_{uv}^c, \quad A_3 := S_{st}^c \cap S_{uv}, \quad \text{and } A_4 := S_{st}^c \cap S_{uv}^c.$$

W.l.o.g. assume that $s, u \in A_1$ (otherwise exchange u and v and/or A_1 and A_3), and that $v \in A_2$ (otherwise exchange A_2 and A_4). For $k, \ell \in \{1, 2\}$ define

$$c_{k\ell} := \sum_{\substack{i \in A_k, j \in B_\ell \\ \{i,j\} \in E}} w_{ij}.$$

First assume that $t \in A_3$. Notice that A_2 is a valid uv -cut in G , and that $S_{uv} = A_1 \cup A_3$ is the uv -MinCut in G , so

$$\begin{aligned} 0 &\leq \text{MC}_{A_2}(G) - \text{MC}_{S_{uv}}(G) \\ &= (c_{21} + c_{23} + c_{24}) - (c_{12} + c_{14} + c_{32} + c_{34}) \\ &= c_{24} - (c_{14} + c_{34}) \end{aligned} \quad (\text{A.2})$$

since $c_{k\ell} = c_{\ell k}$ for all $k, \ell = 1, 2$. Now assume first that $t \in A_3$. Then, A_3 is a valid st -cut in G , and $S_{st} = A_1 \cup A_2$ is the st -MinCut in G , so that

$$\begin{aligned} 0 &\leq \text{MC}_{A_3}(G) - \text{MC}_{S_{st}}(G) \\ &= (c_{31} + c_{32} + c_{34}) - (c_{13} + c_{14} + c_{23} + c_{24}) \\ &= c_{34} - (c_{14} + c_{24}) \end{aligned} \quad (\text{A.3})$$

Adding (A.2) and (A.3) yields $0 \leq -2c_{14}$ and thus $c_{14} = 0$. Using (A.2) (or (A.3)) again gives $c_{24} = c_{34}$, so that

$$\text{MC}_{A_2}(G) = c_{21} + c_{23} + c_{24} = c_{12} + c_{14} + c_{32} + c_{34} = \text{MC}_{S_{uv}}(G),$$

i.e. A_2 also is a uv -MinCut in G . As A_2 is not affected by the contraction of S_{st}^c , the claim follows under the assumption that $t \in A_3$.

If, on the other hand, $t \in A_4$, we proceed analogously to before: Modify (A.2) by noting that A_1 is a valid uv -cut in G , resulting in $0 \leq c_{13} - (c_{23} + c_{34})$, and further modify (A.3) by noting that now A_4 is a valid st -cut in G , yielding $0 \leq c_{34} - (c_{13} + c_{23})$, so that together $c_{34} = 0$ and consequently $c_{13} = c_{23}$. Finally, this results in

$$\text{MC}_{A_1}(G) = c_{12} + c_{14} + c_{13} = c_{12} + c_{14} + c_{32} + c_{34} = \text{MC}_{S_{uv}}(G),$$

yielding the claim also in the case of $t \in A_4$. \square

In the following we present a short and direct proof that the **Xist** algorithm indeed considers st -MinCut partitions for all $s, t \in V^{\text{loc}}$ and, more significantly, that under the mild **Assumption 1**, the **basic Xvst algorithm** and **Xist** yield the same output. To that end we closely follow the proofs of Gusfield (1990, Section 2.1), adapting them to our case whenever necessary, in particular to accommodate for the fact that we are interested in the st -MinCut partitions (and not only the cut values themselves) and also the fact that we restrict our attention to only a subset $A \subseteq V$ of vertices (which can be arbitrary for the proof, but for our purposes will of course be V^{loc}).

To build intuition, we can consider the vector τ in the **Xist** algorithm as representation of a tree – in fact, in **Theorem A.5** it is shown that this is the Gomory-Hu tree. The interpretation is very simple: There is an edge between i and τ_i for every $i = 1, \dots, |A|$, where $A \subseteq V$ is the subset of vertices (on $G = (V, E, \mathbf{W})$) **Xist** is applied to, so τ can be thought of as a tree with branches between i and τ_i . In the following we abbreviate $N := |A|$ and let $A = \{1, \dots, N\} \subseteq V := \{1, \dots, n\}$ to ease notation. Call $u, v \in A$ *neighbours* at some point in **Xist** the uv -MinCut is computed (i.e. if $v = \tau_u$). Further, for any $u, v \in A$, a sequence of neighbours $\bar{uv} := \{u, v_1, \dots, v_k, v\}$ such that $u = \tau_{v_1}$, $v_k = \tau_v$ and $v_i = \tau_{v_{i+1}} \in A$ for all $i = 1, \dots, k-1$, for some $k \leq N$, is called a *directed path* from u to v .

