Clustering in Hypergraphs to Minimize Average Edge Service Time*

Ori Rottenstreich[1], Haim Kaplan[2], and Avinatan Hassidim[3]

1 **Princeton University, Princeton, NJ, USA**
orir@cs.princeton.edu
2 **Tel Aviv University, Tel Aviv, Israel**
haimk@post.tau.ac.il
3 **Bar-Ilan University, Ramat Gan, Israel**
avinatan@cs.biu.ac.il

Abstract

We study the problem of clustering the vertices of a weighted hypergraph such that on average the vertices of each edge can be covered by a small number of clusters. This problem has many applications such as for designing medical tests, clustering files on disk servers, and placing network services on servers. The edges of the hypergraph model groups of items that are likely to be needed together, and the optimization criteria which we use can be interpreted as the average delay (or cost) to serve the items of a typical edge. We describe and analyze algorithms for this problem for the case in which the clusters have to be disjoint and for the case where clusters can overlap. The analysis is often subtle and reveals interesting structure and invariants that one can utilize.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Clustering, average cover time, hypergraphs, set cover

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.64

1 Introduction

Between 15% and 20% of the population suffers from some form of allergic contact dermatitis [26]. One of the most common ways to treat this is to find the allergen, and avoid it. In order to find the allergen the doctor applies *patch tests* to the patient. Each patch test is applied by attaching a patch containing a cluster of several different allergens to the patient's back. The doctor first decides which allergens to test based on anamnesis. Then she picks a set of clusters that contains all suspected allergens and applies the corresponding patch tests. The study of this paper answers the question how to cluster different allergens together such that common anamnesis require a small number of patch tests. This is in order to reduce the cost and patient's discomfort. Such an abstraction is relevant of course in any scenario in which tests (medical or other) are performed in clusters and one has to design the clusters.[1]

* Work by Haim Kaplan has been supported by Grant 1161/2011 from the German-Israeli Science Foundation, by Grant 1841-14 from the Israel Science Foundation, and by the Israeli Centers for Research Excellence (I-CORE) program (center no. 4/11).

[1] This medical setting may remind the reader of *group testing*. In group testing we want to locate individuals who have a certain property by testing the individuals against groups of properties, rather than against individual ones, and we want to minimize the number of groups. Here we also group properties into tests, but we may have different properties that we try to locate among different subsets and our objective is different.

A similar clustering problem arises in several other application areas. For example, in network design when a network operator has to apply a subset of functions (services such as Deep Packet Inspection, Network Address Translation, etc.) to the packets of each flow. In networks supporting Network Function Virtualization (NFV) these functions are implemented in software on general-purpose servers, where each server can run a limited number of functions [10, 27]. Here we need to assign the functions to servers (a cluster is a set of functions assigned to the same server) such that heavy flows can be served by a small number of servers to minimize delay. Unlike the medical setting, for NFV it is often the case that we need to apply the functions to the packets within a prescribed order.[2]

For a different application consider the task of assigning papers to sessions in a conference with a single track. We would like to construct the program such that attendees interested in particular topics can hear all talks on these topics by attending a small number of sessions.

Another application is in disk servers where one would like to cluster on the same server files that are often read together for minimizing the number of servers that have to be accessed.

1.1 Formal definition of our clustering problem

Our input is a hypergraph $G = (V, E)$ where $|V| = n$. Each edge $e \in E$ has a positive *weight ("frequency")* $w(e)$ satisfying $\sum_e w(e) = 1$. Our goal is to partition V into a collection P of $\alpha = \lceil n/c \rceil$ disjoint clusters, $P = \{B(1), B(2), \ldots, B(\alpha)\}$, $B(i) \subseteq V$, where each cluster is of size no larger than c. For each edge $e \in E$ we define the service time of e to be $t(e) = |\{B \in P \mid B \cap e \neq \emptyset\}|$. Our objective is to compute a clustering that minimizes[3] the average service time $\sum_e w(e)t(e)$.

We also consider the variant of this problem in which the clusters can overlap. For a given number of clusters α and cluster size c such that $\alpha \geq \lceil n/c \rceil$ we want to compute a collection of clusters $P = \{B(1), B(2), \ldots, B(\alpha)\}$ such that each cluster is of size at most c and $\cup_{B \in P} B = V$. In this case we may be able to cover e with a subset of the clusters $\{B \in P \mid B \cap e \neq \emptyset\}$. So, our clustering algorithm is also required to compute a small cover $P(e) \subseteq \{B \in P \mid B \cap e \neq \emptyset\}$, such that $e \subseteq \cup_{B \in P(e)} B$, for each edge e. We define $t(e) = |P(e)|$ and our goal is to compute a clustering and edge covers $P(e)$ that minimize the average service time $\sum_e w(e)t(e)$.

Last we consider the version of this problem in which each edge $e \in E$ is an *ordered* tuple of vertices (rather than a subset of the vertices), say $e = (v_1, \ldots, v_{|e|})$. In this case $P(e)$ has to be a sequence of clusters $B_1, B_2, \ldots, B_{t(e)}$, possibly with repetitions[4] such that there exist $i_1, i_2, \ldots i_{t(e)}$ where $\{v_1, \ldots, v_{i_1}\} \subseteq B_1$, $\{v_{i_1+1}, \ldots, v_{i_2}\} \subseteq B_2$, etc.

We denote the optimal average service time by T_{OPT}. In the above applications, the average time can describe the number of patch tests that have to be applied on average for an anamnesis, the average number of servers a flow has to visit to implement its required functions, the number of sessions one has to attend or the number of disk servers required to be accessed. The maximal allowed cluster size c models a restriction on the number of examined allergens in a patch test, the maximal number of functions that can be implemented in a server, the number of papers in a conference session or the number of files a disk server can save.

---

[2] In this setting the maximum load on a processor is also a relevant metric, which is not a part of this treatment.

