ClusterFuG: Clustering Fully connected Graphs by Multicut

Ahmed Abbas [1] **Paul Swoboda** [1 2]

Abstract

We propose a graph clustering formulation based on multicut (a.k.a. weighted correlation clustering) on the complete graph. Our formulation does not need specification of the graph topology as in the original sparse formulation of multicut, making our approach simpler and potentially better performing. In contrast to unweighted correlation clustering we allow for a more expressive weighted cost structure. In dense multicut, the clustering objective is given in a factorized form as inner products of node feature vectors. This allows for an efficient formulation and inference in contrast to multicut/weighted correlation clustering, which has at least quadratic representation and computation complexity when working on the complete graph. We show how to rewrite classical greedy algorithms for multicut in our dense setting and how to modify them for greater efficiency and solution quality. In particular, our algorithms scale to graphs with tens of thousands of nodes. Empirical evidence on instance segmentation on Cityscapes and clustering of ImageNet datasets shows the merits of our approach.

1. Introduction

Graph-based clustering approaches, primarily among them multicut (Chopra & Rao, 1993), are theoretically appealing: They do not need specification of the number of clusters, but infer them as part of the optimization process. They allow for a flexible clustering objective with attractive and repulsive costs between pairs of nodes. They are also theoretically well-understood as optimization problems with intensively studied polyhedral descriptions. Efficient solvers that scale well and give high quality solutions have also been developed.

[1]MPI for Informatics, Saarland Informatics Campus, Germany [2]University of Mannheim, Germany. Correspondence to: Ahmed Abbas <ahmed.abbas@mpi-inf.mpg.de>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

As a drawback, graph-based clustering approaches need specification of the underlying graph topology. In practice, this means an additional engineering effort as well as the possibility to not get it right, which would decrease the downstream task performance. Naively circumventing this challenge by using the complete graph is not scalable – the number of edges grows quadratically. One approach to resolve this conundrum is graph structure learning e.g., by extending (Kazi et al., 2022), but adds considerable additional complexity.

We propose a method to solve graph clustering efficiently on complete graphs. Our formulation will use the well-known edge-based multicut formulation and only restrict the way edge costs can be computed: they need to be based on inner products of node features. This has two advantages: First, it reduces storage requirements. Instead of storing a full adjacency matrix of edge costs as in multicut, which grows quadratically with the number of nodes, we only need to store a linear number of node features and can compute edge costs on demand. Second, operations needed in multicut algorithms can be made scalable. Instead of operating on the complete graph we can sparsify it adaptively during the solving process. This allows to simulate the workings of multicut algorithms on complete graphs by working on a small subset of it. The key technical ingredient to obtain these sparse subgraphs will be fast nearest neighbor search, for which efficient and scalable implementations exist (Johnson et al., 2019). In effect, this allows us to solve large dense multicut instances in moderate time, which is not possible with existing solvers. In detail, our contribution is as follows:

Formulation: We propose multicut on complete graphs with factorized edge costs as an efficiently representable graph clustering formalism.

Algorithm: We propose scalable algorithms for solving the dense multicut problems, one mimicking exactly the original greedy additive edge constraction (GAEC) algorithm (Keuper et al., 2015), the other a more efficient variant in the spirit of the balanced edge contraction heuristic (Kardoost & Keuper, 2018).

Empirical: We show efficacy in terms of memory and runtime of our solvers and show the merit of using them

for image segmentation on Cityscapes and clustering of ImageNet classification dataset[1].

2. Related work

Multicut and correlation clustering: The original multicut problem is formulated as an extension of the min-cut problem to multiple terminals with non-negative edge costs (Hu, 1963). In machine learning the multicut problem is defined differently and is equivalent (up to variable involution) to the correlation clustering problem (Demaine et al., 2006), i.e. arbitrary edges costs and no terminals. For the purpose of this work we will use the latter definition of multicut. The polyhedral geometry of the multicut problem has been studied in (Deza et al., 1992; Chopra & Rao, 1993; Oosten et al., 2001). An equivalent problem for multicut on complete graphs is the clique partitioning problem studied in (Grötschel & Wakabayashi, 1990).

Although the multicut problem is NP-Hard (Bansal et al., 2004; Demaine et al., 2006), greedy algorithms perform well in practice for computer vision and machine learning tasks (Keuper et al., 2015; Levinkov et al., 2017; Bailoni et al., 2022). More involved algorithms include message passing in the dual domain for multicut, studied in (Swoboda & Andres, 2017; Lange et al., 2018; Abbas & Swoboda, 2022). These algorithms give lower bounds and improved primal solutions. Another line of efficient primal heuristics is based on move-making (Beier et al., 2014; 2015). All these graphs, while efficient, scale with the number of edges, making them unsuitable for very large dense graphs. Algorithms for correlation clustering on complete graphs were proposed in (Pan et al., 2015; Veldt, 2022). However, they only allow unweighted edges. In this paper we consider efficient algorithms on full graphs and with weighted edges.

K-Means: The K-means problem (Lloyd, 1982) is similar to our approach in that it works directly on feature representations and its objective is based on L_2-distances between features. Similarly to our algorithm, large number of points are handled by efficiently computing kNN-graphs (Qaddoura et al., 2020), thereby reducing run time. In contrast to multicut, the number of clusters must be given a-priori, while in multicut it is derived as part of the optimization process.

Other clustering approaches: There are a number of other paradigms for clustering. A prominent approach is spectral clustering, in which a weighted graph is given and a clustering is computed with the help of the eigenvectors of the graph Laplacian (Von Luxburg, 2007; Jia et al., 2014). The work (Dhillon et al., 2007) shows connections between weighted k-means and multiple spectral clustering approaches. As for K-means and unlike multicut, spectral clustering requires the number of clusters to be specified.

[1]Code available at `https://github.com/aabbas90/cluster-fug`.

