Learning Graphical Models over partial k-trees

Mukund Narasimhan *and* **Jeff Bilmes**
Dept. of Electrical Engineering
University of Washington
Seattle, WA 98195

UWEE Technical Report
Number UWEETR-2006-0001
January 2006

Department of Electrical Engineering
University of Washington
Box 352500
Seattle, Washington 98195-2500
PHN: (206) 543-2150
FAX: (206) 543-3842
URL: http://www.ee.washington.edu

Learning Graphical Models over partial k-trees

Mukund Narasimhan and **Jeff Bilmes**
Dept. of Electrical Engineering
University of Washington
Seattle, WA 98195

University of Washington, Dept. of EE, UWEETR-2006-0001
January 2006

Abstract

We show that Queyranne's algorithm for finding a non-trivial minimizer of a symmetric submodular function can be used as a clustering algorithm. For submodular clustering criterion, such as graph-cut or minimum description length based criteria, we can find the optimal clustering for 2 clusters, and near optimal clusterings for more than 2 clusters. Due to results by Rizzi and Nagamochi and Ibaraki, the algorithm can also be used for more general functions, allowing it to be used for various other criteria such as the single-linkage criterion.

The goal of data clustering is to find a partition of the data (feature vectors, points in a metric space, random variables, pixels in images, etc.) into sets of similar items. This is perhaps one of the most common and widely used unsupervised data analysis technique. Clustering often appears under different names, such as mixture modeling, dimensionality reduction, voronoi tessellation, vector quantization, phylogenetic tree analysis, and image segmentation. Broadly speaking, there are two reasons for clustering data. The first is for general data analysis and exploration. In this case, the clustering is used to analyze, visualize, and represent high-dimensional, and high volumes of data to reveal attributes of the data that are not immediately obvious. The second reason is to produce an intermediate representation that is used to generate other results. For example, clustering is often used to determine probabilistic mixture models (such as mixtures of Gaussians) which might then be used for classification.

Because there are so many different and varied applications for clustering, there are many different algorithms, and many different criteria for clustering. The different clustering techniques can be categorized into two broad classes. The first consists of techniques that specify an objective function for clustering, and an algorithm for optimizing (perhaps heuristically) this objective function. This includes algorithms such as the many variants of K-Means (or Lloyd's algorithm) [30, 29, 2, 11], Spectral clustering (which was shown to approximately minimize a balanced cut criterion) [31, 22, 35], and various graph based algorithms [24, 1, 6, 12, 35, 18, 23]. The second class of techniques generate the clustering as the result of an algorithmic process, without a clearly defined objective function. We include in this class algorithms that are used primarily for visualization of the data. Examples include multidimensional scaling (perhaps coupled with matrix based algorithms) [8, 5] and self-organizing maps [25, 19].

In this chapter, we restrict our attention to only the first class. We show that some commonly used clustering objective functions are either submodular functions, or reducible to submodular functions. As a result, Queyranne's algorithm for minimizing symmetric submodular functions can be used for finding the optimal clustering into two clusters. Further we can find clusterings which are within a constant factor of optimal when more than two clusters are required. For some of the submodular criteria considered in this chapter, there are alternate algorithms which can be used to determine the optimal clustering (for two clusters). In fact, as each of these alternate algorithms work only on a specific objective function (instead of the entire class of submodular functions), they can be much faster than Queyranne's algorithm. For example, when the criterion is an undirected or directed graph cut criterion, we can use either a flow based algorithm [15, 13], or other dedicated graph cut algorithms [17, 36] to find the optimal solutions. Similarly when the objective function is the single linkage criterion, then we might use MST algorithm [9, 20] for finding the optimal solution. These algorithms are faster than Queyranne's algorithm as they are able to exploit problem specific information. However none of these algorithms can, for example, optimize a (positive linear) combination of of the different criteria. However, since the class of submodular functions is closed under positive linear combinations (and a number of other operations), Queyranne's algorithm can be used for this purpose.

In this chapter, we also present other submodular criteria for which algorithms which guarantee optimality (even for two clusters) were not previously known. For example, we show that Minimum Description Length criterion [26] is submodular, and as a result, resolve an open problem of finding the optimal 2-clustering with respect to this objective. Besides the optimality result for the MDL criterion, the chief contribution of this chapter is to show that the *same algorithm* can be used to optimize a number of widely used criteria, and hence can be used for many application specific criterion derived from combinations of simpler submodular functions for which efficient algorithm are not known.

1 Background and Notation

A *clustering* of a finite set S is a partition $\{S_1, S_2, \ldots, S_k\}$ of S. Therefore, $S = \cup_{i=1}^{k} S_i$, and $S_i \cap S_j = \phi$ if and only if $i \neq j$ (in particular $S_i \neq \phi$ for all $1 \leq i \leq k$). We will call the individual elements of the partition the clusters of the partition. If there are k clusters in the partition, then we say that the partition is a k-clustering. Let $\mathcal{C}_k(S)$ be the set of all k-clusterings for $1 \leq k \leq |S|$. Formally, the clustering problem can be described as follows.

Problem 1 (Clustering)**.** *Given a finite set S, and a clustering criterion $J_k : \mathcal{C}_k(S) \rightarrow \mathbb{R}$, find*

$$\operatorname*{argmin}_{\{A_1, A_2, \ldots, A_k\} \in \mathcal{C}_k(S)} J_k\left(\{A_1, A_2, \ldots, A_k\}\right)$$

It is shown in [20] that $|\mathcal{C}_k(S)|$, the number of k-clusteringss for a base set S of size $|S| = n > k$ is approximately $k^n/k!$. There are two consequences of this:

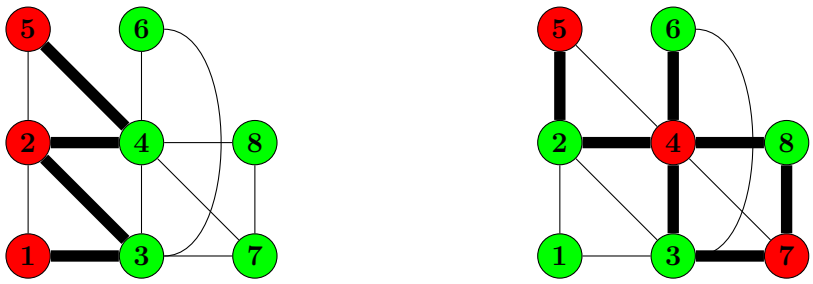

Figure 1: The Graph-Cut criterion: The cost of the clustering is the sum of the weights of the edges that are between elements of different clusters. If all edges have weight 1, then $J_2^{\mathsf{gc}}(\{1,2,5\},\{3,4,6,7,8\}) = 4$ and $J_2^{\mathsf{gc}}(\{4,5,7\},\{1,2,3,6,8\}) = 7$.

1. The objective $J_k : \mathcal{C}_k(S) \to \mathbb{R}$ cannot be represented as a table of values for every element of $\mathcal{C}_k(S)$. Therefore, J_k must have some structure allowing for efficient representation.

2. Exhaustive enumeration to select the optimal partition is not (efficiently) possible. Hence algorithms that determine the optimal solution must exploit the structure of J_k.

The structure of J_k is often either derived from a probabilistic model, or from a pairwise similarity or distance measure. We now list some criteria which are submodular, and hence can be optimized using Queyranne's algorithm.

1.1 The Graph-Cut Criterion

Suppose that S is the vertex set of a graph $G = (S, E)$, and let $w : E \to \mathbb{R}^+$ assign weights to edges of G. The weight $w(\{x, y\})$ is assumed to be a measure of similarity between x and y. Hence if x and y are very similar, then $w(\{x, y\})$ is large, while $w(\{x, y\})$ is small if the objects are very dissimilar. If the edge $\{x, y\}$ is not present in the graph(so $\{x, y\} \notin E$), the it is assumed that $w(\{x, y\}) = 0$. The graph-cut criterion is given by

$$J_k^{\mathsf{gc}}(\{A_1, A_2, \ldots, A_k\}) = \sum_{\substack{\{a_i,a_j\}\in E \\ a_i\in A_i, a_j\in A_j, A_i\neq A_j}} w(\{a_i, a_j\})$$

In other words, the cost of the clustering is the weights of the edges *broken* by the partition. Figure 1 illustrates this criterion for $k = 2$. The emphasized edges are the edges that do not have both ends in the same partition, and hence are the edges that are *broken* by the clustering.

Let us denote by $E(A)$ the set of edges that have (at least) one end in A. Let $|E(A)|$ be the sum of the weights of the edges in $E(A)$. The edges that are broken by the partition $\{A, S \setminus A\}$ are the edges that are adjacent to both A and $S \setminus A$, namely $E(A) \cap E(S \setminus A)$. As was pointed out in Example **??**, the function

$$\gamma_{\mathsf{gc}}(A) = |E(A)| = \sum_{\substack{\{a,b\}\in E \\ a\in A \text{ or } b\in A}} w(\{a, b\})$$

is an increasing submodular function. Therefore the function $f : 2^S \to \mathbb{R}$ given by

$$f(A) = \frac{\gamma_{\mathsf{gc}}(A) + \gamma_{\mathsf{gc}}(S \setminus A)}{2} - \gamma_{\mathsf{gc}}(S) = \sum_{\substack{\{a,b\}\in E \\ a\in A, b\in S\setminus A}} w(\{a, b\})$$

is a symmetric submodular function by Lemma **??** and because $\gamma(S)$ is a constant. Observe that this is exactly the number of edges broken by the cut. Therefore,

$$J_2^{\mathsf{sc}}(\{A, S \setminus A\}) = f(A)$$

and therefore we can use Queyranne's algorithm to find $\mathsf{argmin}\, J_2^{\mathsf{gc}}$. Note that there are alternate algorithms that can also compute the minimum of this function such as those described in [15, 13, 17, 36]).