[3] We assume without loss of generality that $|e| > 1$ for every $e \in E$ since edges of size 1 just contribute their weight to the cost of any clustering.

[4] That is we may have $B_i = B_j$ for $i \neq j$, $t(e)$ is the size of $P(e)$ counting repetitions.

Table 1 Summary of the results.

# Clusters α	Cluster size c	Hypergraph edges e	Result
Unordered edges			
n/c	2	–	Optimal algo.
–	2	$\lvert e\rvert = 2$	Optimal algo.
n/c	–	$\lvert e\rvert = 2$	$\frac{2c+1}{c+2}$ Approx. algo.
n/c	3	$\lvert e\rvert \leq 3$	5/3 Approx. algo.
–	2	$\lvert e\rvert = 3$	NP-hardness
n/c	3	$\lvert e\rvert = 2$	NP-hardness
n/c	$n/2$	$\lvert e\rvert = 2$	NP-hardness
–	–	–	Bi-criteria approx. algo.
Ordered edges			
n/c	–	–	Approx. algo.

1.2 Our results

We give an algorithm that computes an optimal clustering for the case of disjoint clusters of size $c = 2$. We also give an optimal algorithm for the case of overlapping clusters of size 2 when our hypergraph is in fact a graph (all edges are of size 2). These algorithms compute the optimal clustering by finding maximum matchings in related graphs. In case of disjoint clusters the construction of the graph is relatively straightforward whereas for overlapping clusters the reduction is more sophisticated and requires solving multiple matching problems (see Section 2 and Section 3.1).

In contrast with these positive results we show that when clusters are allowed to overlap, $c = 2$, and the edges of the hypergraph are of size 3 the problem is already NP-hard. Moreover, when the clusters are required to be disjoint the problem becomes NP-hard for $c = 3$ even if $|e| = 2$ for all $e \in E$, so we cannot hope for polynomial algorithms that compute the optimal clustering in a more general setting. This motivates the design of approximation algorithms.

To understand which approximation ratios we are targeting, notice that for any $e \in E$, $t(e) \leq |e|$ and since each cluster is of size at most c, $t(e) \geq \lceil |e|/c \rceil$. This implies that the average service time of any two clusterings is within a factor of c from each other. In particular an arbitrary clustering gives a c-approximation. (Clearly an approximation ratio of $\max\{|e| \mid e \in E\}$ is also achieved by an arbitrary clustering.) Getting an approximation ratio strictly better than c is not trivial.

For disjoint clusters and hypergraphs with edges of size 2 or 3 we describe and analyze a greedy strategy. If all edges are of size 2 we show that this algorithm obtains a clustering of cost at most $(2c+1)/(c+2)$ times the cost of the optimal clustering. When edges are of size 2 or 3 and $c = 3$ we prove that the approximation ratio is at most 5/3. We can generalize the greedy algorithm (in several ways) for hypergraphs with larger edges and for larger values of c but these variants are more complicated to analyze (see Section 2.1 and Section 2.2).

Our analysis (for hypergraphs of edges of size 2 and 3) is subtle and relies on the fact that a clustering has to pay more for edges that cannot be covered by a single cluster. We show that when the optimal clustering is much better than the greedy one, then the subsets of the edges that they cover are almost disjoint. Since they are almost disjoint, it must be that the optimal clustering covers many edges by more than a single cluster (those that are covered by a single cluster in the greedy clustering), and hence it has to pay for them. Interestingly, we observe that for hyperedges with edges of size 3 or more, a stronger phenomena occurs: If

the optimal clustering covers many more edges than the greedy clustering then it must be the case that there are edges which are not covered by neither the greedy nor the optimal clustering. We do not know exactly how to exploit this phenomenon, and leave it as an open question. We hope that this observation would lead to a tighter analysis of a generalization of the greedy for hypergraphs with larger edges.

We give a bi-criteria approximation algorithm to compute overlapping clusters in any hypergraph for any value of c. This algorithm produces a clustering of $O(\alpha \log M \log c)$ clusters whose average service time is larger than the optimal service time with α clusters by a factor of $O(\log M \log c)$. Here M denotes the maximum cardinality of an edge (see Section 3.2).

We use our approximation algorithm for disjoint clusters in the case where all edges are of size 2 to develop an approximation algorithm for the case of disjoint clustering in an ordered hypergraph with edges of arbitrary sizes (see Section 3.3).

Our results are summarized in Table 1.

1.3 Related work

Clustering has always been an important problem, and a lot of research has been done on this topic. Clustering has applications in many different fields, including machine learning, vision, information retrieval and bioinformatics. Different applications have different metrics for the quality of the clustering, and consequently use different algorithms [16]. Some of the more common quality measures include various distance metrics (e.g., the Davies-Bouldin index [12] or the Dunn index [14]), spectral properties [25, 31, 29], and correlation (in correlation clustering [5]).

Clustering is usually a partition of the data (often represented as a graph), but overlapping clusters have also been studied for at least 45 years [11]. Recent applications include solving partial differential equations [6, 18], analysis of social networks [24, 3], wireless networks [1] and solving algorithmic problems on large graphs [7, 4]. One of the challenges in this case is to define the right measure for the quality of the clustering. Taking the standard measure (e.g., the sum of the weights of the edges crossing clusters over the sum of the weight of edges inside clusters) does not give any benefit to overlapping clusters.

One way to measure the quality of a partition is to perform a random walk, and see how long it stays in the same cluster (equivalently how often the random walk crosses clusters). Anderson et al. generalize this metric to overlapping clusters [4]. Their clustering is composed of overlapping clusters that cover all the vertices in the graph, and in addition, each vertex has its primary cluster (one of the clusters it belongs to). To evaluate a clustering and a choice of primary clusters, they start a random walk at a random vertex v_1. Let t_1 denote the number of steps the walk stays in the primary cluster of v_1. Let v_2 denote the first vertex outside that cluster the walk visits. Let t_2 denote the number of steps the walk stays in the primary cluster of v_2, etc. The clustering is good if the expected value of $t_1 + t_2 + t_3 + \ldots$ is large.