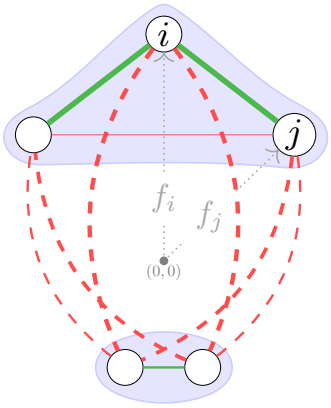

Figure 1: Example illustration of dense multicut problem (3) on 5 nodes. Each node i is associated with a vector $f_i \in \mathbb{R}^2$ and all possible edges between distinct nodes are considered (i.e., the complete graph). The edge cost between a pair of nodes i, j is measured by $\langle f_i, f_j \rangle$ and attractive/repulsive edges are colored green/red. Edge thickness represents absolute edge cost. Also shown is the optimal partitioning to 2 clusters with cut edges denoted by dashed lines.

3. Method

A *decomposition (or clustering)* of a weighted graph $G = (V, E, c)$ with vertices V, edges E and edge costs $c \in \mathbb{R}^E$ can be obtained by solving the following multicut problem

$$\min_{y \in \mathcal{M}_G} \sum_{ij \in E} c_{ij} y_{ij}. \tag{1}$$

We say that an edge ij with $c_{ij} > 0$ is *attractive*. Its endpoints prefer to be in the same cluster. In the opposite case $c_{ij} < 0$ we call the edge *repulsive*. The set $\mathcal{M}_G$ enumerates all possible partitions of G defined as

$$\mathcal{M}_G = \left\{ \mathbb{1}_{\delta(V_1,\dots,V_n)} : \begin{array}{c} n \in \mathbb{N} \\ V_i \cap V_j = \varnothing \quad \forall i \neq j \\ V_1 \dot\cup \dots \dot\cup V_n = V \end{array} \right\}. \tag{2}$$

where $\delta(\cdot, \dots, \cdot) \subseteq E$ is the set of edges straddling distinct components and $\mathbb{1}_\delta$ is the indicator vector of δ.

The goal of our work is to consider the scenario when the graph G is complete i.e., $E = \{ij : i \in V, j \in V \setminus \{i\}\}$. For large graphs storage and processing of edge costs c becomes prohibitive. To address this issue we instead require as input a feature vector $f_i \in \mathbb{R}^d$ for each node i in V. The edge costs between a pair of nodes i and j can then be measured on-demand through some function $s(f_i, f_j) \to \mathbb{R}$. In this

case the multicut problem becomes

$$\min_{y \in \mathcal{M}_G} \sum_{i \in V} \sum_{j \in V \setminus i} s(f_i, f_j) y_{ij}, \tag{3}$$

which we term as dense multicut problem. An illustration of our formulation is given in Figure 1. In the following we first revisit an algorithm to approximately solve (1) and show its extensions for dense multicut problem (3).

3.1. Greedy Additive Edge Contraction

The greedy additive edge contraction (GAEC) scheme (Keuper et al., 2015) computes approximate solution of the multicut problem (1) as given in Algorithm 1. It initializes each node as a separate cluster and iteratively contracts a pair of nodes i, j with the largest non-negative cost c_{ij} (if it exists). Let m be the node i and j are contracted to. The edge costs of edges incident to m are

$$c_{ml} = c_{il} + c_{jl}, l \in \mathcal{N}_i \cup \mathcal{N}_j \setminus \{i, j\}, \tag{4}$$

where costs of non-existing edges are assumed to be 0 and $\mathcal{N}_i$ corresponds to neighbours of i in graph G. For complete graphs directly applying this algorithm by operating on edge costs is computationally expensive. Moreover, since each node is connected to all other nodes ($\mathcal{N}_i = V \setminus \{i\}$), cost updates (4) during edge contraction take $\mathcal{O}(|V|)$ instructions.

Algorithm 1: `GAEC (Keuper et al., 2015)`

Data: Weighted graph $G = (V, E, c)$
Result: Clusters V

while $\max_{uv \in E} c_{uv} \geq 0$ **do**
  $m := ij = \arg\max_{uv \in E} c_{uv}$
  `// Aggregate edge costs`
  $c_{ml} = c_{il} + c_{jl}$, $l \in \mathcal{N}_i \cup \mathcal{N}_j \setminus \{i, j\}$
  `// Update edges`
  $E' = \{ml | l \in \mathcal{N}_i \cup \mathcal{N}_j \setminus \{i, j\}\}$
  $E = E' \cup E \setminus \{il\}_{l \in \mathcal{N}_i} \cup \{jl\}_{l \in \mathcal{N}_j}$
  `// Update nodes`
  $V = (V \cup \{m\}) \setminus \{i, j\}$

Contraction on complete graphs: We show how to perform a more efficient (and equivalent) contraction by operating on the node features f by our formulation (3) for the particular case of $s(\cdot, \cdot)$ defined as

$$s(f_i, f_j) = \langle f_i, f_j \rangle. \tag{5}$$

From now on, unless stated otherwise, our edge costs will be given by (5).

Lemma 3.1 (Contraction with node features)**.** *Assume edge costs are measured by* (5) *and nodes* i *and* j *are contracted to* m*. Then features of node* m *given by*

$$f_m = f_i + f_j \tag{6}$$

produce contracted edge costs according to (4)*.*

Proof. By applying (5) for $l \in V$ and comparing with (4) we get

$$\begin{aligned} s(f_m, f_l) = \langle f_m, f_l \rangle &= \langle f_i, f_l \rangle + \langle f_j, f_l \rangle \\ &= s(f_i, f_l) + s(f_j, f_l) \,. \quad \square \end{aligned}$$

Next we will build on the previous result to devise heuristics for solving dense multicut problem (3) efficiently.

GAEC for complete graphs: We devise an algorithm which exactly imitates GAEC (Keuper et al., 2015) but is applicable to our formulation on complete graphs (3). Specifically to make GAEC efficient with node features and a complete graph, we sparsify the original graph G by working on its directed k-nearest neighbours (NN) graph $(V, \mathcal{A})$. The NN graph stores candidate edges for contraction. The arc set $\mathcal{A}$ is populated by nearest neighbour search w.r.t. feature similarity (5) and is updated on each edge contraction. We denote outgoing neighbours of i as $\mathcal{N}_i^+ = \{l | (i, l) \in \mathcal{A}\}$ and similarly $\mathcal{N}_i^-$ as incoming neighbours. We write $\mathcal{N}_i^+ \cup \mathcal{N}_j^+$ as $\mathcal{N}_{ij}^+$ and similarly for incoming neighbours. Lastly the set $\arg\text{top-k}_{s \in S}\, g(s)$ contains the k elements of S having the largest values of $g(s)$. The complete strategy to obtain a feasible solution of dense multicut problem is described in Algorithm 2. It imitates Algorithm 1 by iteratively searching and contracting the most attractive edge, but it restricts its search only to the NN graph thereby reducing computation. After contraction, the NN graph is updated (lines 5-8) by only recomputing nearest neighbors of nodes which were affected by the contraction in the NN graph.