However, the problem of finding the minimum of J_k^{gc} for $k > 2$ is NP-complete in general (though it can be solved in time polynomial in $|S|$, and exponential in k, yielding a tractable algorithm for small values of k). Observe that

$$J_k^{\mathsf{sc}}\left(\{A_1, A_2, \ldots, A_k\}\right) = \frac{\sum_{i=1}^{k} \gamma_{\mathsf{gc}}(A_i)}{2} - \gamma_{\mathsf{gc}}(S) = \sum_{\substack{\{a_i,a_j\}\in E \\ a_i\in A_i, a_j\in A_j, A_i\neq A_j}} w(\{a_i, a_j\})$$

Observe that finding $\mathsf{argmin} J_k$ is equivalent to finding

$$\underset{\{A_1,A_2,\ldots,A_k\}}{\mathsf{argmin}} \sum_{i=1}^{k} \gamma_{\mathsf{gc}}(A_i)$$

While finding the solution to this problem is NP-complete, Queyranne's algorithm yields an approximation algorithm for this problem, with approximation guarantee $2(1 - 1/k)$ for $k > 2$.

1.2 The Feature-Similarity Criterion

Let F be a collection of features. Each object $s \in S$ is associated with some subset of these features. In this case, we would like to define similarity between objects (or sets of objects) in terms of the number of shared features. In other words, two objects $s_1, s_2 \in S$ are very similar if s_1 and s_2 have a large set of features in common. We can construct a bipartite graph $G = (S, F, E)$, so that there is an edge $\{s, f\} \in E$ if the object s has the feature f. Let $\Gamma : S \to 2^F$ map objects to the set of features that that the objects are associated with:

$$\Gamma(s) = \{\, f \in F : \{s, f\} \in E\}$$

We can extend this function to subsets of S:

$$\Gamma(A) = \bigcup_{s\in A} \Gamma(s) = \{\, f \in F : \exists\, \{s, f\} \in E \text{ for some } s \in A\}$$

So for the bipartite graph shown in Figure 2, $\Gamma(\{1, 2, 5\}) = \{A, B, D, E\}$, and $\Gamma(\{3, 4, 6\}) = \{B, C, D, E, F\}$. We may associate weights with features, and so we define

$$\gamma_{\mathsf{fs}}(A) = \sum_{f\in\Gamma(A)} w(f)$$

Like γ_{gc} defined in the previous section, γ_{fs} is also an increasing submodular function.

We measure the degree of similarity of two subsets of S by the (weighted) number of features common to both sets. In other words, if $\{A, S \setminus A\}$ is a partition, then the weight of the shared features is given by

$$f(A) = \frac{\gamma_{\mathsf{fs}}(A) + \gamma_{\mathsf{fs}}(S \setminus A)}{2} - \gamma_{\mathsf{fs}}(S) = \sum_{\{\, f\in F : \{a,f\}\in E, \{b,f\}\in E, a\in A, b\in S\setminus A\}} w(f)$$

The function $f : 2^S \to \mathbb{R}$ a symmetric submodular function by Lemma **??** and because $\gamma_{\mathsf{fs}}(S)$ is a constant. Therefore,

$$J_2^{\mathsf{fs}}\left(\{A, S \setminus A\}\right) = f(A) = \frac{\gamma_{\mathsf{fs}}(A) + \gamma_{\mathsf{fs}}(S \setminus A)}{2} - \gamma_{\mathsf{fs}}(S)$$

can be minimimized using Queyranne's algorithm. For $k > 2$, we take

$$J_k^{\mathsf{fs}}\left(\{A_1, A_2, \ldots, A_k\}\right) = \sum_{i=1}^{k} \gamma_{\mathsf{fs}}(A_i)$$

and finding $\mathsf{argmin}\, J_k$ is NP-complete for $k > 2$. As in the previous section, Queyranne's algorithm yields an approximation algorithm for this problem, with approximation guarantee $2(1 - 1/k)$ for $k > 2$.

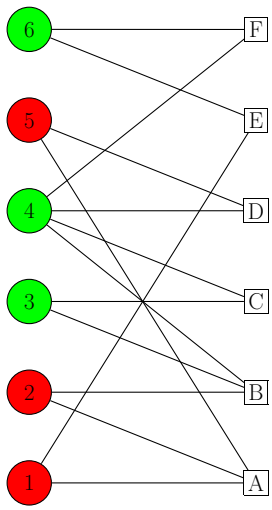

Figure 2: The Bipartite graph for the Feature-Similarity criterion. The cost of a partition is the weight of the common features. In this case, $\Gamma(\{1,2,5\}) = \{A,B,D,E\}$, and $\Gamma(3,4,6) = \{B,C,D,E,F\}$. Therefore, the set of common features is $\{B,D,E\}$, and so the cost of the partition is $w(B) + w(D) + w(E)$.

1.3 The Minimum Description Length Criterion

Supppose that S is a collection of random variables for which we have a (generative) joint probability model. Since we have the joint probabilities of all subsets of the random variables, the entropy of any subset of S is well defined. The expected coding (or description) length of any collection $T \subseteq S$ of random variables using an optimal coding scheme is known to be the Shannon entropy of the set of random variables, denoted $H(T)$. The partition $\{S_1, S_2\}$ of S that minimizes the coding length is therefore $\arg\min_{\{S_1,S_2\}\in\mathcal{C}_2(S)} H(S_1) + H(S_2)$. Now,

$$\begin{aligned}\operatorname*{arg\,min}_{\{S_1,S_2\}\in\mathcal{C}_2(S)} H(S_1) + H(S_2) &= \operatorname*{arg\,min}_{\{S_1,S_2\}\in\mathcal{C}_2(S)} H(S_1) + H(S_2) - H(S) \\ &= \operatorname*{arg\,min}_{\{S_1,S_2\}\in\mathcal{C}_2(S)} I(S_1;S_2)\end{aligned}$$

where $I(S_1;S_2)$ is the mutual information between S_1 and S_2 because $S_1 \cup S_2 = S$ for all $\{S_1,S_2\} \in \mathcal{C}_2(S)$, Therefore, the problem of partitioning S into two parts to minimize the description length is equivalent to partitioning S into two parts to minimize the mutual information between the parts. It was shown in Chapter 2 that H is an increasing submodular function (like γ_{gc} and γ_{fs} defined in the previous sections), and the function $f : 2^S \rightarrow \mathbb{R}$ defined by $f(T) = I(T; S \setminus T)$ is symmetric and submodular. Therefore,

$$J_2(\{A, S \setminus A\}) = I(A; S \setminus A) = H(A) + H(S \setminus A) - H(S)$$

Clearly the minima of this function correspond to partitions that minimize the mutual information between the parts. Therefore, the problem of partitioning in order to minimize the mutual information between the parts can be reduced to a symmetric submodular minimization problem, which can be solved using Queyranne's algorithm in time $O(|S|^3)$ assuming oracle queries to a mutual information oracle. While implementing such a mutual information oracle is not trivial, for many realistic applications (including one we consider in this paper), the cost of computing a mutual information query is bounded above by the size of the data set, and so the entire algorithm is polynomial in the size of the data set.

For $k > 2$, we can similarly define

$$J_k(\{A_1, A_2, \ldots, A_k\}) = \sum_{i=1}^{k} H(A_i)$$

since $H(A_i)$ is the description length of the cluster A_i. While this problem is also NP-complete, Queyranne's algorithm yields a $2(1 - 1/k)$ factor approximation for it.

1.4 The general setup

We will assume the following setup. We are given an increasing submodular function $F : 2^S \to \mathbb{R}$. We can use this function to create a symmetric submodular function $f : 2^S \to \mathbb{R}$ given by

$$f(X) = F(X) + F(S \setminus X) - F(S)$$

As this is symmetric and submodular, this function can be minimized using Queyranne's algorithm, and hence we can find the optimal 2-parititon that minimizes the clustering criterion

$$J_k(\{A_1, A_2, \ldots, A_k\}) = \sum_{i=1}^{k} F(A_i)$$

Observe that the function $f'(X) = F(X) + F(S \setminus X)$ could have just as easily been used as the symmetric submodular function to minimize as the addition of the constant $F(S)$ cannot alter the optimal solution. However, we prefer to use f for two reasons. First, it is a direct generalization of the graph cut function. To see the second reason, observe that for any $X \subseteq S$, and for any symmetric submodular function $f'' : 2^S \to \mathbb{R}$, we must have

$$2f''(X) = f''(X) + f''(S \setminus X) \geq f''(\phi) + f''(S) = 2f''(S)$$

Therefore setting $f''(S) = 0$ acts as a kind of normalization. It further ensures that f'' is always non-negative. Therefore, we will prefer to use f instead of f'. We use this normalization property of f' in the construction of the Gomory-Hu tree in the next section.