Lemma A.4. *For a weighted graph $G = (V, E, \mathbf{W})$ apply **Xist** to an $A \subseteq V$ with $s, t \in A$, $s \neq t$, such that s and t are connected by a directed path \bar{st} , and let $v \in A$ be connected by a directed path to t such that $v < u$ for any $u \in \bar{st} \setminus \{t\}$. Then $s \in S_{vt}$ if and only if $v \in \bar{st}$.*

Proof. At the beginning of **Xist**, $\tau_s = 1$, and, from iterations 2 to $s - 1$ in line 5, τ_s changes from one j to one ℓ , $j, \ell \in \{1, \dots, N\}$, if and only if (between iterations 2 and $i - 1$) j and ℓ were *neighbours* (i.e. the cut between j and $\tau_j = \ell$ is computed at some point throughout **Xist**). Consequently, a node $j \in A$ was a neighbour of s at some point if and only if $j \in \overline{1s}$. As $t < v$, v must have been a neighbour of s before computation of the vt -cut. Moreover, as $v < u$ for any $u \in \overline{st} \setminus \{t\}$, t is a neighbour of s throughout the computation of the vt -MinCut (with partition S_{vt}). Thus, if $v \in \overline{st}$, $s \in S_{vt}$, and $s \notin S_{vt}$ otherwise. \square

Theorem A.5. *For a weighted graph $G = (V, E, \mathbf{W})$ and any subset $A \subseteq V$, **Xist** applied to G and A considers st -MinCuts for all pairs $s, t \in A$.*

Proof. First, note that for each $i = 2, \dots, N$, **Xist** computes an $i\tau_i$ -MinCut since always $\tau_i < i$ holds, and thus, after an $i\tau_i$ -MinCut is computed in step i , τ_i is not changed afterwards.

Let now $s, t \in A$, $s \neq t$, be arbitrary, and, depending on the context, consider the (directed) path \overline{st} either as a sequence of vertices s, v_1, \dots, v_k, t or edges $(s, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k), (v_k, t)$. Abbreviating $c_{uv} := \text{MC}_{S_{uv}}(G)$ and $C_{uv} := \{c_{ij} \mid (i, j) \in \overline{uv}\}$ (for any $u, v \in V$, $u \neq v$), we will show that

$$c_{st} = \min C_{st}$$

which yields the claim immediately because **Xist** computes all cuts in \overline{st} . By **Lemma A.1**, “ \geq ” holds. For the opposite direction, suppose that “ \leq ” does not hold, and let $s, t \in A$, $s \neq t$, be the vertices forming the shortest path in τ with this property, i.e. such that $c_{st} > \min C_{st}$.

First, assume that s and t are connected by a directed path in τ , i.e. $s = \tau_{v_1}, v_k = \tau_t$ and $v_i = \tau_{v_{i+1}}$ for all $i = 1, \dots, k - 1$. As the case $k = 0$ was tackled in the beginning of this proof, $v := v_k \in A \setminus \{s, t\}$. Due to \overline{st} being minimal (and **Lemma A.1**), $c_{sv} = \min C_{sv}$, so $c_{sv} = c_{vt} = \min C_{st}$ by **Corollary A.2** and the supposition that $c_{st} > \min C_{st}$. However, by **Lemma A.4**, S_{vt} is also a valid st -cut, so $c_{st} \leq c_{vt} = \min C_{st}$, leading to a contradiction.

Second, we tackle the case where s and t are not connected by a directed path, but rather that there is an $r \in A$ such that \overline{sr} and \overline{rt} each form a directed path in τ . As by the first case, $c_{sr} = \min C_{sr}$ and $c_{rt} = \min C_{rt}$ and thus, again, by **Corollary A.2** and the supposition we have $c_{sr} = c_{rt} = \min C_{st} = \min C_{sr} \cup C_{rt}$. Define $(u, v) \in \overline{sr}$ to be the vertices closest to r with $c_{uv} = \min C_{sr} = c_{sr}$. By **Lemma A.4**, $s \in S_{uv}$ and, since $c_{st} > \min C_{st} = \min C_{sr}$, $t \in S_{uv}$.