Algorithm 2: `Dense GAEC`

Data: Node features $f_i, \forall i \in V$; Number of nearest neighbours k
Result: Clusters V

`// Find nearest neighbours of each node`
1 $\mathcal{A} = \{(i, j) | i \in V, j \in \arg\text{top-k}_{i' \neq i} \langle f_i, f_{i'} \rangle\}$
2 **while** $\max_{(u,v) \in \mathcal{A}} \langle f_u, f_v \rangle \geq 0$ **do**
3 $m := (i, j) = \arg\max_{(u,v) \in \mathcal{A}} \langle f_u, f_v \rangle$
  `// Update nodes`
4 $f_m = f_i + f_j$
5 $V = (V \cup m) \setminus \{i, j\}$
  `// Update nodes having` i, j `as NN`
6 $H = \{(q, r) | q \in \mathcal{N}_{ij}^-, r \in \arg\text{top-k}_{l \in V \setminus q} \langle f_q, f_l \rangle\}$
  `// NN of merged node`
7 $H = H \cup \{(m, r) | r \in \arg\text{top-k}_{l \in V \setminus m} \langle f_m, f_l \rangle\}$
  `// Add arcs and remove arcs with` i, j
8 $\mathcal{A} = (\mathcal{A} \cup H) \setminus (\{(\cdot, i)\} \cup \{(\cdot, j)\} \cup \{(i, \cdot)\} \cup \{(j, \cdot)\})$

Proposition 3.2 (Dense Greedy Contraction)**.** *Algorithm 2 always merges a pair of nodes i and j with the largest edge cost i.e.,*

$$(i,j) \in \arg\max_{(u,v)\in\mathcal{A}} \langle f_u, f_v\rangle \implies \langle f_i, f_j\rangle \geq \max_{u,v\neq u} \langle f_u, f_v\rangle. \tag{7}$$

Proof. The statement is trivially satisfied before any merge operation is performed since $\mathcal{A}$ is constructed by nearest neighbour search over all nodes in line 1 of the algorithm. We now show that after each merge operation (i.e., after line 8 of the algorithm) the statement (7) still holds. We define $Q = m \cup \mathcal{N}_{ij}^-$. Two cases can arise:

Case 1: $\{i,j\} \cap Q \neq \varnothing$: Due to nearest neighbour search for all nodes in Q at lines 6 and 7, the statement holds.

Case 2: $\{i,j\} \cap Q = \varnothing$: In this case if i is the contracted node m from the last edge contraction operation then $(i,j) \in \mathcal{A}$ due to line 6. If $i \neq m$ then it remains connected to its nearest neighbours either due to the initial NN search at line 1 or the NN update at lines 6 and 7. □

Note that the claim of Prop. 3.2 does not ensure that the arcset $\mathcal{A}$ will contain all nearest neighbour arcs after contraction. Instead it guarantees that the most attractive edge will always be present in the nearest neighbour graph, foregoing the need to search in the complete graph. This proves that the Algorithm 2 performs locally optimal merges as proposed in (Keuper et al., 2015) and is also scalable to large complete graphs. As a downside the algorithm requires costly nearest neighbour search after every edge contraction. Since computing nearest neighbours and contracting edges is not commutative, in the worst case one has to recompute the nearest neighbours on the contracted graph from scratch.

Incremental nearest neighbours: For faster nearest neighbour updates after edge contraction we show how to reuse more of the previously computed nearest neighbors through the following two approaches. First, for all nodes whose nearest neighbours are merging nodes (i.e., line 6 of Alg. 2), we check if merged node m is already a nearest neighbour without requiring exhaustive search. Specifically assume a contracting node i was a k-nearest neighbour of some other node $q \in V \setminus i$. Then the merged node m is a k-nearest neighbour of q if $\langle f_q, f_m\rangle \geq \min_{l\in\mathcal{N}_q^+} \langle f_q, f_l\rangle$. This check can be cheaply performed for all such nodes thereby reducing computation. Second, we devise a criterion which can allow to efficiently populate nearest neighbours of the contracted node m.

Proposition 3.3 (Incremental nearest neighbours)**.** *Let the k-nearest neighbours $\mathcal{N}_i^+, \mathcal{N}_j^+$ of nodes i and j be given. Assume that nodes i, j are merged to form a new node m.*

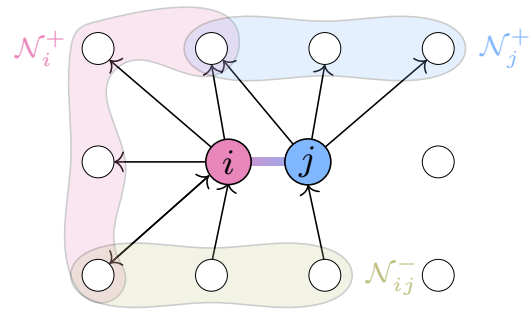

Figure 2: Illustration of nearest neighbour graph and an edge ij being contracted. The set $\mathcal{N}_{ij}^+ = \mathcal{N}_i^+ \cup \mathcal{N}_j^+$ is searched first to find nearest neighbours of the merged node efficiently (Prop. 3.3). The nodes in set $\mathcal{N}_{ij}^-$ need to update their nearest neighbours since their current nearest neighbour nodes i, j are getting contracted. Only the arcs to/from i, j are shown.