2 Gomory-Hu Trees and Approximations

It was shown in Section **??** that Queyranne's algorithm returns an (approximate) minimizer of an (approximately) symmetric submodular function. Therefore, we may use Queyranne's algorithm to find the optimal 2-partition for a symmetric submodular criterion. In this section, we consider the problem of finding k-clustering. Because the problem of finding a k-partition is NP-complete for the graph-cut function [34], which is a symmetric and submodular function, the more general problem of finding a k-partition for symmetric submodular functions is NP-complete.

In [16], Gomory and Hu showed how to construct a tree which essentially "encodes" all the solutions to all optimization problems of the form

$$\min_{X \in \mathcal{S}(x,y)} f(X)$$

when $f(X)$ is the Graph-Cut criterion, x, y are distinct elements of S, and $\mathcal{S}(x, y)$ is the collection of sets that contain x, but do not contain y:

$$\mathcal{S}(x, y) = \{\, X \subseteq S : x \in S, y \notin S\}$$

In other words, $\mathcal{S}(x, y)$ is the collection of subsets of S that seperate x from y. The Gomory-Hu tree is a spanning tree on the vertex set S, whose edges have weights associated with them. Since it is a tree, the removal of any edge disconnects it, and hence produces a partition of S into two parts. The key property of the tree is this: For every pair of vertices $s, t \in S$, there is a unique path in the tree between s and t. Let $e_1, e_2, \ldots, e_m$ be the set of edges on this path, and let $w(e_1), w(e_2), \ldots, w(e_m)$ be the corresponding weights. If e_k is the edge with the smallest weight, then the removal of e_k produces a partition $\{S_s, S_t\}$ where $s \in S_s$ and $t \in S_t$, and S_s is the (approximate) solution to $\mathsf{argmin}_{X \in \mathcal{S}(s,t)} f(X)$.

Goemans and Ramakrishnan [14] observe that such a tree (also called a cut-equivalent tree) exists for every symmetric submodular function. It is also observed in [14] that a Gomory-Hu tree for any symmetric submodular function can be constructed by $|S| - 1$ calls to a procedure that finds solutions to problems of the form $\mathsf{argmin}_{X \in \mathcal{S}(s,t)} f(X)$.

While the Gomory-Hu tree has many applications, we shall be interested in using it to produce approximately optimal clusterings. Saran and Vazirani [34] showed that by deleting the $k - 1$ lightest weight edges of the Gomory-Hu tree for the graph-cut function produced a partition which was within a factor of $2(1 - 1/k)$ of the optimal partition with respect to the graph-cut criterion. We will show that this result also holds for the Gomory-Hu tree for arbitrary symmetric submodular functions. We will also use the Gomory-Hu tree to show that a recursive bisection algorithm yields the same approximation ratio.

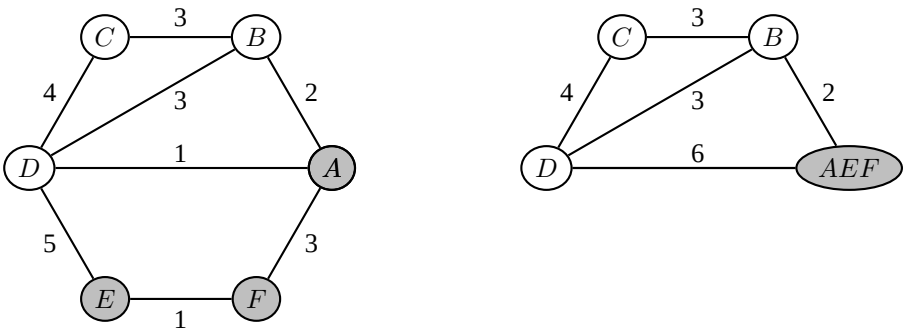

Figure 3: The graph cut function for any graph $G = (V, E)$ is a symmetric and submodular function. Merging any subset of the vertices, for example $\{A, E, F\}$, to form a single node yields a new graph, and hence a new graph cut function.

2.1 Construction of a Gomory-Hu tree for symmetric submodular functions

Queyranne's algorithm works by repeatedly identifying pendent-pairs, and then merging the pair of elements identified. A pendent-pair (t, u), defined in Definition **??**, satisfies

$$\{u\} \in \operatorname*{argmin}_{U \in \mathcal{S}(u,t)} f(U)$$

In other words, if (t, u) is a pendent-pair, then either $\{u\}$ minimizes f (over all all non-trivial subsets of S) or the minimizer

$$\operatorname*{argmin}_{X \in \mathcal{S}(x,y)} f(X)$$

can be chosen so that it does not separate u from t. For the second case, we can merge u and t, treating them as a single element, and the minimum for this merged function yields the minimum for the original function.

The idea of merging elements to generate a function on a smaller set of elements is also central for the construction of the Gomory-Hu tree. For this application, we will need to merge arbitrary sets (i.e., more than a pair of elements) into a single element. As for the case of merging pairs, the operation of merging sets results in a symmetric submodular function defined on a smaller set.

Lemma 1. *Suppose that $f : 2^S \to \mathbb{R}$ is a symmetric submodular function. Let $A \subseteq S$. By merging the elements of A into a single element m_A we get the set $S_A = (S \setminus A) \cup \{m_A\}$, and the function $f_A : S_A \to \mathbb{R}$ defined by*

$$f_A(X) = \begin{cases} f(X) & \text{if } X \subseteq (S \setminus A) \text{ (and so } m_A \notin X\text{)} \\ f((X - m_A) \cup A) & \text{if } m_A \in X \end{cases}$$

is symmetric and submodular.

As an example, consider the graph shown in Figure 3. Merging the set of vertices $\{A, E, F\}$ results in a new graph with fewer vertices. The graph cut function on this reduced vertex set is also symmetric and submodular.

The following lemma was presented in [16] for graph-cut functions. The proof generalizes immediately to symmetric submodular functions.

Lemma 2. *Suppose that $a, b \in S$ are distinct elements, and*

$$A \in \operatorname*{argmin}_{X \in \mathcal{S}(a,b)} f(X)$$

If $c, d \in A$ are distinct elements, then we can find an element

$$C \in \operatorname*{argmin}_{X \in \mathcal{S}(c,d)} f(X)$$

so that either $S \setminus A \subseteq C$ or $S \setminus A \subseteq S \setminus C$.

Proof. Let us denote by $\overline{A}$ the set $S \setminus A$. By the symmetry of f, we must have

$$\overline{A} \in \operatorname*{argmin}_{X \in \mathcal{S}(b,a)} f(X)$$

In particular, $b \in \overline{A}$, and therefore either $b \in \overline{A} \cap C$ or $b \in \overline{A} \cap \overline{C}$. Let us consider both these cases.

1. Suppose $b \in \overline{A} \cap C$. Because $c \in C$ we must have $c \in \overline{A} \cup C$. Since $d \notin C$ and $d \notin \overline{A}$, we must have $\overline{A} \cup C \in \mathcal{S}(c,d)$. Therefore, by the minimality of C we have

   $$f(C) \leq f(\overline{A} \cup C)$$

   Further, $b \in \overline{A} \cap C$ and $a \notin \overline{A} \cap C$. Therefore $\overline{A} \cap C \in \mathcal{S}(a,b)$ and hence

   $$f(\overline{A}) = f(A) \leq f(\overline{A} \cap C)$$

   Adding these two inequalities, we get

   $$f(\overline{A}) + f(C) \leq f(\overline{A} \cap C) + f(\overline{A} \cup C)$$

   By the submodularity of f, we have

   $$f(\overline{A}) + f(C) \geq f(\overline{A} \cap C) + f(\overline{A} \cap B)$$

   Therefore, equality must hold. In particular, it follows that

   $$f(C) = f(C \cup \overline{A})$$

   Hence the minimizer can be chosen so that it does not separate the elements of $\overline{A}$.

2. Suppose $b \in \overline{A} \cap \overline{C}$. Because $c \notin \overline{C}$ and $c \notin \overline{A}$, we must have $c \notin \overline{C} \cup \overline{A}$. Therefore, $\overline{C} \cup \overline{A} \in \mathcal{S}(d,c)$ and hence

   $$f(\overline{C}) \leq f(\overline{A} \cup \overline{C})$$

   Because $b \in \overline{A} \cap \overline{C}$, and because $a \notin \overline{A} \cap \overline{C}$ we have $\overline{A} \cap \overline{C} \in \mathcal{S}(a,b)$, and hence

   $$f(\overline{A}) \leq f(\overline{A} \cap \overline{C})$$

   Combining with the submodular inequality, we get $f(\overline{C}) = f(\overline{C} \cup \overline{A})$.

Therefore, the minimizer can be chosen to not separate the elements of $\overline{A}$ in this case as well. □

Therefore, if $A \in \operatorname{argmin}_{X \in \mathcal{S}(a,b)} f(X)$, then for any distinct $c, d \in A$, we can treat the elements in $\overline{A}$ as an atomic unit (by merging the elements of $\overline{A}$). We formalize this in the following corollary.