There has been many works that deal with non overlapping hypergraph clustering problems. Motivated by applications in computer vision, Agarwal et al. [2] proposed a two phased approach, which first projects the hypergraph to a weighted graph, and then uses graph clustering techniques. Zhou et al. [32] generalize the spectral techniques to work directly on hypergraphs, without the projection stage. Shashua et al. [28] use tensor factorization instead of spectral methods. Lately, Leordeanu and Sminchisescu [22] used an iterative method based on solving a series of LPs, to obtain faster clustering algorithms. Finally, Bulò and Pelillo [8] apply a game theoretic approach, in which every cluster is being

controlled by an agent who tries to maximize the size of her cluster, and the equilibrium status determines the partition. We note that the classical work on hypergraph clustering deal with non overlapping clusters, and that the used metrics differ from ours.

After the selection of clusters, the decision which of them to use to cover each of the edges is an instance of the minimum set cover problem. In the set cover problem the input is a universe U of $n = |U|$ elements and k sets $S_1, \ldots, S_k \subseteq U$. The goal is to find a collection with a minimal number of sets such that its union equals the universe U. Set cover is NP-hard as shown in Karp's seminal paper [21]. A greedy algorithm, selecting as the next set one that covers a maximal number of elements that have not been covered, gives a $\ln n$ approximation. Feige showed that assuming $P \neq NP$, no polynomial-time algorithm can obtain an approximation ratio better than $\ln n$ [17]. Furthermore, if the cardinality of each set is at most c, the greedy algorithm obtains roughly a $1 + \ln c$ approximation [20, 23, 9]. Trevisan [30] adjusted the parameters in Feige's reduction, to show that if the largest set is of size c, no polynomial-time algorithm can obtain an approximation ratio better than $\ln c - O(\ln \ln c)$, assuming $P \neq NP$.

2 Disjoint clusters

In this section we study the case that $\alpha = n/c$, i.e., the clusters are disjoint.

We start with the case of $c = 2$ for which we can find the optimal clustering as follows. We construct a weighted complete (undirected) simple graph $\Lambda = (V, F)$ over the vertices of our input hypergraph $G = (V, E)$ and set the weight $c(u, v)$ of an edge $(u, v) \in F$ to be $\sum_{e \in E | \{u,v\} \subseteq e} w(e)$. We claim that a maximum perfect matching in Λ gives an optimal clustering. Correctness of this algorithm follows from the observation that the average service time of a clustering $P = \{B(1), B(2), \ldots, B(n/c)\}$ is exactly

$$\sum_{e \in E} w(e) \left(|e| - |\{B \in P \mid B \subseteq e\}| \right) = \sum_{e \in E} w(e)|e| - \sum_{B \in P} \sum_{e | B \subseteq e} w(e).$$

This holds since we can trivially serve an edge e with $|e|$ clusters – one per vertex. Each cluster B_i with $|B_i| = 2$ and $B_i \subseteq e$ can be used to serve two of the vertices of e and thereby reduces by one the total number of required clusters. Since clusters are disjoint their contributions add up. We note that in the special case where $|e| = 2$ for each $e \in E$ and E connects every pair of vertices from V then Λ and G are identical, and the optimal solution is given by a maximum matching in G.

2.1 The greedy algorithm for a graph

We start with the case where $|e| = 2$ for every $e \in E$, meaning that our input hypergraph $G = (V, E)$ is in fact a graph. The value of c is arbitrary, and assume for simplicity that $|V|$ is a multiple of c.

A *solution* is a partition of V into disjoint clusters of c vertices. We refer to a partition of the vertices in which every cluster is of size *at most* c as a *partial solution*. We define the *score* $s(B)$ of a cluster B (of size at most c) to be the sum of the weights of the edges contained in B, i.e., $s(B) = \sum_{e \in E | e \subseteq B} w(e)$.

We analyze a simple greedy algorithm that at every step, chooses as the next cluster the set B of maximum score among all possible clusters of size c consisting of uncovered vertices.

▶ **Theorem 1.** *The greedy algorithm results in an average service time of at most* $\frac{2c+1}{c+2} T_{OPT}$.

For $c = 3$ the approximation ratio is $7/5$, and for $c = 4$ it is $3/2$.

We need the following definitions for the proof of Theorem 1. Given a solution or a partial solution X, we define its score $s(X)$ as the sum of the scores of its clusters. In particular, we consider the optimal and the greedy partitions denoted by OPT and GREEDY with scores of $s(OPT)$ and $s(GREEDY)$, respectively. Finally, let $\mathcal{W}$ denote the sum of the weights of the edges in the graph G. We begin with the following lemma that relates $s(GREEDY)$, $s(X)$ for some solution X, and $\mathcal{W}$.

▶ **Lemma 2.** *For every graph $G = (V, E)$ and every solution X and $c \geq 2$, the following relations hold:*
(i) $s(GREEDY) \geq s(X)/c$,
(ii) $(c-1) \cdot s(GREEDY) + \mathcal{W} \geq 2s(X)$.

Proof Outline. The proof is by induction on the number of vertices in the graph. The basis is the case where $|V| \leq c$. In this case $GREEDY$ selects one cluster containing all vertices of G so $s(GREEDY) = \mathcal{W}$. It follows that $s(GREEDY) \geq s(X) \geq s(X)/c$ and $(c-1)s(GREEDY) + \mathcal{W} \geq s(GREEDY) + \mathcal{W} \geq 2 \cdot s(X)$.

Induction step: We assume the lemma holds for any graph with less than $|V|$ vertices. Let $B_1 \subseteq V$ be the first cluster of size c that GREEDY chooses in G. Let $s(B_1)$ be the score of this cluster, that is $s(B_1) = \sum_{v,v' \in B_1, (v,v') \in E} w(v, v')$.