Then edge costs between nodes $v \in V \setminus \mathcal{N}_{ij}^+$ and m are bounded from above by

$$b_{ij} := \min_{p\in\mathcal{N}_i^+} \langle f_i, f_p\rangle + \min_{q\in\mathcal{N}_j^+} \langle f_j, f_q\rangle$$

Proof. Since neighbours of i are computed by nearest neighbours search we have for all nodes $p' \notin \mathcal{N}_i^+$

$$\langle f_i, f_{p'}\rangle \leq \min_{p\in\mathcal{N}_i^+} \langle f_i, f_p\rangle,$$

and similarly for node j. Then by definition of v and Lemma 3.1 we obtain

$$\begin{aligned}\langle f_m, f_v\rangle &= \langle f_i, f_v\rangle + \langle f_j, f_v\rangle \\ &\leq \min_{p\in\mathcal{N}_i^+} \langle f_i, f_p\rangle + \min_{q\in\mathcal{N}_j^+} \langle f_j, f_q\rangle .\end{aligned}$$ □

The above proposition gives an upper bound of feature similarity (i.e., edge cost) of merged node m with all nodes not in $\mathcal{N}_{ij}^+$. Thus if a node in $\mathcal{N}_{ij}^+$ exceeds this upper bound it is more similar to m than all nodes not in $\mathcal{N}_{ij}^+$. This allows to possibly skip recomputing the nearest neighbors of m in Alg. 2 (line 7).

Lemma 3.4. *If*

$$|\{p \in \mathcal{N}_{ij}^+ : \langle f_m, f_p\rangle \geq b_{ij}\}| \geq k \tag{8}$$

then k-nearest neighbour of node m given by $\arg\text{top-k}_{v\in V\setminus\{i,j,m\}} \langle f_m, f_v\rangle$ *can be chosen as* $\arg\text{top-k}_{p\in\mathcal{N}_{ij}^+} \langle f_m, f_p\rangle$.

Proof. Since the elements of $\mathcal{N}_{ij}^+$ already satisfy the bound b_{ij} from Prop. 3.3 and there are at least k many such elements, the k-nearest neighbours of node m can be taken from $\mathcal{N}_{ij}^+$. □

Both of these approaches for efficiently updating the NN graph after contraction are used in Alg. 3. Additionally in Alg. 3 we skip exhaustive search if a node still has p-many nearest neighbours where $p \in [1, k)$. Algorithm 3 can be used instead of lines 6 and 7 in Alg. 2 for improved performance. See Figure 2 for an illustration on nearest neighbour graph and edge contraction update.

Algorithm 3: `Incremental NN update`

Data: Contracting nodes i, j; Contracted node m; NN graph $(V, \mathcal{A})$; Node features $f_i, \forall i \in V$; Num. of neighbours k;

Result: Nearest neighbour arcs H to add in $\mathcal{A}$

`// NNs of m by Prop. 3.3`

1 $H = \{(m, l) | l \in \mathcal{N}^+_{ij}, \langle f_m, f_l \rangle \geq b_{ij}\}$

`// Keep at most k NN`

2 $H = \arg\text{top-k}_{(m,l)\in H} \langle f_m, f_l \rangle$

3 **if** $H = \varnothing$ **then**

4 $\quad H = \{(m, r) | r \in \arg\text{top-k}_{l \in V \setminus m} \langle f_m, f_l \rangle\}$

5 **for** $q \in \mathcal{N}^-_{ij} \setminus \{i, j\}$ **do**

$\quad$ `// Check if m a NN of q`

6 $\quad$ **if** $\langle f_q, f_m \rangle \geq \min_{l \in \mathcal{N}^+_q} \langle f_q, f_l \rangle$ **then**

7 $\quad\quad H = H \cup (q, m)$

8 $\quad$ **else**

9 $\quad\quad H = H \cup \{(q, r) | r \in \arg\text{top-k}_{l \in V \setminus q} \langle f_q, f_l \rangle)\}$

3.2. Lazy Edge Contraction

We further forego the need for nearest neighbours recomputation after edge contraction by lifting the restriction of performing only greedy moves. This allows to maximally utilize the NN graph: the algorithm performs contractions, including non-greedy ones, until no contraction candidates are present in the NN graph. Specifically we do not perform the exhaustive search in lines 4 and 9 of Alg. 3 and only return the nearest neighbours which are easily computable. The NN graph is repopulated as lazily as possible i.e., when no contraction candidates are left. In addition to being more efficient this strategy is reminiscent of the balanced edge contraction approach of (Kardoost & Keuper, 2018). The authors normalized the edge costs with cluster size of two end-points. These normalized edge costs were used to find the edge to contract. This strategy encouraged consecutive contractions to occur at different regions of the graph. As our lazy approach does not always make the nearest neighbours of the contracted node available thus contractions can only be done to nodes other than the contracted node. This also produces contractions in different regions.

Lastly we explore efficient methods for approximate nearest neighbour search (Malkov & Yashunin, 2018) for populating the initial NN graph. For later searches we still use exact methods as the search space is reduced due to contractions.

3.3. Varying Affinity Strength

Our basic edge costs computed by $\langle f_i, f_j \rangle$ for two features f_i and f_j have one fundamental limitation: Clusters will by default occupy whole quadrants. In other words, whenever two features have angle lower than $90°$ they are attractive and will prefer to be in the same cluster, see Figure 3. In order to let our formulation favor larger or smaller clusters, we modify our original similarity function $s(\cdot, \cdot)$ by adding an additional term indicated by α-variables:

$$\overline{f}_i = [f_i; \alpha_i], \tag{9}$$

$$s(\overline{f}_i, \overline{f}_j) = \langle f_i, f_j \rangle \pm \alpha_i \cdot \alpha_j \,, \tag{10}$$

where we choose positive sign for favoring larger clusters and negative for smaller clusters. In our experiments we will set $\alpha_i = \alpha > 0$, with $-$ in (10) to prefer many small sized clusters. Moreover, we note that our contraction mechanism carries over directly to this extended setting.