Corollary 3. *Suppose that $A \in \operatorname{argmin}_{X \in \mathcal{S}(a,b)} f(X)$, and $c, d \in A$ are distinct elements. If*

$$C' \in \operatorname*{argmin}_{X \in \mathcal{S}_{\overline{A}}(c,d)} f_{\overline{A}}(X)$$

then $C \in \operatorname{argmin}_{X \in \mathcal{S}(c,d)} f(X)$ where

$$C = \begin{cases} C & \text{if } m_{\overline{A}} \notin C' \\ (C - m_{\overline{A}}) \cup \overline{A} & \text{if } m_{\overline{A}} \in C' \end{cases}$$

This observation is used in Algorithm 1 for constructing the Gomory-Hu tree (also called the cut-tree). This observation, and Algorithm 1 was first given by [16] for the case of a graph cut function. However, the generalization to symmetric submodular functions is immediate.

Figure 4 displays the first few steps in the running of Algorithm 1 for the graph cut function for the graph shown.

Now, P_i is a partition of S for every $1 \leq i \leq |S|$, with P_1 consisting of a single partition with all the elements of S, and $P_{|S|}$ putting each element in a partition by itself. At each iteration, the pair (P_i, E_i) forms a tree. Therefore, after $|S|$ iterations, the edges are between partitions that consist of exactly one element each. By identifying the edges between partitions that consist of one element each with an edge between the (unique) element of the partitions, we get a tree with vertex set S. This is the Gomory-Hu tree for (S, f), and we denote it by (S, E). We now show some properties of this tree.

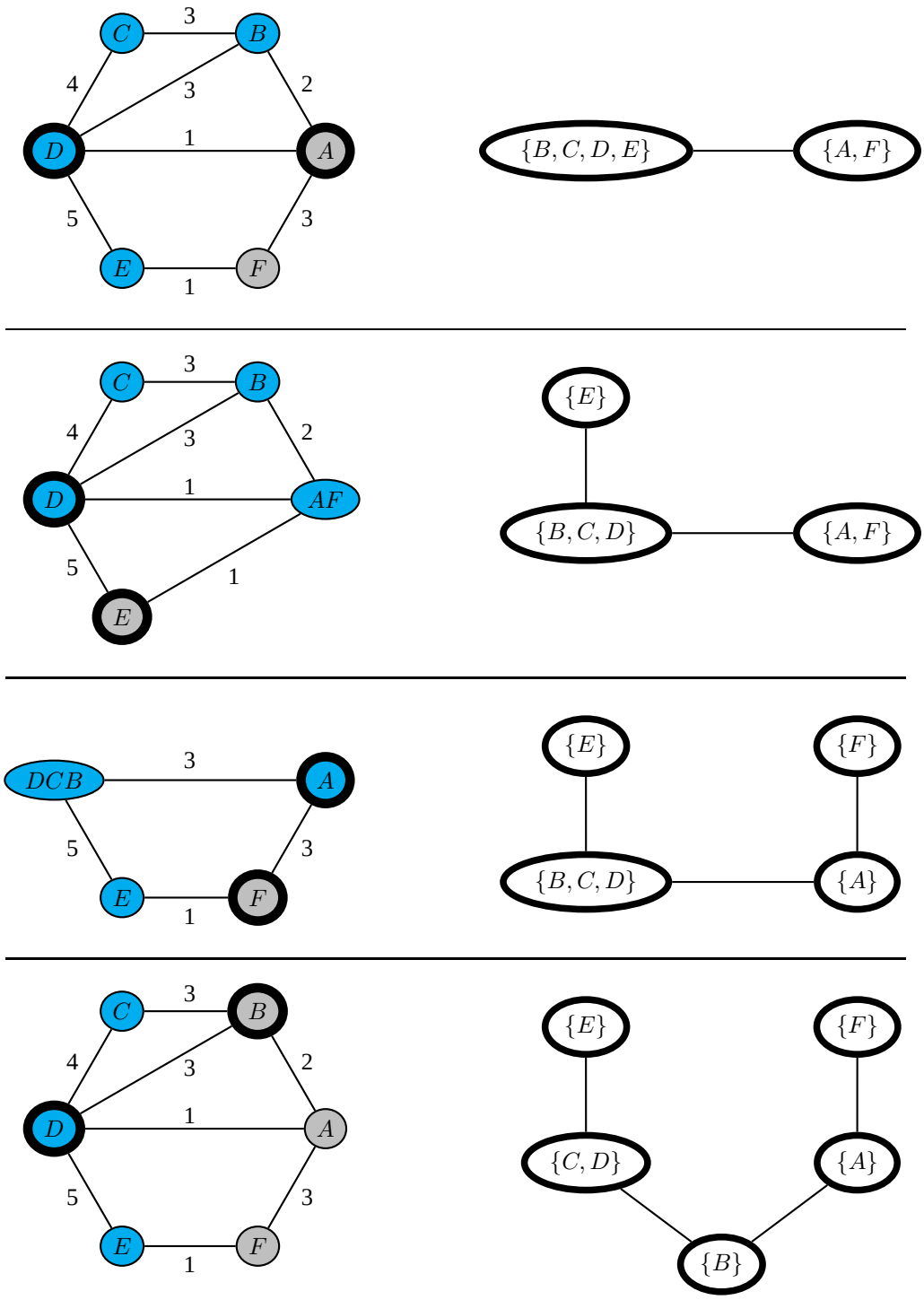

Figure 4: The first four steps in the running of Algorithm 1 for the graph cut function on the graph shown. The graph on the left represents the graph cut function f_i, while the tree (P_i, E_i) is shown on the right. The nodes a and b selected on line 5 of Algorithm 1 are shown with a thick border, and the cut (A, B) is shown in different colors.

Data: A set S and a submodular function f
Result: A Gomory-Hu tree (S, E) for f
$P_1 \leftarrow \{S\}$; $E_1 \leftarrow \phi$; /* P_1 is a partition with 1 part */
for $i \leftarrow 1 \ldots |S| - 1$ **do**
  /* Invariant : (P_i, E_i) is a tree */
  Pick some set (partition) $Q \in P_i$ with $|Q| \geq 2$;
  Pick distinct $a, b \in Q$;
  $S_i \leftarrow Q \cup \{m_R : R \in P_i, R \neq Q\}$; /* Merge all partitions other than Q /*
  Let $f_i : 2^{S_i} \rightarrow \mathbb{R}$ be the merged function on S_i;
  Find $A \in \mathsf{argmin}_{X \in \mathcal{S}(a,b)} f_i(A)$; /* Find the best partition of S_i */
  $A' \leftarrow Q \cap A$; $B' \leftarrow Q \setminus A'$; /* $\{A', B'\}$ is a partition of Q */
  Let $P_{i+1} \leftarrow (P_i \setminus Q) \cup \{A', B'\}$; /* P_{i+1} is a refinement of P_i */
  $E^{\mathsf{nq}}_{i+1} \leftarrow \{\{R, S\} \in E_i : \{R, S\} \in E_i, R, S \neq Q\}$; /* Edges not adjacent to Q */
  $E^{\mathsf{qa}}_{i+1} \leftarrow \{\{R, A'\} : \{R, Q\} \in E_i, m_R \in A\}$; /* Edges adjacent to A' side of Q */
  $E^{\mathsf{qb}}_{i+1} \leftarrow \{\{R, B'\} : \{R, Q\} \in E_i, m_R \notin A\}$; /* Edges adjacent to B' side of Q */
  $E_{i+1} \leftarrow E^{\mathsf{nq}}_{i+1} \cup E^{\mathsf{qa}}_{i+1} \cup E^{\mathsf{qb}}_{i+1} \cup \{\{A', B'\}\}$;
end
return $(S, E_{|S|})$;

Algorithm 1: GomoryHuTree

Lemma 4. *Suppose that the elements $a, b \in Q$ are selected at some iteration $1 \leq i \leq |S| - 1$. Let*

$$A'' = A' \cup \bigcup_{m_R \in A} R \qquad \text{and} \qquad B'' = B' \cup \bigcup_{m_R \in S_i \setminus A} R$$

where m_R is the node in S_i resulting from merging the partition $R \in P_i$. Then $\{A'', B''\}$ is a partition of S, and this partition is the optimal partition that separates a from b (for f). In other words

$$A'' \in \operatorname*{argmin}_{X \subseteq \mathcal{S}(a,b)} f(X)$$

Proof. This follows immediately from Corollary 3. □

We may associate weights $w : E \rightarrow \mathbb{R}$ with the edges of the Gomory-Hu tree (S, E). The removal of any edge $e \in E$ results in exactly two components. Let $\{X, S \setminus X\}$ be the corresponding partition of S, and let $w(e) = f(X) = f(S \setminus X)$.

Proposition 5. *Suppose that $a, b \in S$ are any two distinct nodes, and the edges $e_1, e_2, \ldots, e_k$ are the edges on the (unique) path between a and b in the tree returned by Algorithm 1. If e_i is the minimum weight edge (for $1 \leq i \leq k$), then the partition $\{A, B\}$ induced by removing the edge e is the minimum weight partition separating a from b (for f). In other words,*

$$A \in \operatorname*{argmin}_{X \in \mathcal{S}(a,b)} f(X)$$

2.2 Using the Gomory-Hu tree for finding approximate k-partitions

Deleting any $k - 1$ edges of the Gomory-Hu tree (or any other tree) results in k components, and hence a k-partition of S. The goal of this section is to show that if the $k - 1$ edges chosen are the $k - 1$ lightest weight edges, then the resulting k-partition is within a factor 2 of the optimal. Let us denote by $\mathcal{L}$ the set of lightest weight $k - 1$ edges in the Gomory-Hu tree. The quantity $\sum_{e \in \mathcal{L}} w(e)$ will be quite important, as we will prove bounds in terms of this quantity.