Let $G' = (V', E')$ be the graph generated by deleting from G the vertices in B_1 and all the edges incident to these vertices. That is, $V' = V \setminus B_1$ and $E' = \{(u, v) \in E \mid \{u, v\} \cap B_1 = \emptyset\}$.

We derive from the clustering X of G, a clustering X' of G', by removing from each cluster in X the vertices in B_1, and keeping only clusters with at least two remaining elements following the removal. Formally, the clusters of X' are $\{A \setminus B_1 \mid |A \setminus B_1| \geq 2, A \in X\}$. Let $GREEDY'$ be the solution obtained by running the greedy algorithm in G', which is the same as $GREEDY \setminus B_1$. We have that $s(GREEDY) = s(B_1) + s(GREEDY')$.

The inductive hypothesis applied to G' gives that $s(GREEDY') \geq s(X')/c$, and that $(c-1)s(GREEDY') + \mathcal{W}' \geq 2s(X')$, where $\mathcal{W}'$ is the total weight of the edges in G'.

Let E_X^- be the set of edges in E that are covered by a cluster of X but not covered by a cluster of X' and let $w(E_X^-)$ be the sum of the weights of the edges in E_X^-. Clearly, $E_X^- \subseteq E \setminus E'$ and we have that $s(X) = w(E_X^-) + s(X')$ and

$$\mathcal{W} \geq \mathcal{W}' + w(E_X^-) \ . \tag{1}$$

Let $X_1 \subseteq X$ be the set of clusters in X covering the edges in E_X^-. Since $|B_1| = c$ we must have that $|X_1| \leq c$. Since B_1 was selected by GREEDY it follows that $s(B_1) \geq s(A)$ for every cluster $A \in X$ and in particular for every cluster $A \in X_1$. So it follows that

$$s(B_1) \geq \frac{s(X_1)}{|X_1|} \geq \frac{w(E_X^-)}{|X_1|} \geq \frac{w(E_X^-)}{c} \ . \tag{2}$$

We can now show that $s(GREEDY) \geq s(X)/c$ by combining the induction hypothesis with Equation (2) as follows.

$$s(GREEDY) = s(B_1) + s(GREEDY') \geq w(E_X^-)/c + s(X')/c = s(X)/c \ .$$

To show that $(c-1)s(GREEDY) + \mathcal{W} \geq 2s(X)$, we distinguish between the cases where
(i) $|X_1| \leq c-1$ and
(ii) $|X_1| = c$.
We establish case (ii) by a stronger version of inequality (1) that holds in case and says that $\mathcal{W} \geq \mathcal{W}' + w(E_X^-) + s(B_1)$. ◀

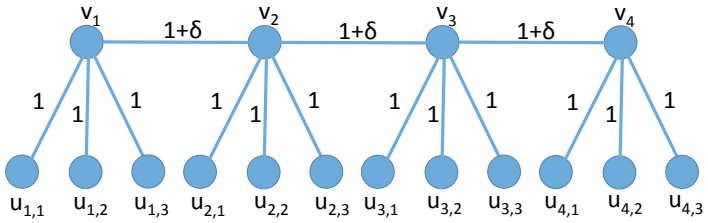


Figure 1 Illustration of the graph $G_{c,\epsilon}$ for $c = 4$ in Lemma 3. It consists of $n = c^2$ vertices $\{v_1, \ldots, v_c\} \cup \{u_{i,j} \mid 1 \le i \le c, 1 \le j \le c-1\}$ and $c^2 - 1$ edges of two types. For every $1 \le i \le c-1$ we have an edge (v_i, v_{i+1}) of weight $1 + \delta$ where $\delta = \epsilon/c(c-1)$. For every $1 \le i \le c$, and every $1 \le j \le c-1$ there is an edge $(v_i, u_{i,j})$ of weight 1.

We now use Lemma 2 to prove Theorem 1.

Proof Outline of Theorem 1. The service time for an edge is 1 if it is contained in one of the clusters of the partition, and 2 otherwise. So we can bound the ratio of the average service time of GREEDY, denoted by T_{GREEDY}, and the average service time of OPT by

$$\frac{T_{GREEDY}}{T_{OPT}} = \frac{2\mathcal{W} - s(GREEDY)}{2\mathcal{W} - s(OPT)} \le \frac{2c+1}{c+2}.$$

The last inequality follows from the bounds in Lemma 2 applied with $X = OPT$ and some algebraic manipulations. ◀

Lemma 2 is tight for any fixed value of c and therefore the approximation ratio of Theorem 1 is also tight. To show this we use the graph $G_{c,\epsilon}$ (for any $\epsilon > 0$ and c) illustrated in Figure 1 for which we prove the following lemma.

▶ **Lemma 3.** *In the graph $G_{c,\epsilon}$, we have $(c-1)s(GREEDY) + \mathcal{W} \le 2s(OPT) + \epsilon$, and simultaneously $s(GREEDY) \le s(OPT)/c + \epsilon$. In particular for this graph $T_{GREEDY} / T_{OPT} \ge (2c+1)/(c+2) - \epsilon$.*

2.2 The case of a hypergraph

Having established tight bounds on the approximation ratio for the case $|e| = 2$, we now move to the more difficult case where $|e| \le 3$ and $c = 3$. We again describe a simple greedy algorithm and bound its approximation ratio.

For $G = (V, E)$, let $E_2 \subseteq E$ and $E_3 \subseteq E$ be the subsets of the edges of size 2 and 3, respectively. A *solution* is a partition of V into triplets. We also refer to a partition of the elements in which every part is of size at most 3 vertices as a *partial solution*. Given a solution or a partial solution X, we denote by X_3 the set of edges $e \in E_3$ that are also triplets in X; by X_2 the set of edges $e \in E_2$ which are contained in a triplet or pair of X; and by $X_{2,3}$ the set of edges $e \in E_3$ such that $|e \cap B| = 2$ for some pair or triplet $B \in X$.