Lemma 3.5. *Aggregating features of the contracted node m by $\overline{f}_m = \overline{f}_i + \overline{f}_j$ is equivalent to setting edge costs as per* (4) *on complete graph.*

Proof. Similar to the proof of Lemma 3.1 as follows

$$\begin{aligned} s(\overline{f}_m, \overline{f}_l) &= \langle f_m, f_l \rangle \pm \alpha_m \cdot \alpha_l \\ &= \langle f_i + f_j, f_l \rangle \pm (\alpha_i + \alpha_j) \cdot \alpha_l \\ &= \langle f_i, f_l \rangle \pm \alpha_i \cdot \alpha_l + \langle f_j, f_l \rangle \pm \alpha_j \cdot \alpha_l \\ &= s(\overline{f}_i, \overline{f}_l) + s(\overline{f}_j, \overline{f}_l) \,. \end{aligned}$$

□

Large clusters: For preferring larger clusters (corresponding to choosing $+$ in (10)), we work directly on the extended feature set $\overline{f}_i = [f_i; \alpha_i]$ and use it in the NN graph.

Small clusters: For preferring smaller clusters (corresponding to choosing $-$ in (10)), we must modify our algorithms slightly. In order to construct NN graphs we will use two sets of features: First, the query nodes will have their features defined by $\hat{f}_i = [f_i, -\alpha_i]$ and second, the pre-existing nodes $j \in V$ in the graph will keep the same features $\overline{f}_j$ from (10). To search for nearest neighbors of node i in the graph V the modified similarity function (10) can be implemented by an inner product as

$$s(\overline{f}_i, \overline{f}_j) = \langle \hat{f}_i, \overline{f}_j \rangle \,. \tag{11}$$

3.4. Computational complexity

Our basic formulation (3) with $\alpha = 0$ in (10) is shown to be solvable in time $\mathcal{O}(|V|^{d^2})$ in (Veldt et al., 2017) through zonotope vertex enumeration (Onn & Schulman, 2001; Stinson et al., 2016). These results are also applicable when choosing $+$ in (10). In our experiments having $-$ in (10) is vital for obtaining a good clustering, in which case the results of (Veldt et al., 2017) are not directly transferable.

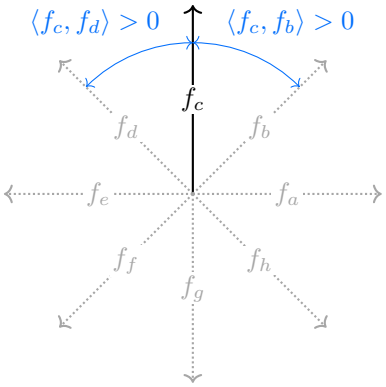

Figure 3: Illustration of edge costs between 8 nodes where feature vectors of each node i is in two-dimensional space i.e., $f_i \in \mathbb{R}^2$. If we want each node to be a separate cluster then the edge costs measured by (5) are not suitable. This is because there will always be atleast two vectors with positive costs preferring to be in the same cluster. Using a large enough positive value of α with a $-$ in (10) this issue can be resolved.

4. Experiments

We study the benefits of the multicut on complete graphs (3) and compare possible algorithms on the tasks of ImageNet (Deng et al., 2009) clustering and Cityscapes (Cordts et al., 2016) panoptic segmentation. All datasets are made available in (Swoboda et al., 2022). The algorithms are

`GAEC`: The greedy additive edge contraction algorithm from (Keuper et al., 2015)(Alg. 1) is run on the complete graph where all edge costs are precomputed and then passed to the algorithm.

`RAMA`: We also compare with the recent GPU-based multicut solver of (Abbas & Swoboda, 2022). Similar to `GAEC` we run it on the complete graph. The solver uses dual optimization for better solution quality and also gives lower bounds to the multicut objective (1). As a drawback it cannot handle large instances due to high memory requirement of complete graphs. We evaluate on NVIDIA A40 GPU with 48GB of memory.

`DGAEC`: Our Algorithm 2 which operates on node features and performs contractions according to Lemma 3.1. The nearest neighbour graph is updated by exhaustive search after edge contraction. The number of nearest neighbours k is set to 1, a larger value does not benefit since Prop. 3.3 is not utilized.

`DGAECInc`: Our Algorithm 2 which additionally makes use of Alg. 3 for incremental neighbour updates after edge contraction. The value of k is set to 5.

`DLAEC`: A variant of our `DGAECInc` where non-greedy moves are also allowed as described in Sec. 3.2.

`DAppLAEC`: Another variant of our `DLAEC` where initial nearest neighbours are computed by approximate nearest neighbour search method (Malkov & Yashunin, 2018) through library of (Johnson et al., 2019).

For all multicut algorithms on all datasets we set the value of affinity strength α_i in (11) to 0.4, preferring small clusters. We do not compare with the randomized algorithm of (Veldt et al., 2017) since it does not account for affinity strength with preference on smaller clusters. All CPU algorithms are run on an AMD 7502P CPU with a maximum of 16 threads to allow for faster nearest neighbour search.

4.1. ImageNet clustering

We evaluate clustering of the ImageNet (Deng et al., 2009) validation set containing $50k$ images. Each image in the dataset acts as a node for our dense multicut formulation. The features of each image are computed by a ResNet50 (He et al., 2016) backbone trained by MoCov3 (Chen et al., 2021) in unsupervised fashion by a constrastive loss on the training split of ImageNet. The features have a dimension of 2048 and are normalized to have unit L_2 norm. We create two problem instances containing $5k$ and $50k$ images by considering 100 and all 1000 classes respectively.

Clustering quality: Before comparing our algorithmic contributions we first test the efficacy of our dense multicut formulation by comparing its clustering result with k-means (Lloyd, 1982) using the implementation from (Pedregosa et al., 2011) and initialization of (Arthur & Vassilvitskii, 2007). Since k-means requires the number of clusters to be known beforehand we set it to the number of classes in the problem instance. For an additional comparison we also run k-means on the number of clusters given by our dense multicut algorithm. The quality of clustering results are evaluated using normalized mutual information (NMI) and adjusted mutual information (AMI) metrics (Vinh et al., 2010). The results are given in Table 1. We observe that although our formulation does not require the number of clusters to be specified, the results are on par with k-means. Additionally the value of affinity strength α does not need to be changed for different problem instances. As compared to k-means our algorithms are much faster especially on the larger instance. The `RAMA` solver of (Abbas & Swoboda, 2022) performs better than all other approaches on the smaller instance but runs out of memory for the larger one. Lastly, our formulation creates more clusters than the number of classes. This is mainly due to presence of outliers in the feature space as the feature extractor is trained without any groundtruth information.