We start with a bound on the the cost of separating any component from the rest of the graph.

Lemma 6. *Suppose that $T = (S, E)$ is a Gomory-Hu tree for (S, F) with corresponding edge weight function $w : E \rightarrow \mathbb{R}$. Suppose that a subset $G \subseteq E$ of edges are deleted, and that A is a (connected) component in the*

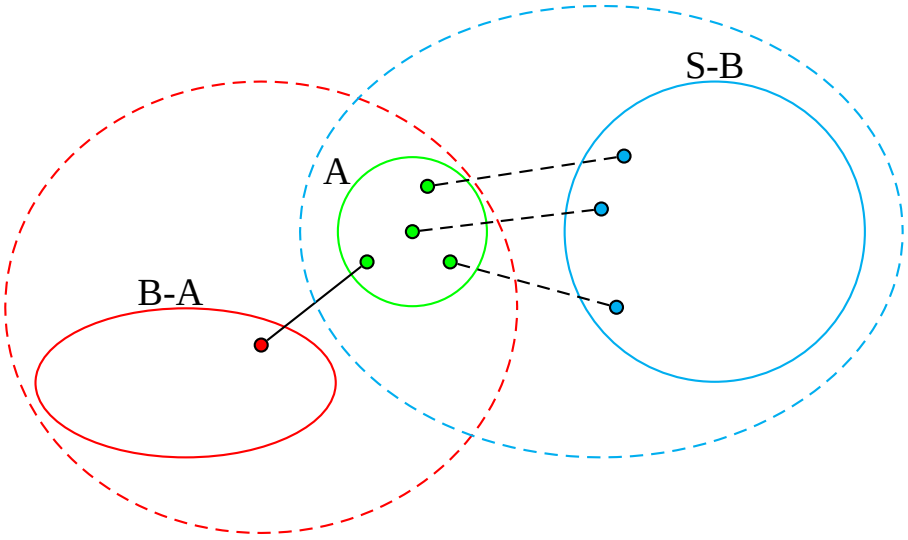

Figure 5: The edges in G'_A going between A and $S \setminus B$ are shown dashed, while the edge in $G_A \setminus G'_A$ going between A and $B \setminus A$ is shown as a solid line.

resulting graph. Let G_A be the subset of G that is adjacent to some node in A. Then

$$f(A) \leq \sum_{e \in G_A} w(e)$$

Proof. We prove this result by induction on $|G_A|$. If $|G_A| = 1$, then the result follows from the way weights are assigned to edges (and in this case, equality holds). Suppose that the result holds for all $|G_A| < k$. Consider the case when $|G_A| = k$. Let $G' \subseteq G$ with $|G'_A| = k - 1$ where G'_A is the subset of G' that is adjacent to some node in A. Then the removal of G'_A results in some component that strictly contains A. Let B be this component. Note that if e is the (unique) edge in $G_A \setminus G'_A$, then the removal of e separates $B \setminus A$ from $(S \setminus B) \cup A$. This is shown in Figure 5.

By the submodular inequality, we have

$$f(B) + f((S \setminus B) \cup A) \geq f(A) + f(S)$$

Since $f(S) = 0$ (see discussion in the previous section), we have

$$\begin{aligned} w(G_A) &= w(e) + w(G'_A) \\ &= f(B) + f((S \setminus B) \cup A) \\ &\geq f(A) \end{aligned}$$

Therefore, the result holds when $|G_A| = k$ and hence, by induction, the result holds for all values of $|G_A|$. □

Corollary 7. *Suppose that $\{B_1, B_2, \ldots, B_k\}$ is the partition obtained by deleting the edges in G, where $G \subseteq E$ is a set of $k - 1$ edges in E. Then*

$$\sum_{i=1}^{k} f(B_i) \leq 2 \cdot \sum_{e \in G} w(e)$$

Proof. Observe that each edge in G is adjacent to exactly two sets in $\{B_1, B_2, \ldots, B_k\}$. Therefore,

$$\sum_{i=1}^{k} f(B_i) \leq \sum_{i=1}^{k} \sum_{e \in G_{B_i}} w(e) = 2 \sum_{e \in G} w(e)$$

□

Lemma 8. *Suppose that $\{A_1, A_2, \ldots, A_k\}$ is any k-partition of S. Then*

$$\sum_{i=1}^{k-1} f(A_i) \geq \sum_{e \in \mathcal{L}} w(e)$$

where $\mathcal{L}$ is a set of the $k-1$ lightest weight edges in E. In particular, since the numbering of the elements of the partition is arbitrary, we have

$$\sum_{i=1}^{k} f(A_i) \geq \left(\frac{k}{k-1}\right) \sum_{e \in \mathcal{L}} w(e)$$

Proof. Construct a graph with vertex set $V_P = \{A_1, A_2, \ldots, A_k\}$. The edges of this graph are

$$E_P = \{\, \{A_i, A_j\} : \{a_i, a_j\} \in E \text{ with } a_i \in A_i, a_j \in A_j \}$$

This is a connected graph (because it results from contacting the set A_i for $1 \leq i \leq k$ in the Gomory-Hu tree which is connected). Therefore, we can discard some subset of the edges to make this graph a tree. Let (V_P, G_P) be this tree. For each edge $e \in G_P$, there is some edge e' in the Gomory-Hu tree so that the end points a_i and a_j of e' are in different partition A_i and A_j. In particular, Since both $\{A_i, S \setminus A_i\}$ and $\{A_j, S \setminus A_j\}$ separate a_i from a_j, we must have $f(A_i) \geq w(e')$ and $f(A_j) \geq w(e')$. Therefore, for each partition A_i, we have $f(A_i) \geq w(e)$ for each edge that is adjacent to A_i. Therefore,

$$\sum_{i=1}^{k-1} f(A_i) \geq \sum_{e \in G_P} w(e) \geq \sum_{e \in \mathcal{L}} w(e)$$

because $\mathcal{L}$ was chosen to be the set of $k-1$ lightest weight edges of the Gomory-Hu tree. Now, observe that we can always number the sets so that $f(A_k) = \max\{f(A_1), f(A_2), \ldots, f(A_k)\}$. Hence $f(A_k) \geq \frac{1}{k-1} \sum_{i=1}^{k-1} f(A_i)$, and therefore,

$$\sum_{i=1}^{k} f(A_k) \geq \left(\frac{k}{k-1}\right) \sum_{i=1}^{k-1} f(A_i) \geq \left(\frac{k}{k-1}\right) \sum_{e \in G_P} w(e) \geq \left(\frac{k}{k-1}\right) \sum_{e \in \mathcal{L}} w(e)$$

□

In [34], Saran and Vazirani showed that deleting the $k-1$ lightest weight edges results in a partition that is within twice the optimal. We can combine the previous propositions to obtain a similar result for symmetric submodular functions.

Proposition 9. *Let $\{B_1, B_2, \ldots, B_k\}$ be the partition obtained by deleting the lightest weight $k-1$ edges. Let $\{A_1, A_2, \ldots, A_k\}$ be any other partition. Then*

$$\sum_{i=1}^{k} f(B_i) \leq 2\left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} f(A_i)$$

Proof.

$$\sum_{i=1}^{k} f(B_i) \leq 2 \sum_{e \in F} w(e) \leq 2\left(1 - \frac{1}{k}\right) \sum_{i=1}^{k} f(A_i)$$

□

Since this holds for any k-partition $\{A_1, A_2, \ldots, A_k\}$, it holds in particular for the optimal k-partition. Therefore, deleting the $k-1$ lightest weight edges yields a factor 2 approximation algorithm. However, we are more interested in approximating the sum $\sum_{i=1}^{k} F(B_i)$ instead of the sum $\sum_{i=1}^{k} f(B_i)$. The recursive bisection algorithm we describe in the next section does just this.

$P_1 \leftarrow \{S\}$;
$\mathsf{totalCost} \leftarrow 0$;
for $i \leftarrow 1 \ldots k-1$ **do**
  $\mathsf{cost}(Q) \leftarrow \min_{X \subseteq 2^Q \setminus \{Q,\phi\}} [F(X) + F(Q \setminus X) - F(Q)]$ for all $Q \in P_i$ with $|Q| > 1$;
  Pick Q_i to minimize $\mathsf{cost}(Q_i)$; $\mathsf{totalCost} \leftarrow \mathsf{totalCost} + \mathsf{cost}(Q_i)$;
  $\{A_i, B_i\} \leftarrow$ optimal partition of Q_i to minimize $F(A_i) + F(B_i) - F(Q_i)$;
  $P_{i+1} \leftarrow P_i \setminus Q_i \cup \{A_i, B_i\}$;
end

Algorithm 2: A recursive bisection algorithm

3 Recursive Bisection

In this section, we consider a different algorithm for finding k-partitions. The algorithm The algorithm is quite simple. There are k iterations, and produce k partitions of S, namely $P_1, P_2, \ldots, P_k$ so that P_i is a refinement of P_{i-1}. At each step i, an element Q of partition P_i is chosen so that the cost of partitioning it is minimized, where the cost of partitioning Q is given by

$$\mathsf{cost}(Q) = \min_{X \subseteq 2^Q \setminus \{Q,\phi\}} \Big[F(X) + F(Q \setminus X) + F(Q) \Big]$$

Observe that $F(X) + F(Q \setminus X) + F(Q)$ is a symmetric submodular function on 2^Q and hence this minimization can be done using Queyranne's algorithm. Observe that P_i is a i-partition, and hence we obtain a k-partition after $k-1$ iterations (each of which requires an application of Queyranne's algorithm).