We define the *score*, $s(X)$, of a solution (or a partial solution) X to be $s(X) = 2w(X_3) + w(X_2) + w(X_{2,3})$. In particular, we define the score $s(B)$ of a pair or a triplet $B \in X$ to be twice the weight of B if $B \in E_3$, plus the sum of the weights of the edges $e \in E$ such that $|e \cap B| = 2$. Intuitively, this is the contribution of B to the reduction in the average service time. The average service time T_X of a solution X equals $3w(E_3) + 2w(E_2) - s(X)$.

We consider a greedy algorithm, denoted by $GREEDY$ that at each step picks a triplet B with maximum score and then removes the vertices of B, and all the edges whose restriction

to the remaining graph is of size at most one. The following analog of Lemma 2 is the main technical lemma of this subsection.

▶ **Lemma 4.** *For any solution X the following relations hold*
(i) $s(GREEDY) \geq s(X)/3$.
(ii) $3w(E_3) + 1.5w(E_2) + 3s(GREEDY) \geq 3s(X)$.

Applying this Lemma to $X = OPT$ we prove the following bound on the approximation ratio of $GREEDY$.

▶ **Theorem 5.** *For hypergraphs with $|e| \leq 3$ for all e and $c = 3$, the average service time of the greedy algorithm is at most $\frac{5}{3} \cdot T_{OPT}$.*

▶ Remark. Our upper bound in Section 2.1 for the case where the input graph G is a graph is tight as Lemma 3 shows. This follows since both parts of Lemma 2 are tight for the graph $G_{c,\epsilon}$. On the other hand, we believe that our result in Theorem 5 for the case in which G is a multigraph is not tight as we suspect that there is no graph for which both parts of Lemma 4 are tight. Lemma 4 can be extended for the case of hyperedges of size even larger than 3, but we believe that this approach is unlikely to provide tight bounds. An obvious open problem is to find a way to strengthen Lemma 4 and improve our bounds for the case where G is a hypergraph.

3 Overlapping clusters

In this section we study the scenario of a general number $\alpha \geq n/c$ of clusters that can overlap and are not necessarily disjoint. In the first part of the section we focus on the case where $c = 2$, that is each cluster can include two vertices from V. We give a polynomial-time algorithm that finds an optimal clustering for the case where $|e| = 2$ for all $e \in E$, i.e., the input is a graph. (We recall that without loss of generality we assumed $|e| > 1$ for every $e \in E$.) In the full version of this paper we show that the problem is NP-hard for hypergraphs in which $|e| = 3$ for all $e \in E$ (and $c = 2$). This motivates the second part of this section in which we describe an approximation algorithm that applies to a general instance of the problem.

3.1 Optimal algorithm for a graph

We consider the case where $c = 2$ and $|e| = 2$ for all $e \in E$. Notice that in Section 2, we gave an algorithm that finds an optimal clustering for the case where $c = 2$ and $\alpha = n/c$ but without any assumption on $|e|$. When $c = 2$, $\alpha = n/c$ and $|e| = 2$ for all e, the algorithm we give here and the algorithm of Section 2 are identical.

To simplify the presentation we assume that the input graph G contains all the edges (some may have weight 0) and thereby a clustering P is just a subset of E. Edges of P are served by a single cluster and each other edge is served by two clusters. Let $w(P) = \sum_{e \in P} w(e)$. It follows that the service time of P is $2 \cdot \sum_{e \in E} w(e) - w(P)$. The following characterization of an an optimal clustering now easily follows.

▶ **Theorem 6.** *Let $c = 2$ and α be an arbitrary value. For any weighted graph $G = (V, E)$, a clustering $P = \{e(1), e(2), \ldots, e(\alpha)\}$ minimizes the average service time if and only if P maximizes $w(P)$ while satisfying $\bigcup_{i \in [1,\alpha]} e(i) = V$.*

Assume without loss of generality that $w(e) \neq w(e')$ by some consistent tie breaking scheme. Let $e_1, e_2, \ldots, e_{\binom{n}{2}}$ be the edges of G in decreasing order of weight (by our tie

breaking). Let P be an optimal clustering that includes the longest prefix of $e_1, e_2, \ldots, e_{\binom{n}{2}}$ (among all optimal clusterings). Let x be the length of this prefix and let $e(1), e(2), \ldots, e(\alpha)$ be the edges of P in non-increasing order of weight. By the choice of P, $e(i) = e_i$ for $1 \leq i \leq x$ and $e(x+1) \neq e_{x+1}$ if $x < \alpha$. Here are a few observations about P that follow from Theorem 6.

We say that a cluster $e(j) \in P$ *first covers* a vertex v, if it is the first (according to the order defined above) among the clusters of P that contains v. Each vertex is first covered by exactly one of the clusters and each of the clusters $e(x+1), \ldots, e(\alpha)$ must first cover either one or two vertices. Indeed, if, say, $e(i)$ for some $x+1 \leq i \leq \alpha$, does not first cover any of its vertices, we can replace it in P by e_{x+1} and get a clustering P' such that, $w(P') \geq w(P)$, P' covers all vertices, and P' contains a longer prefix of edges in the sequence $e_1, e_2, \ldots, e_{\binom{n}{2}}$, contradicting the choice of P.

Let y_1 (resp. y_2) be the number of clusters among $e(x+1), \ldots, e(\alpha)$ that first cover a single vertex (resp. two vertices). Clearly we have that $\alpha = x + y_1 + y_2$. Let α_x be the number of distinct vertices in the first x pairs, i.e. $\alpha_x = |\bigcup_{i=1}^{x} e(i)|$. Since there are n vertices and each is first covered exactly once, then $n = \alpha_x + y_1 + 2y_2$. The two equalities imply that

$$y_2 = n - \alpha_x - \alpha + x \;\text{ and }\; y_1 = \alpha - x - y_2 = 2\alpha - 2x - n + \alpha_x. \tag{3}$$

Consider a vertex v that is first covered by a cluster $e(j)$, which first covers only v. We claim that $e(j)$ is the edge of largest weight (first in $e_1, e_2, \ldots, e_{\binom{n}{2}}$) that covers v, as otherwise we can replace it in P by an edge of larger weight, while still covering all vertices of G, contradicting the maximality of P.