Let $r_1, r_2 \in A$ such that $(r_1, r) \in \overline{sr}$ and $(r, r_2) \in \overline{rt}$. W.l.o.g. assume that **Xist** computed the r_1r -MinCut before the rr_2 -MinCut. Then $s, u \in S_{r_1r} \cap S_{uv}$,

$v \in S_{r_1r} \cap S_{uv}^c$ and $t \in S_{r_1r}^c \cap S_{uv}$. We can now apply Lemma A.3 to the cuts S_{r_1r} and S_{uv} . This leads to two cases: If $r \in S_{uv}$, $S_{r_1r} \cap S_{uv}$ forms an uv -MinCut that is also a valid vr -cut, and if $r \in S_{uv}^c$, $S_{r_1r} \cap S_{uv}^c$ constitutes an uv -MinCut that is also a valid st -cut. In the second case, by definition of u and v , $c_{st} \leq c_{uv} = \min C_{st}$, showing the claim. In the first case, $c_{vr} \leq \min C_{st}$, but, since v and r are connected by a directed path in τ , by the above considerations $c_{vr} = \min C_{vr}$ as well as $\min C_{vr} > \min C_{st}$ since (u, v) was the closest pair to r that attains the minimum $\min C_{st}$. Together, $c_{vr} > \min C_{st}$, though this immediately contradicts the definition of (u, v) , finishing the proof. \square

A.3 Proof of Theorem 3.2

Proof. Since an entry w_{ij} of \mathbf{W} is zero if $\{i, j\} \notin E$, we only need V stored as a *list* (or a *self-balancing binary search tree*), and \mathbf{W} stored as an *array*, as input for the **basic Xvst algorithm** and **Xist**. We first analyze **Xist** line by line:

- 1 $\mathcal{O}(1)$ computations.
- 2 Determining V^{loc} requires computing the degree of every vertex and determining the local maxima. This can be done simultaneously, and takes at most n^2 additions and $n - 1$ comparisons while taking $\mathcal{O}(n^2)$ time.
- 3 Determining N takes $\mathcal{O}(N)$ computations, but only determining whether the algorithm terminates here requires only $\mathcal{O}(1)$ computations.
- 4 $\mathcal{O}(N)$ runtime due to the creation of τ .
- 5–14 Because of line 5 the following steps are executed exactly $N - 1$ times:
 - 6 $\mathcal{O}(1)$ computations.
 - 7 Orlin's algorithm Orlin, 2013 for computing an st -MinCut partition (by solving the dual problem of computing a max flow from s to t) takes $\mathcal{O}(nm + m^{31/16} \log^2 n)$, which can be reduced for specific types of graphs, e.g. $m = \mathcal{O}(n)$ yields $\mathcal{O}(n^2 / \log(n))$. If $m = \Omega(n^{1+\varepsilon})$, King, S. B. Rao, and Tarjan, 1994 provided an $\mathcal{O}(nm)$ algorithm which was recently improved by Orlin and Gong, 2021 who obtained $\mathcal{O}\left(\frac{nm \log n}{\log \log n + \log \frac{m}{n}}\right)$ for $m \geq n$. Thus, by King, S. B. Rao, and Tarjan, 1994; Orlin, 2013; Orlin and Gong, 2021, an st -MinCut partition can be computed in $\mathcal{O}(nm)$ time for all possible values of m and n .

9 By assumption, the computation time of the XC value for partition S_{st}^c is $\mathcal{O}(\kappa)$.

10–13 $\mathcal{O}(1)$ computations.

14–17 Iterating through all $j = 1, \dots, N$ takes $\mathcal{O}(N)$, and for each j , only $\mathcal{O}(1)$ computations are done, yielding $\mathcal{O}(N)$ in total.

Therefore, **Xist** runs in $\mathcal{O}(N \max\{nm, \kappa\})$ time for general m , and in $\mathcal{O}(N \max\{n^2/\log n, \kappa\})$ time for $m = \mathcal{O}(n)$ by utilizing Orlin, 2013 as noted above. As the **basic Xvst algorithm** iterates over all pairs of vertices in V in lines 2 to 9, to ascertain its complexity we simply substitute N by n^2 to obtain $\mathcal{O}(n^2 \max\{nm, \kappa\})$. \square