Lemma 10. *At iteration i, the value of* totalCost *is $\sum_{Q \in P_i} F(Q) - F(S)$. Therefore,*

$$\sum_{Q \in P_k} F(Q) - F(S) = \sum_{i=1}^{k-1} \mathsf{cost}(Q_i)$$

Proof. This is clearly true for $i = 1$. Suppose now that result holds for $i = m - 1$. Since

$$\Big[\sum_{Q \in P_m} F(Q) - F(S) - \Big] - \Big[\sum_{Q \in P_{m-1}} F(Q) - F(S) - \Big] = F(A_m) + F(B_m) - F(Q_m) = \mathsf{cost}(Q_m)$$

it follows that the result also holds for $i = m$. Therefore the resuld holds for all i. □

Lemma 11. *Suppose that $\mathcal{L}$ is the set of $k-1$ lightest weight edges in the Gomory-Hu tree for $f : 2^S \to \mathbb{R}$ given by $f(X) = F(X) + F(S \setminus X) - F(X)$. Then*

$$\sum_{i=1}^{k=1} \mathsf{cost}(Q_i) \leq \sum_{e \in \mathcal{L}} w(e)$$

Proof. We will show that $\mathsf{cost}(Q_i) \leq w(e_i)$ where e_i is the ith lightest weight edge in $\mathcal{L}$. At the first iteration, the minimum weight edge measures the cost of partitioning S, which is exactly what $\mathsf{cost}(Q_1)$ is since the only parititon in P_1 is S. Therefore the assertion holds for $i = 1$. Suppose that the assertion holds for $i < m$. Let (a_i, b_i) be the end-points of the edges e_i for $1 \leq i \leq k-1$. At stage m, it must be the case that (a_i, b_i) are in the same partition for some $1 \leq i \leq m$. Therefore, there is a partition $\{A, B\}$ of S which sepatates a_i and b_i for which $F(A) + F(B) - F(S) = w(e_i)$. Suppose that $a_i, b_i \in Q \in P_i$. Then because F is submodular, we must have $F(A) + F(B \cup (A \cap Q)) \geq F(A \cap Q) + F(S)$, and hence $F(A) - F(S) \geq F(A \cap Q) - F(B \cup (A \cap Q))$. Therefore,

$$w(e_i) = F(A) + F(B) - F(S) \geq F(A \cap Q) + F(B) - F(B \cup (A \cap Q))$$

Further, we have $F(B) + F((A \cup B) \cap Q) \geq F(B \cap Q) + F(B \cup (A \cap Q))$ and hence $F(B) - F(B \cup (A \cap Q)) \geq F(B \cap Q) - F((A \cup B) \cap Q)$. Therefore,

$$w(e_i) \geq F(A \cap Q) + F(B) - F(B \cup (A \cap Q)) \geq F(A \cap Q) + F(B \cap Q) - F((A \cup B) \cap Q)$$

Therefore, we can split some $Q \in P_i$ at a cost of no more than $w(e_i)$. Therefore, $\mathsf{cost}(Q_i) \leq w(e_i)$, and so

$$\sum_{i=1}^{k-1} \mathsf{cost}(Q_i) \leq \sum_{e \in \mathcal{L}} w(e)$$

□

Corollary 12. *Suppose that $\{B_1, B_2, \ldots, B_k\}$ is the set of partitions produced by the Algorithm 2. Then*

$$\sum_{i=1}^{k} F(B_i) - F(S) \leq \sum_{e \in \mathcal{L}} w(e)$$

Corollary 13. *Suppose that $\{B_1, B_2, \ldots, B_k\}$ is the set of partitions produced by the Algorithm 2. Then*

$$\sum_{i=1}^{k} F(B_i) \leq 2 \sum_{i=1}^{k} F(A_i)$$

Proof.

$$\begin{aligned}
\sum_{i=1}^{k} F(B_i) &\leq F(S) + \sum_{e \in \mathcal{L}} w(e) \\
&\leq F(S) + \sum_{i=1}^{k} f(A_i) \\
&\leq F(S) + \sum_{i=1}^{k} F(A_i) \\
&\leq 2 \sum_{i=1}^{k} F(A_i)
\end{aligned}$$

□

This result is interesting in that it produces a bound in terms of the Gomory-Hu tree, even though it does not actually produce a Gomory-Hu tree.

The recursive-bisection algorithm uses Queyranne's algorithm for finding a biparitition at each iteration. In [33], Rizzi pointed out that Queyranne's algorithm works for a larger class of functions than just submodular functions. In the next section, we consider one such important function, and we show that in this case the recursive-bisection algorithm leads to the optimal solution.

4 The Single-Linkage Criterion

Suppose that S is a metric space with distance function $d : S \times S \to \mathbb{R}$. Intuitively, if $x, y \in S$ are very similar, then the distance $d(x, y)$ between the two objects is small, while $d(x, y)$ is large if the objects are dissimilar. We can use this metric to define a distance $D : 2^S \times 2^S \to \mathbb{R}$ between subsets of S as follows.

$$D(A, B) = \min_{a \in A, b \in B} d(a, b)$$

Lemma 14. *The function $D : 2^S \times 2^S \to \mathbb{R}$ satisfies the following properties.*

1. *$D(A, B) = 0$ if $A \cap B \neq \phi$.*
2. *$D(\cdot, \cdot)$ is symmetric: $D(A, B) = D(B, A)$ for all $A, B \subseteq S$.*
3. *$D(A, B) \leq D(A, C) + D(C, B)$.*

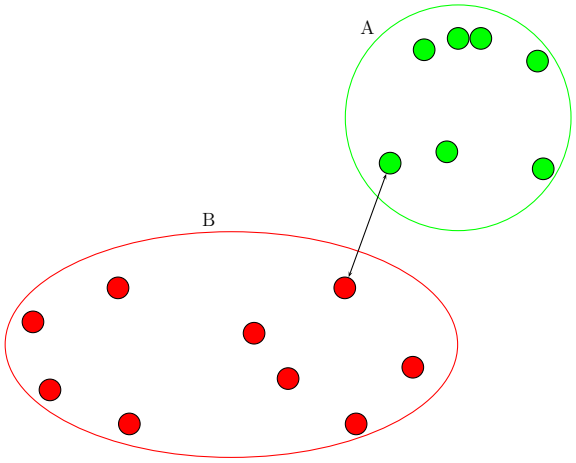

Figure 6: The Single-Linkage Criterion

Proof. If $A \cap B \neq \phi$, then there is some element $x \in A \cap B$. Hence $D(A,B) \leq d(x,x) = 0$. Now, $D(A,B)$ is always non-negative because $d(x,y)$ is always non-negative, and hence $D(A,B) = 0$. The second assertion is immediate from the definition of $D(\cdot,\cdot)$, while the third assertion follows from the triangle inequality for the metric $d(\cdot,\cdot)$. □

A partition $\{A, S \setminus A\}$ of S that maximizes $D(A, S \setminus A)$ is a partition that is as *seperated* as possible, and so is a natural criterion for clustering. The single-linkage criterion is given by

$$J_k^{\mathsf{sl}}(\{A_1, A_2, \ldots, A_k\}) = -\min_{i \neq j} D(A_i, A_j) = -\min_{\substack{A_i \neq A_j \\ a_i \in A_i, a_j \in A_j}} d(a_i, a_j)$$

Figure 6 illustrates this for $k = 2$. Observe that this criterion works find even if $d(x,y)$ is an element of some ordered set $\mathcal{R}$ instead of a real number.

We now show that Queyranne's algorithm can be used for finding a partition that maximizes $D(A, S \setminus A)$ (or minimizes $-D(A, S \setminus A)$). The function, as defined is not submodular. However, we can still use Queyranne's algorithm due to a result by Rizzi [33] who showed that Queyranne's algorithm works even when the objective function f is not submodular, as long as f is *monotone* and *consistent*.

Definition 15. *Suppose that f is a function defined on pairs of disjoint subsets of S. Then f is monotone if*

$$f(R,T') \leq f(R,T) \text{ for all } R, T, T' \subseteq S \text{ with } T' \subseteq T \text{ and } R \cap T = \phi$$

and f is consistent if

$$f(A, W \cup B) \geq f(B, A \cup W) \text{ if } A, B, W \subseteq S \text{ are disjoint and satisfy } f(A,W) \geq f(B,W)$$

Therefore, our next goal is to show that the function $-D(\cdot,\cdot)$ is monotone and consistent. Once we show this, it will follow from Rizzi's result that we can find a 2-clustering $\{S_1, S_2\} = \{S_1 S \setminus S_1\}$ that minimizes $-D(S_1, S_2)$, and hence maximizes $D(S_1, S_2)$.