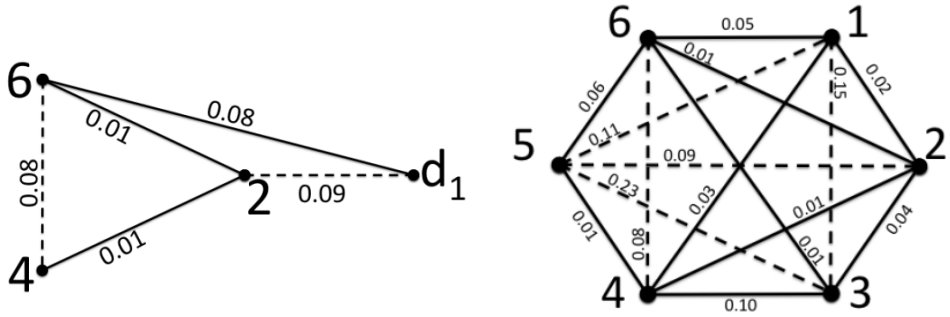
We now turn to describe the algorithm. We first sort the pairs of vertices by their weight and compute the order $e_1, e_2, \ldots, e_{\binom{n}{2}}$. Then we iterate over all possible values of $x \in [0, \alpha]$. For each value x, we construct a clustering $P = \{e(1), e(2), \ldots, e(\alpha)\}$ with $e(i) = e_i$ for $1 \leq i \leq x$ that maximizes $w(P)$ among all such clusterings. We compute α_x and the values of y_1 and y_2 given by Equation (3). To find the $y_1 + y_2 = \alpha - x$ additional clusters, we consider the induced subgraph Λ of G on the $n - \alpha_x$ vertices not covered by the largest x edges of G. We add to this induced subgraph, y_1 additional dummy vertices $d_1, \ldots, d_{y_1}$, and obtain a graph Λ' with $n - \alpha_x + y_1 = 2(\alpha - x) = 2(y_1 + y_2)$ vertices. In Λ' we add an edge (d_i, v) for each $1 \leq i \leq y_1$ if the edge of largest weight covering v is e_j for some $j > x + 1$. We set $w(d_i, v) = w(e_j)$. A maximum perfect matching in Λ' gives us the $n - x$ remaining clusters as argued by the following lemma.

▶ **Lemma 7.** *A perfect matching in Λ' corresponds to the $y_1 + y_2$ edges among $e_{x+2}, \ldots, e_{\binom{n}{2}}$ of largest weight containing the $n - \alpha_x$ vertices of G that are not in $e_1, \ldots, e_x$. An edge of Λ corresponds to itself and an edge (d_i, v), where d_i is a dummy vertex, corresponds to the edge incident to v of largest weight in G.*

The total weight of the clustering defined by a perfect matching in Λ' is given by the sum of the weights of $e_1, \ldots, e_x$ and the weight of the matching. The optimal clustering is selected as the one with the maximum total weight among the clusterings that we get for the various values of x. An example demonstrating the algorithm is illustrated in Figure 2. Finding a maximum weight perfect matching in a general (not necessarily bipartite) graph is a classical combinatorial optimization problem that can be solved in polynomial-time by various implementations of Edmond's algorithm [15] (see also [13] and the references there). The current best strongly polynomial bound is $O(qr + r^2 \log r)$ by Gabow [19] (here q denotes the number of edges and r denotes the number of vertices). Since the number of vertices in

{3,5} 0.23, **{1,3} 0.15**, **{1,5} 0.11**, {3,4} 0.10, {2,5} 0.09, {4,6} 0.08, {5,6} 0.06, {1,6} 0.05, {2,3} 0.04, {1,4} 0.03, {1,2} 0.02, {2,4} 0.01, {4,5} 0.01, {2,6} 0.01, {3,6} 0.01.

(a) The graph edges, sorted by weight.



(b) The corresponding constructed graph Λ'. **(c)** The complete weighted graph.

Figure 2 Illustration of the optimal algorithm for $c = 2$ in a graph (edges of two vertices) (for the described edge weights with $\alpha = 5, n = 6$). (a) shows the graph edges sorted in a non-increasing order of their edge weights, shown next to each edge. The first $x = 3$ pairs appear in bold with $\alpha_x = |\{1, 3, 5\}| = 3$ and $n - \alpha_x = 3$. (b) presents the constructed graph Λ' with vertices $2, 4, 6$ that do not appear in the $x = 3$ pairs and a single additional dummy vertex. (c) shows the complete weighted graph for all vertices in which dashed edges represent clusters in an optimal clustering.

any of the graphs in which we compute a perfect matching is $O(\alpha)$, each maximum matching problem is solved in $O(\alpha^3)$ time. In total we solve at most α matching problems in $O(\alpha^4)$ time. Since $\alpha \geq n/2$, this clearly dominates the time it take to sort the $O(n^2)$ edges. The following theorem summarizes the result of this section.

▶ **Theorem 8.** *The algorithm described above computes an optimal assignment for the case of $c = 2$, $|e| \leq 2$, and runs in $O(\alpha^4)$ time.*

In the special case where $\alpha = n/c = n/2$ the optimal solution corresponds to a maximum perfect matching in G. Indeed, for $x = 0$ we have $\alpha_x = 0$, $y_2 = n - \alpha_x - \alpha + x = n - 0 - \alpha = n/2$, and $y_1 = \alpha - x - y_2 = 0$. So for this value of x there are no dummy vertices in Λ' and an optimal assignment is given by a maximum matching in G. There is no need to try other values of x.