Table 1: Quality of clustering on ImageNet validation set. t [s]: compute time in seconds, NMI: normalized mutual information, AMI: adjusted mutual information, # clusters: number of clusters, †: out of GPU memory. For k-means the number of clusters was specified as input.

Method	t [s] ↓	NMI ↑	AMI ↑	# clusters
ImageNet-100 ($\|V\| = 5k$)				
k-means	16	0.42	0.27	100
k-means	32	0.53	0.26	333
RAMA	**0.9**	**0.57**	**0.29**	639
DGAECInc	42	0.43	0.22	343
DAppLAEC	3.2	0.47	0.26	333
ImageNet-1000 ($\|V\| = 50k$)				
k-means	701	0.54	0.2	1000
k-means	1801	**0.61**	0.19	2440
RAMA	†	†	†	†
DGAECInc	2964	0.49	0.19	2488
DAppLAEC	**65**	0.56	**0.26**	2440

Table 2: Comparison of algorithms for solving dense multicut problem on two splits of Imagenet validation set. t [s]: compute time in seconds, Obj: objective value of clustering (3), †: out of GPU mem. ⋆: no result within 3 hours.

	ImageNet-100		*ImageNet*-1000	
Method	t [s] ↓	Obj ↓	t [s] ↓	Obj ↓
GAEC	4.5	-6.84e5	552	**-9.353e7**
RAMA	**0.9**	**-6.95e5**	†	†
DGAEC	132	-6.84e5	⋆	⋆
DGAECInc	42	-6.84e5	2934	**-9.353e7**
DLAEC	5	-6.83e5	341	-9.332e7
DAppLAEC	3.2	-6.83e5	**65**	-9.332e7

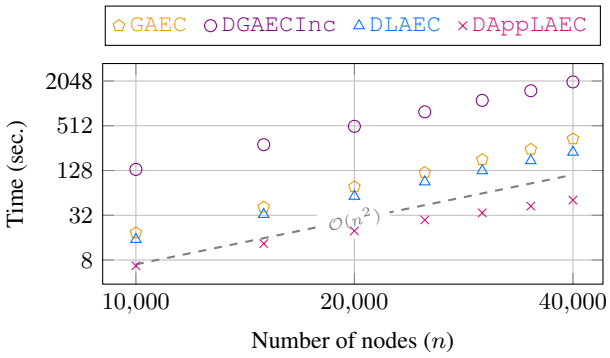

Figure 4: Runtime comparison of our algorithms on instances of varying sizes taken from ImageNet val. set by varying number of classes. Both axes are in log-scale. Our algorithms DGAECInc and DLAEC show quadratic complexity. Our algorithm DAppLAEC behaves subquadratically benefiting from approximate nearest neighbour search.

Algorithms comparison: We compare different algorithms for solving dense multicut problem (3) for imageNet clustering in Table 2. Firstly, we see that on the smaller instance the GPU based solver RAMA (Abbas & Swoboda, 2022) gives the best performance. Secondly using incremental nearest neighbour search through Alg. 3 gives better run time than exhaustive search. Lastly our non-greedy algorithms give the best run time among all CPU-based algorithms although with slightly worse objectives.

On the smaller instance, RAMA outperforms other algorithms in terms of the objective value (3) and also gives better clustering quality as compared to k-means. As a drawback RAMA cannot handle large dense multicut instances. This shows multicut on complete graphs can be a suitable alternative to k-means. We speculate that algorithmic improvements on top of our proposed algorithms will further improve clustering quality for large graphs.

Lastly we perform empirical time complexity analysis of our algorithms showing quadratic and subquadratic behaviour in Figure 4. More details are provided in the Appendix.

4.2. Panoptic segmentation

We evaluate our method on the task of panoptic segmentation (Kirillov et al., 2019) on the Cityscapes dataset (Cordts et al., 2016). The panoptic segmentation task consists of assigning a class label to each pixel and partitioning different instances of classes with object categories (e.g. car, person etc.). We focus on the task of partitioning for which the multicut formulation (1) has been used by (Kirillov et al., 2017; Abbas & Swoboda, 2021). The latter work used a carefully crafted graph structure. Our dense multicut (3) formulation foregoes the need for finding a suitable graph structure. We use the pretrained Axial-ResNet50 (Wang et al., 2021) network from (Yu et al., 2022), made available by (Weber et al., 2021) to compute the node features. Specifically, the network computes L_2-normalized, 128-dimensional and $4\times$ downsampled features in its intermediate stages which we use for our study without any training.

For our evaluation we first compute semantic class predictions and then create a dense multicut instance for each semantic category with objects (i.e., car, person etc.). Such classes are also known as *thing* classes. The goal of the multicut problem is then to partition all nodes belonging to same semantic class to different objects. This strategy creates a total of 1631 dense multicut problem instances of varying sizes from 500 images of the Cityscapes validation set. The

Table 3: Comparison of panoptic segmentation on Cityscapes dataset. Multicut on sparse graph of (Abbas & Swoboda, 2021) is computed by Alg. 1. For dense multicut we use the DAppLAEC algorithm. PQ_{th}: Average panoptic quality of all *thing* classes.

	Panoptic quality (%) ↑	
Category	Sparse multicut	Dense multicut
Person	40.0	**46.9**
Rider	53.0	**54.4**
Car	50.7	**60.5**
Truck	**52.7**	52.3
Bus	**72.1**	71.1
Train	**65.6**	62.9
Motorcycle	**47.0**	46.8
Bicycle	45.7	**46.9**
PQ_{th}	53.3	**55.2**

largest problem instance contains around $43k$ nodes. To upsample the clustering back to original image resolution we interpolate the node features back to input image resolution. Afterwards each upsampled node is assigned to the cluster whose mean feature embedding is most similar.

Clustering quality: As a first point of comparison we check whether formulating a multicut problem on the complete graph by (3) is beneficial as compared to a handcrafted sparse graph structure. We take the sparse graph structure from (Abbas & Swoboda, 2021) as a baseline. Their graph also includes long-range edges for dealing with occlusions leading to about $10 \cdot |V|$ edges in total. We compute the edge costs in this sparse graph in the same way as for our dense formulation and use Alg. 1 for computing multicut.