Lemma 16. *If $R \subseteq T$, then $D(U,T) \leq D(U,R)$ (and hence $-D(U,R) \leq -D(U,T)$).*

Proof. This would imply that $-D$ is monotone. To see this, observe that

$$\begin{aligned} D(U,T) &= \min_{u \in U, t \in T} d(u,t) = \min\left(\min_{u \in U, r \in R} d(u,r), \min_{u \in U, t \in T \setminus R} d(u,t)\right) \\ &\leq \min_{u \in U, r \in R} d(s,r) = D(U,R) \end{aligned}$$

□

Lemma 17. *Suppose that A, B, W are disjoint subsets of S and $D(A, W) \leq D(B, W)$. Then $D(A, W \cup B) \leq D(B, A \cup W)$.*

Proof. To see this first observe that $D(A, B \cup W) = \min(D(A, B), D(A, W))$ because

$$D(A, W \cup B) = \min_{a \in A, x \in W \cup B} D(a, x) = \min \left(\min_{a \in A, w \in W} D(a, w), \min_{a \in A, b \in B} D(A, b) \right)$$

It follows that

$$\begin{aligned} D(A, B \cup W) &= \min (D(A, B), D(A, W)) \\ &\leq \min (D(A, B), D(B, W)) \\ &= \min (D(B, A), D(B, W)) \\ &= D(B, A \cup W) \end{aligned}$$

□

Therefore, if $-D(A, W) \geq -D(B, W)$, then $-D(A, W \cup B) \geq -D(B, A \cup W)$. Hence $-D(\cdot, \cdot)$ is consistent. Therefore, $-D(\cdot, \cdot)$ is symmetric, monotone and consistent. Hence it can be minimized using Queyranne's algorithm [33], and so we have a procedure to compute optimal 2-clusterings with respect to J_2^{sl}. We now extend this to compute optimal k-clusterings.

4.1 Optimal k-clusterings

We start off by extending our objective function for k-clusterings in the obvious way. The function $D(R, T)$ can be thought of as defining the *separation* or *margin* between the clusters R and T. The natural generalization to more than two clusters is

$$J_k^{\mathsf{sl}}(\{S_1, S_2, \ldots, S_k\}) = \min_{i \neq j} D(S_i, S_j) = \min_{\substack{S_i \neq S_j \\ s_i \in S_i, s_j \in S_j}} d(s_i, s_j)$$

Note that $J_2^{\mathsf{sl}}(\{R, T\}) = D(R, T)$ for a 2-clustering. The function $J_k^{\mathsf{sl}} : \mathcal{C}_k(S) \to \mathbb{R}$ takes a single clustering as its argument. However, $D(\cdot, \cdot)$ takes two disjoint subsets of S as its arguments the union of which need not be S in general. The margin is the distance between the closest elements of different clusters, and hence we will be interested in finding k-clusters that maximize the margin. Therefore, we seek an element in $\mathcal{O}_k(S) = \arg\max_{\{S_1, S_2, \ldots, S_k\} \in \mathcal{C}_k(S)} J_k^{\mathsf{sl}}(\{S_1, S_2, \ldots, S_k\})$. Let $v_k(S)$ be the margin of an element in $\mathcal{O}_k(S)$. Therefore, $v_k(S)$ is the best possible margin of any k-clustering of S. An obvious approach to generating optimal k-clusterings given a method of generating optimal 2-clusterings is the following. Start off with an optimal 2-clustering $\{S_1, S_2\}$. Then apply the procedure to find 2-clusterings of S_1 and S_2, and stop when you have enough clusters. There are two potential problems with this approach. First, it is not clear that an optimal k-clustering can be a refinement of an optimal 2-clustering. That is, we need to be sure that there is an optimal k-clustering in which S_1 is the union of some of the clusters, and S_2 is the union of the remaining. Second, we need to figure out how many of the clusters S_1 is the union of and how many S_2 is the union of. In this section, we will show that for any $k \geq 3$, there is always an optimal k-clustering that is a refinement of any given optimal 2-clustering. A simple dynamic programming algorithm takes care of the second potential problem.

We begin by establishing some relationships between the separation of clusterings of different sizes. To compare the separation of clusterings with different number of clusters, we can try and merge two of the clusters from the clustering with more clusters. Say that $\mathcal{S} = \{S_1, S_2, \ldots, S_k\} \in \mathcal{C}_k(S)$ is any k-clustering of S, and $\mathcal{S}'$ is a $(k-1)$-clustering of S obtained by merging two of the clusters (say S_1 and S_2). Then $\mathcal{S}' = \{S_1 \cup S_2, S_3, \ldots, S_k\} \in \mathcal{C}_{k-1}(S)$.

Lemma 18. *Suppose that $\mathcal{S} = \{S_1, S_2, \ldots, S_k\} \in \mathcal{C}_k(S)$ and $\mathcal{S}' = \{S_1 \cup S_2, S_3, \ldots, S_k\} \in \mathcal{C}_{k-1}(S)$. Then $J_k^{\mathsf{sl}}(\mathcal{S}) \leq J_{k-1}^{\mathsf{sl}}(\mathcal{S}')$. In other words, refining a partition can only reduce the margin.*

Therefore, refining a clustering (i.e., splitting a cluster) can only reduce the separation. An immediate corollary is the following.

Corollary 19. *If $\mathcal{T}_l \in \mathcal{C}_l(S)$ is a refinement of $\mathcal{T}_k \in \mathcal{C}_k(S)$ (for $k < l$) then $J_l^{\mathsf{sl}}(\mathcal{T}_l) \leq J_k^{\mathsf{sl}}(\mathcal{T}_k)$. It follows that $v_k(S) \geq v_l(S)$ if $1 \leq k < l \leq n$.*

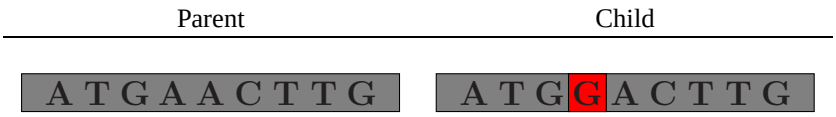

Figure 7: An example of a point mutation: The parent and child differ in a single position

Proof. It suffices to prove the result for $k = l - 1$. The first assertion follows immediately from Lemma 18. Let $\mathcal{S} \in \mathcal{O}_l(S)$ be an optimal l-clustering. Merge any two clusters to get $\mathcal{S}' \in \mathcal{C}_k(S)$. By Lemma 18, $v_k(S) \geq J_k^{\mathsf{sl}}(\mathcal{S}') \geq J_l^{\mathsf{sl}}(\mathcal{S}) = v_l(S)$. □

Next, we consider the question of constructing larger partitions (i.e., partitions with more clusters) from smaller partitions. Given two clusterings $\mathcal{S} = \{S_1, S_2, \ldots, S_k\} \in \mathcal{C}_k(S)$ and $\mathcal{T} = \{T_1, T_2, \ldots, T_l\} \in \mathcal{C}_l(S)$ of S, we can create a new clustering $\mathcal{U} = \{U_1, U_2, \ldots, U_m\} \in \mathcal{C}_m(S)$ to be their common refinement. That is, the clusters of $\mathcal{U}$ consist of those elements that are in the same clusters of both $\mathcal{S}$ and $\mathcal{T}$. Formally,

$$\mathcal{U} = \{\, S_i \cap T_j \,:\, 1 \leq i \leq k,\, 1 \leq j \leq l\}$$

Lemma 20. *Let $\mathcal{S} = \{S_1, S_2, \ldots, S_k\} \in \mathcal{C}_k(S)$ and $\mathcal{T} = \{T_1, T_2, \ldots, T_l\} \in \mathcal{C}_l(S)$ be any two partitions. Let $\mathcal{U} = \{U_1, U_2, \ldots, U_m\} \in \mathcal{C}_m(S)$ be their common refinement. Then $J_k^{\mathsf{sl}}(\mathcal{U}) = \min\left(J_k^{\mathsf{sl}}(\mathcal{S}), J_k^{\mathsf{sl}}(\mathcal{T})\right)$.*

Proof. It is clear that $J_m^{\mathsf{sl}}(\mathcal{U}) \leq \min\left(J_k^{\mathsf{sl}}(\mathcal{S}), J_l^{\mathsf{sl}}(\mathcal{T})\right)$. To show equality, note that if a, b are in different clusters of $\mathcal{U}$, then a, b must have been in different clusters of either $\mathcal{S}$ or $\mathcal{T}$. □

This result can be thought of as expressing a relationship between J_k^{sl} and the lattice of partitions of S.