3.2 A Bi-Criteria Approximation Algorithm

We describe an approximation algorithm that applies to *a general instance* of the problem. Let c denote the maximum cluster size and α the number of clusters as before, and let $M = \max_{e \in E} |e|$. We assume that the algorithm has an estimate β of T_{OPT}, the optimal average service time with α clusters, such that $T_{OPT} \leq \beta < 2T_{OPT}$. In case such an estimate is not available, we can run the algorithm with $\beta = M/2^i$ for $i = 0, 1 \ldots, \lfloor \log M \rfloor$. Since $1 \leq T_{OPT} \leq M$, one of these values of β must be in the required range. Our approximation algorithm is bi-criteria, it computes a clustering with an average service time $O(T_{OPT} \log M \log c)$ and $O(\alpha \log M \log c)$ clusters.

Our algorithm adds a single cluster per iteration. At each iteration we compute a *score* for each tentative cluster D, denoted by $\phi(D)$, and we pick the cluster of maximum score. To compute $\phi(D)$ for each cluster D, we maintain for each edge e, the subset

$A(e)$ of the uncovered (by clusters picked at previous iterations) vertices of e. For each edge e we compute the fraction of e covered by D. We define this fraction to be *large* if it is at least $\frac{1}{4\beta}$. The score $\phi(D)$, is the weighted sum of the large fractions that D covers. That is, $\phi(D) = \sum_e w(e)|A(e) \cap D|/|A(e)|$, where the sum is over all e such that $|A(e) \cap D|/|A(e)| \geq \frac{1}{4\beta}$.

We prove that after a *phase* consisting of at most $8\alpha(\log M + 1)$ iterations we completely cover edges of total weight at least a $1/4$ (using $8\alpha(\log M + 1)$ clusters), where each edge which is completely covered is covered by $\leq 8T_{OPT}(\log M + 1)$ clusters. The optimal service time of the remaining (not completely covered) edges (of total weight at most $3/4$) is at most $4/3T_{OPT}$. In the next phase we apply the same procedure to these remaining edges with their new value of T_{OPT} (and estimate β). By the same argument, in the next phase we cover edges whose weights sum to $1/4$ of the total weight of the remaining edges (that were not covered in the first phase) using $\leq 8 \cdot (4/3T_{OPT})(\log M + 1)$ clusters. It follows that each phase adds $O(\alpha \log M)$ clusters and increases the average service time by $O(T_{OPT} \log M)$.

After $O(\log c)$ phases the leftover uncovered edges are of total weight $\leq 1/c$. We cover these remaining $1/c$ fraction of the edges arbitrarily using at most $\lceil n/c \rceil \leq \alpha$ additional clusters including all functions. The following theorem summarizes our result.

▶ **Theorem 9.** *The algorithm described above computes a solution with an average service time $O(T_{OPT} \log M \log c)$ that uses $O(\alpha \log M \log c)$ clusters.*

The running time of our algorithm is exponential in c since we have to compute the scores of all (unused) clusters of size c in each iteration.

3.3 Directed Hypergraphs

In some applications demands have to be served according to a specific order. In this case our input is a *directed* hypergraph meaning that each edge is an *ordered* tuple of vertices. We show how to use our approximation algorithm for graphs (all edges are of size 2 and unordered) specified in Theorem 1 to obtain a clustering with small average service time for directed hypergraphs. We only consider the case where $\alpha = n/c$ (disjoint clusters).

Recall that in the directed case which we consider here $P(e)$ is a sequence of clusters (possibly with repetitions) that cover the vertices of e in order consistent with the order of e. The service time $t(e)$ is the length of the sequence $P(e)$.

For an ordered hyperedge $e = (v_1, v_2, \ldots, v_k)$ let $U(e)$ be the set of the $k-1$ edges $\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\}$. We have the following lemma.

▶ **Lemma 10.** *Let $\alpha = n/c$ and let B be a clustering of the vertices of a directed hypergraph H into α disjoint clusters. Let $t(e)$ be the service time for serving an ordered edge e by B. Consider the set $U(e)$ of $|e|-1$ (unordered) edges, each equals to a consecutive pair of vertices in e as defined above. Then the total service time of serving $U(e)$ by B is $t' = t(e) + |e| - 2$.*

Consider for example the edge $e = (8, 2, 1, 7, 3)$ and the clusters $B(1) = \{1, 2, 3\}, B(2) = \{4, 5, 6\}, B(3) = \{7, 8, 9\}$. With this clustering, the edge e must be covered first by cluster 3 (to cover vertex 8), then cluster 1 (covering vertices 2 and 1), then cluster 3 again (to cover vertex 7), and finally cluster 1 again (to cover vertex 3), so $t(e) = 4$. The set $U(e)$ consists of the four edges $\{8, 2\}$, $\{2, 1\}$, $\{1, 7\}$, $\{7, 3\}$. Each of these edges but $\{2, 1\}$ requires two clusters to be covered for a total of 7 which is indeed $t(e) + |e| - 2$.

Lemma 10 suggests the following reduction from the ordered problem to an unordered problem in which $|e| = 2$ for all e.

Given a directed hypegraph $H = (V, E)$ we construct a graph $G = (V, E')$ on the same vertex set V and with $E' = \cup_{e \in E} U(e)$. We set the weight of an edge $e' \in U(e)$ to be $w(e') = w(e)/(W-1)$ where $W = \sum_{e \in E} w(e)|e|$. Note that $W - 1 = \sum_{e \in E} w(e)(|e| - 1)$ is a normalization factor that makes the weights of the edges in U sum to 1.[5]

Consider a clustering B of n vertices into $\alpha = n/c$ clusters. Then by Lemma 10

$$\sum_{e \in E'} w(e)t(e) = \sum_{e \in E} \frac{w(e)}{W-1}(t(e) + |e| - 2).$$

So if T' is the average service time of G by B and T is the average service time of H by B then $T' = (T + W - 2)/(W - 1)$. By rearranging we get that $T = (W-1)T' - W + 2$.