In Table 3 we compare the quality of clustering through the panoptic quality metric (Kirillov et al., 2019). We observe that our dense multicut formulation performs better than multicut on the sparse handcrafted graph. This improvement is significant for classes which can have many instances of the same class within an image (i.e. person, car) thus making the partitioning problem difficult. For classes with large objects (e.g. truck) having more edges does not help since the sparse graph can already capture most inter-pixel relations. On average our dense multicut formulation gives better results than sparse multicut while alleviating the need for designing a graph structure.

Algorithms comparison: We compare dense multicut algorithms for the panoptic segmentation task in terms of objective value and run time. We were not able to run RAMA since the GPU could not store large graphs. The comparison of performance to the remaining algorithms averaged over all problem instances is given in Table 4.

Table 4: Comparison of algorithms for solving dense multicut problem on Cityscapes validation set. (t [s]): average compute times in seconds, (Obj): average objective value of clustering (3). The average is calculated over all problem instances.

Method	t [s] ↓	Obj ($\times 10^6$) ↓
GAEC	7.7	-6.338
DGAEC	84.1	-6.338
DGAECInc	3.2	-6.338
DLAEC	2.1	-6.340
DAppLAEC	**1.5**	**-6.341**

Table 5: Results of panoptic segmentation via dense multicut with different values of attraction/repulsion strength α in (10). PQ_{th}: Avg. panoptic quality over all *thing* classes.

α	0.2	0.3	0.4	0.5	0.6	0.7	0.8
PQ_{th}	54.5	**55.8**	55.2	55.0	54.1	52.0	49.3

In terms of run time, we see that our most naive algorithm DGAEC is slower than GAEC which directly operates on edge costs. Our other algorithms surpass GAEC reaching up to an order of magnitude run time improvement with lazy edge contractions and approximate initial nearest neighbours search. In terms of objective value we see slight improvement by our lazy contraction algorithms as compared to the greedy ones.

Sensitivity of affinity strength: In Table 5 we study the effect of changing the value of α from (10). The results highlight that having $\alpha > 0$ is essential for good clustering quality. Last, we see further improvement if the value of α is set differently for each semantic class. We refer to the Appendix for further results.

5. Conclusion

We have demonstrated that optimizing multicut on large complete graphs is possible when using factorized edge costs through inner products of features. We speculate that further algorithmic improvements are possible e.g. by performing dual optimization directly on the node features.

As a potential theoretical advantage our approach sidesteps the need for learning graph structure. This offers a possibility to embed it as a differentiable layer in neural networks, using e.g. the work (Vlastelica et al., 2019).

6. Acknowledgements

We are grateful to all reviewers especially reviewer TfhS for their helpful suggestions. We also thank Jan-Hendrik Lange for discussion about related works.

References

Abbas, A. and Swoboda, P. Combinatorial optimization for panoptic segmentation: A fully differentiable approach. *Advances in Neural Information Processing Systems*, 34: 15635–15649, 2021.

Abbas, A. and Swoboda, P. RAMA: A Rapid Multicut Algorithm on GPU. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8193–8202, June 2022.

Arthur, D. and Vassilvitskii, S. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pp. 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

Bailoni, A., Pape, C., Hütsch, N., Wolf, S., Beier, T., Kreshuk, A., and Hamprecht, F. A. GASP, a Generalized Framework for Agglomerative Clustering of Signed Graphs and Its Application to Instance Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11645–11655, 2022.

Bansal, N., Blum, A., and Chawla, S. Correlation clustering. *Machine learning*, 56(1-3):89–113, 2004.

Beier, T., Kroeger, T., Kappes, J. H., Kothe, U., and Hamprecht, F. A. Cut, glue & cut: A fast, approximate solver for multicut partitioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 73–80, 2014.

Beier, T., Hamprecht, F. A., and Kappes, J. H. Fusion moves for correlation clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3507–3516, 2015.

Chen, X., Xie, S., and He, K. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9640–9649, 2021.

Chopra, S. and Rao, M. R. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

Demaine, E. D., Emanuel, D., Fiat, A., and Immorlica, N. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Deza, M., Grötschel, M., and Laurent, M. Clique-web facets for multicut polytopes. *Mathematics of Operations Research*, 17(4):981–1000, 1992.

Dhillon, I. S., Guan, Y., and Kulis, B. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29 (11):1944–1957, 2007. doi: 10.1109/TPAMI.2007.1115.

Grötschel, M. and Wakabayashi, Y. Facets of the clique partitioning polytope. *Mathematical Programming*, 47 (1):367–387, 1990.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, T. C. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963.

Jia, H., Ding, S., Xu, X., and Nie, R. The latest research progress on spectral clustering. *Neural Comput. Appl.*, 24(7–8):1477–1486, jun 2014. ISSN 0941-0643. doi: 10.1007/s00521-013-1439-2.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7 (3):535–547, 2019.

Kardoost, A. and Keuper, M. Solving minimum cost lifted multicut problems by node agglomeration. In *Asian Conference on Computer Vision*, pp. 74–89. Springer, 2018.

Kazi, A., Cosmo, L., Ahmadi, S.-A., Navab, N., and Bronstein, M. M. Differentiable graph module (dgm) for graph convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):1606–1617, 2022.

Keuper, M., Levinkov, E., Bonneel, N., Lavoué, G., Brox, T., and Andres, B. Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

Kirillov, A., Levinkov, E., Andres, B., Savchynskyy, B., and Rother, C. Instancecut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9404–9413, 2019.

Lange, J.-H., Karrenbauer, A., and Andres, B. Partial optimality and fast lower bounds for weighted correlation clustering. In *International Conference on Machine Learning*, 2018.

Levinkov, E., Kirillov, A., and Andres, B. A comparative study of local search algorithms for correlation clustering. In *GCPR*, 2017.

Lloyd, S. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

Onn, S. and Schulman, L. J. The vector partition problem for convex objective functions. *Mathematics of Operations Research*, 26(3):583–590, 2001.

Oosten, M., Rutten, J. H., and Spieksma, F. C. The clique partitioning problem: facets and patching facets. *Networks: An International Journal*, 38(4):209–226, 2001.

Pan, X., Papailiopoulos, D., Oymak, S., Recht, B., Ramchandran, K., and Jordan, M. I. Parallel correlation clustering on big graphs. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Qaddoura, R., Faris, H., and Aljarah, I. An efficient clustering algorithm based on the k-nearest neighbors with an indexing ratio. *International Journal of Machine Learning and Cybernetics*, 11(3):675–714, 2020.

Stinson, K., Gleich, D. F., and Constantine, P. G. A randomized algorithm for enumerating zonotope vertices. *arXiv preprint arXiv:1602.06620*, 2016.

Swoboda, P. and Andres, B. A message passing algorithm for the minimum cost multicut problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Swoboda, P., Hornakova, A., Roetzer, P., Savchynskyy, B., and Abbas, A. Structured prediction problem archive. *arXiv preprint arXiv:2202.03574*, 2022.

Veldt, N. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In *International Conference on Machine Learning*, pp. 22060–22083. PMLR, 2022.

Veldt, N., Wirth, A. I., and Gleich, D. F. Correlation clustering with low-rank matrices. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 1025–1034, 2017.

Vinh, N. X., Epps, J., and Bailey, J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010. URL `http://jmlr.org/papers/v11/vinh10a.html`.

Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Wang, H., Zhu, Y., Adam, H., Yuille, A., and Chen, L.-C. Max-deeplab: End-to-end panoptic segmentation with mask transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5463–5474, 2021.

Weber, M., Wang, H., Qiao, S., Xie, J., Collins, M. D., Zhu, Y., Yuan, L., Kim, D., Yu, Q., Cremers, D., Leal-Taixe, L., Yuille, A. L., Schroff, F., Adam, H., and Chen, L.-C. DeepLab2: A TensorFlow Library for Deep Labeling. *arXiv: 2106.09748*, 2021.

Yu, Q., Wang, H., Qiao, S., Collins, M., Zhu, Y., Adam, H., Yuille, A., and Chen, L.-C. k-means mask transformer. In *European Conference on Computer Vision*, pp. 288–307. Springer, 2022.

Appendix

A. Time complexity analysis

Theoretically all of our algorithms have asymptotic time complexity of $\mathcal{O}(d \cdot |V|^3)$. However, empirically we observe our algorithms show quadratic behaviour and get faster through Prop. 3.3 and lazy contractions. We analyse empirical complexity of our algorithms in Figure 4. More details about worst-case complexity of our algorithms are as under

`GAEC`: In worst case scenario Alg. 2, the set $\mathcal{N}_{ij}^-$ in line 6 can be $V \setminus \{i, j\}$. Therefore nearest neighbour search in line 6 has complexity $\mathcal{O}(d \cdot |V|^2)$ making each edge contraction operation quadratic. Overall complexity of the algorithm will then be $\mathcal{O}(d \cdot |V|^3)$.

`DGAEC`: In worst case scenario Prop. 3.3 might not help requiring exhaustive search for nearest neighbours. Thus asymptotic complexity remains same as `DGAEC`.

`LAEC`: Worst case complexity remains same as `DGAEC`.

`DAppLAEC`: Here we use approximate nearest neighbour search (Malkov & Yashunin, 2018) but only to populate initial set of nearest neighbours i.e., the most costly operation. For later iterations we still use exhaustive search therefore asymptotic complexity remains the same as `DLAEC`.

Note that above complexity analysis assumes that the number of nearest neighbours k is set to 1. This offers very limited potential for incremental nearest neighbour updates. A larger value of k gives much speedup due to Alg. 3. A case distinction is provided below

$k = 1$: Assume the k-nearest neighbour graph with $k = 1$ before edge contraction has the structure: $V = \{1, 2, ..., n\}$, $\mathcal{A} = \{(2, 1), (3, 1), ...(n, 1)\}$. Thus node 1 is the nearest neighbour of all other nodes. If an edge containing node 1 is contracted it will force all other nodes to recompute their nearest neighbours. Note that there are still $\mathcal{O}(|V|)$ many remaining nodes requiring nearest neighbour update. Due to this worst-case scenario time complexity of one edge contraction becomes quadratic making overall runtime cubic in the number of nodes.

$k \gg 2$: Assume the node set after contracting an edge ij is $V' := V \setminus \{i, j\}$. Then each node in V' still has $k - 2$ many nearest neighbours from within V'. In this case nearest neighbour queries only need to be performed between the merged node and nodes in V'. In such case an edge contraction operation can have linear complexity instead of quadratic in the number of nodes. Since we use a value of $k \in [1, 5]$ in all our algorithms utilizing incremental updates, they show such behaviour. This is also demonstrated in empirical analysis from Figure 4.

B. Influence of affinity strength

On the Cityscapes dataset we compare panoptic quality on different object classes by varying the value of affinity strength α in (11). The results are given in Table 6. We observe that for classes contain many small objects large value of α is suitable whereas for classes with large objects small value of α is preferable. Although our default value of 0.4 already makes dense multicut outperform the baseline, further improvement is still possible e.g. by tuning α.

Table 6: Comparison of panoptic segmentation on Cityscapes dataset for different values of affinity strength α (11). All results are computed using the DAppLAEC algorithm. Largest values in each row are highlighted with bold.

	Panoptic quality on varying values of α								
Category	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Person	31.5	38.1	43.2	46.9	49.8	52.6	54.3	**55.0**	52.4
Rider	51.1	53.0	53.9	54.5	**55.5**	55.4	53.9	51.0	45.5
Car	45.6	52.9	57.8	60.5	63.3	**64.8**	64.1	62.2	57.8
Truck	**54.1**	53.7	52.7	52.3	49.0	47.8	45.4	41.5	34.7
Bus	**75.1**	74.2	73.5	71.2	69.3	63.6	58.5	54.5	47.3
Train	**75.0**	74.9	71.5	62.9	56.3	51.7	45.1	40.4	32.3
Motorcycle	45.5	46.1	48.0	46.8	48.7	**49.1**	47.8	45.2	39.8
Bicycle	38.1	43.2	45.6	46.9	47.8	**48.0**	46.9	44.6	40.4
Average (PQ_{th})	52.0	54.5	**55.8**	55.2	55.0	54.1	52.0	49.3	43.8