Since all edges in G are of size 2 then we can apply to G the approximation algorithm of Theorem 1 which guarantees an approximation ratio of $\frac{2c+1}{c+2}$. The resulting clustering has the following guarantee for H.

▶ **Theorem 11.** *The clustering obtain for G by the algorithm of Theorem 1 has an average service time $T \leq \frac{2c+1}{c+2} T_{OPT(H)}$ when applied to serve the hyperedges of H, where $T_{OPT(H)}$ is the smallest possible average service time for H.*

Proof. The average service time T of the obtained clustering satisfies that $T \leq (W-1) \cdot \frac{2c+1}{c+2} \cdot T_{OPT(G)} - W + 2$, where $T_{OPT(G)}$ is the average service time of the optimal solution to G. The optimal solution of H induces a solution of G with a service time T' such that $T' = (T_{OPT(H)} + W - 2)/(W-1)$. It follows that $T_{OPT(G)} \leq T' = \frac{T_{OPT(H)} + W - 2}{W-1}$. By substituting the last equation into the previous we get $T \leq \frac{2c+1}{c+2} T_{OPT(H)}$. ◀

4 Conclusions and Open Problems

We introduce the problem of clustering vertices of a weighted hypergraph to minimize the average service time of its edges. For disjoint clusters we described a natural greedy algorithm and analyzed it in two cases: when edges are of size 2, and when edges are of size at most 3 and clusters are of size 3. The latter analysis is subtle and uses an interesting set of invariants. This greedy algorithm can be naturally generalized for larger edges and cluster sizes. Our analysis, however, gets complicated and the number of cases seems to get out of control. Is there an alternative simpler way to analyze such a generalization? One can also try to deal with larger clusters via an hierarchical approach: First cluster the vertices into smaller clusters, then contract these small clusters, and cluster the contracted hypergraph into small clusters again. Finding a way to analyze this hierarchical approach is an open problem. An interesting problem for an experimental research is to compare the performance of the hierarchical and non-hierarchical approaches on some interesting data sets. To conclude, finding a general algorithm for disjoint clusters which is amenable to analysis, and has good approximation ratio is a very interesting challenge (or alternatively proving inapproximability results). Such an algorithm, if practical, will find numerous applications. For overlapping clusters we gave a bicriteria approximation algorithm. A natural open question is to find an algorithm with a guaranteed approximation ratio with respect to the optimal clustering with α clusters that does not require more than α clusters.

[5] Notice that in the constructed unordered instance U we may have identical edges.

References

1 Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer communications*, 30(14):2826–2841, 2007.

2 Sameer Agarwal, Jongwoo Lim, Lihi Zelnik-Manor, Pietro Perona, David J. Kriegman, and Serge J. Belongie. Beyond pairwise clustering. In *IEEE CVPR*, 2005.

3 Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

4 Reid Andersen, David F. Gleich, and Vahab Mirrokni. Overlapping clusters for distributed computation. In *ACM Web search and data mining*, 2012.

5 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

6 Rafael Bru, Francisco Pedroche, and Daniel B. Szyld. Additive Schwarz iterations for Markov chains. *SIAM Journal on Matrix Analysis and Applications*, 27(2):445–458, 2005.

7 Rafael Bru, Francisco Pedroche, and Daniel B. Szyld. Cálculo del vector PageRank de Google mediante el método aditivo de Schwarz. In *Congreso de Métodos Numéricos en Ingeniería*, 2005.

8 Samuel Rota Bulò and Marcello Pelillo. A game-theoretic approach to hypergraph clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1312–1327, 2013.

9 Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

10 Rami Cohen, Liane Lewin-Eytan, Joseph Naor, and Danny Raz. Near optimal placement of virtual network functions. In *IEEE Infocom*, 2015.

11 A. J. Cole and D. Wishart. An improved algorithm for the Jardine-Sibson method of generating overlapping clusters. *The Computer Journal*, 13(2):156–163, 1970.

12 David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2):224–227, 1979.

13 Ran Duan. A simpler scaling algorithm for weighted matching in general graphs. *CoRR*, abs/1411.1919, 2014. URL: `http://arxiv.org/abs/1411.1919`.

14 Joseph C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104, 1974.

15 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

16 Vladimir Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD explorations newsletter*, 4(1):65–75, 2002.

17 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

18 Andreas Frommer and Daniel B. Szyld. Weighted max norms, splittings, and overlapping additive Schwarz iterations. *Numerische Mathematik*, 83(2):259–278, 1999.

19 Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *ACM-SIAM SODA*, 1990.

20 David S. Johnson. Approximation algorithms for combinatorial problems. In *ACM symposium on Theory of computing*, 1973.

21 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations, IBM Thomas J. Watson Research Center*, 1972.

22 Marius Leordeanu and Cristian Sminchisescu. Efficient hypergraph clustering. In *AISTATS*, 2012.

23 László Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.

24 Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert E. Tarjan. Clustering social networks. In *Algorithms and Models for the Web-Graph*, pages 56–67. Springer, 2007.

25 Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14:849–856, 2002.

26 Matthias Peiser et al. Allergic contact dermatitis: epidemiology, molecular mechanisms, in vitro methods and regulatory aspects. *Cellular and Molecular Life Sciences*, 69(5):763–781, 2012.

27 Ori Rottenstreich, Isaac Keslassy, Yoram Revah, and Aviran Kadosh. Minimizing delay in network function virtualization with shared pipelines. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):156–169, 2017.

28 Amnon Shashua, Ron Zass, and Tamir Hazan. Multi-way clustering using super-symmetric non-negative tensor factorization. In *ECCV*, 2006.

29 Daniel A. Spielmat and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *IEEE Foundations of Computer Science*, 1996.

30 Luca Trevisan. Non-approximability results for optimization problems on bounded degree instances. In *ACM symposium on Theory of computing*, 2001.

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

32 Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *NIPS*, 2006.