Lemma 21. *Suppose that $\mathcal{S} = \{S_1, S_2\} \in \mathcal{O}_2(S)$ is an optimal 2-clustering. Then there is always an optimal k-clustering that is a refinement of $\mathcal{S}$.*

Proof. Suppose that this is not the case. If $\mathcal{T} = \{T_1, T_2, \ldots, T_k\} \in \mathcal{O}_k(S)$ is an optimal k-clustering, let r be the number of clusters of $\mathcal{T}$ that "do not respect" the partition $\{S_1, S_2\}$. That is, r is the number of clusters of $\mathcal{T}$ that intersect both S_1 and S_2 :

$$r = |\{\, 1 \leq i \leq k \,:\, T_i \cap S_1 \neq \phi \text{ and } T_i \cap S_2 \neq \phi\}|$$

Pick $\mathcal{T} \in \mathcal{O}_k(S)$ to have the smallest r. If $r = 0$, then $\mathcal{T}$ is a refinement of $\mathcal{S}$ and there is nothing to show. Otherwise, $r \geq 1$. Assume WLOG that $T_1^{(1)} = T_1 \cap S_1 \neq \phi$ and $T_1^{(2)} = T_1 \cap S_2 \neq \phi$. Then $\mathcal{T}' = \left\{T_1^{(1)}, T_1^{(2)}, T_2, T_3, \ldots, T_k\right\} \in \mathcal{C}_{k+1}(S)$ is a refinement of $\mathcal{T}$ and satisfies $J_k^{\mathsf{sl}}(\mathcal{T}') = J_k^{\mathsf{sl}}(\mathcal{T})$. This follows from Lemma 3 along with the fact that

- $D(T_i, T_j) \geq J_k^{\mathsf{sl}}(\mathcal{T})$ for any $2 \leq i < j \leq k$,
- $D(T_1^{(i)}, T_j) \geq J_k^{\mathsf{sl}}(\mathcal{T})$ for any $i \in \{1, 2\}$ and $2 \leq j \leq k$,
- $D(T_1^{(1)}, T_1^{(2)}) \geq J_k^{\mathsf{sl}}(\{S_1, S_2\}) = v_2(S) \geq v_k(S) = J_k^{\mathsf{sl}}(\mathcal{T})$

Now, pick two clusters of $\mathcal{T}'$ that are either both contained in the same cluster of $\mathcal{S}$ or both "do not respect" $\mathcal{S}$. Clearly this can always be done. Merge these clusters together to get an element $\mathcal{T}'' \in \mathcal{C}_k(S)$. By Lemma 18 merging clusters cannot decrease the margin. Therefore, $J_k^{\mathsf{sl}}(\mathcal{T}'') = J_k^{\mathsf{sl}}(\mathcal{T}') = J_k^{\mathsf{sl}}(\mathcal{T})$. However, $\mathcal{T}''$ has fewer clusters that do not respect $\mathcal{S}$ hand T has, and hence we have a contradiction. □

This lemma implies that Queyranne's algorithm, along with a simple dynamic programming algorithm can be used to find the best k clustering with time complexity $O(k\,|S|^3)$. Observe that in fact this problem can be solved in time $O(|S|^2)$ ([9]). Even though using Queyranne's algorithm is not the fastest algorithm for this problem, the fact that it optimizes this criterion implies that it can be used to optimize conic combinations of submodular criteria and the single-linkage criterion.

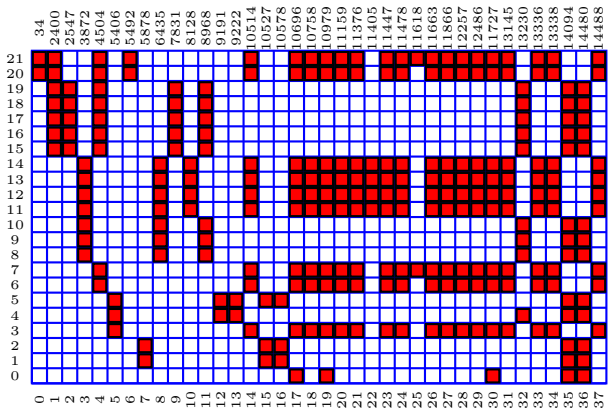

Figure 8: The Single Nucleotide Polymorphisms that occur in the ACE gene in 22 Chromosomes (data taken from [37]) The numbers above the column represent the position on the chromosome. A filled indicates the presence of the rare allele, while a blank square indicates the presence of the common allele.

5 An Application: Mining SNPs

Most of the variation in the human genome is due to point mutations, which substitutes a single nucleotide for another. For example, the parent and child nucleotide sequences shown in Figure 7 differ in just one position, and so could be the result of a point mutation. When such mutations occur, different individuals in the population could have different nucleotides at a given site/position. This polymorphism at the site/position of the mutation is called a **S**ingle **N**ucleotide **P**olymorphism (SNP). SNPs may occur in both coding (in which case the SNP could be either non-synonymous or synonymous) and non-coding regions (UTRs, introns, intergenic regions). Normally, a position in the genome in which more than one nucleotide is observed in the population is considered a SNP only if it is undergoes stable inheritance, and at least 1% of the population has the variation. It is estimated that there are about 5 million commonly occurring SNPs which account for the bulk of human genomic diversity and occur on the average of 1 every 600 base-pairs. For most SNPs, there are only two variants, which we term as the common allele, and the rare allele. Figure 8 shows the the polymorphisms that occur in the **A**ngiotensin **C**onverting **E**nzyme (ACE) gene in 22 chromosomes. The rows of this matrix represent individual chromosomes, while the columns of the matrix represent a particular site on the chromosome. A filled square indicates the presence of the rare allele.

While most of these SNPs are not associated with any (obvious) external phenotype, they have an impact on the protein synthesized, and hence are associated with either diseases, or the effectiveness of medications. For example, a mutation at location 14188 of the ACE gene determines the effectiveness of certain blood pressure medications. Following such discoveries, it is hoped that mapping individual SNPs will help with the following goals:

1. Understanding genetic component of disease by providing disease markers.

2. Understanding genetic component of drug responses allowing for personalized medicine.

Both these goals require that we map all the SNPs. However, since determining each SNP can cost between $0.10-$1.00, cataloging all the 100,000 or so SNPs for each individual is too expensive. therefore, we need a cost-effective mechanism of capturing as many of the SNPs as possible. As can be seen from Figure 8, the presence of SNPs at different sites can be highly correlated, and therefore, we might be able to infer the entire genome with reasonably high accuracy by observing only a few positions. For example, sets of SNPs that are in close proximity are likely to be inherited in blocks, and hence the presence or absence of these SNPs are highly correlated. Alleles consisting of such blocks of SNPs form a haplotype reflecting descent from a single ancient ancestral chromosome. Since there is very limited diversity in these blocks, knowledge of one or a few of the SNPs in these blocks is sufficient to reconstruct the remaining SNPs with high accuracy. This block effect is referred to as *linkage disequilibrium* (LD) which is formally defined as the non-random association of SNPs in a short block of a chromosome. If a particular SNP causes (or increases susceptibility to) a genetic disease, then any other SNP in high LD with the disease causing SNP will show significant statistical association with the disease, and hence can be used as a genetic marker for the disease. Typically

two SNPs are inherited as a block if recombination does not occur at some spot between the SNPs. Since the genome has very long segments with very low recombination punctuated with short segments of very high recombination, it is possible that there could be a very large block which is inherited as a unit. In such large blocks, it is quite possible that additional mutations occur (especially when the population size is large) leading to changes in the haplotype block differing from the ancestral chromosome. Nonetheless, it is very likely that even large haplotype blocks can be reconstructed very accurately using just a few SNPs. The goal therefore is to select a minimal set of *haplotype tagging SNPs* that will allow accurate reconstruction of the remaining SNPs.

We can think of this as a inference problem. We have a correlated set of random variables, $\{X_v\}_{v \in V}$. Here V is the set of positions of the SNPs. X_v is a binary random variable which is 0 when the common allele occurs at position v, and 1 when the rare allele occurs at position v. Since the random variables are not independent, knowledge of one or more of the random variables allows to predict the unknown random variables with accuracy that exceeds random guessing. Therefore, we want to find a minimal set of random variables, which if observed, allows us to infer the remaining random variables with the desired accuracy.

5.1 Prior Work

Many approaches to this problem have been proposed. Most block based approaches assume that the size of the block is fixed (and specified), and the chromosome/gene is partitioned into these blocks and a dynamic programming algorithm selects the optimal set of SNPs within sliding blocks of the specified size [38, 4, 32]. There are three main issues with this approach. First, the block size is not fixed and varies across the chromosome. Further, it is hard to predict the block size. Finally, even if the block size is known, the optimal algorithms are typically exponential in the size of the block and so could be computationally very expensive. The prior approaches to block-based haplotype tagging SNPs assume that the blocks are contiguous blocks of the genome. However, it has been observed that there can be significant LD even across very long stretches of the genome, and two SNPs separated by a large distance can have high LD even if neither has high LD with any SNP in the region between these SNPs. We present an alternative way of selecting haplotype blocks which are not (necessarily) composed of SNPs from a contiguous region of the genome, but instead, we select SNPs to be in a block based on the Minimum Description Length criterion. This results in a polynomial time algorithm for partitioning the SNPs into haplotype block.

Block-free approaches search for the optimal set of SNPs that will allow reconstruction regardless of the block structure. In such case, the problem could be formulated either as finding the most informative set of k SNPs, or as finding the minimum sized set of SNPs that allow reconstruction to a prescribed degree of accuracy. We will assume the former which can be thought of as a budgeted SNP selection problem. A naive approach to this problem is to enumerate all subsets of k SNPs and then pick the optimal set according to some measure of informativeness. This leads to an algorithm which is exponential in k. However the problem is NP-complete [4] and so (in the worst case) all algorithms that guarantee optimality will take exponential time. Another approach proposed in [27] attempts to construct "eigenSNPs" by performing a principal components analysis of the SNP data. The eigneSNPs are then analyzed to pick the best set of SNPs. This is not guaranteed to yield the optimal set of SNPs (which is to be expected because the problem is NP-complete), but it also does not give an approximation guarantee. One problem with this approach is that the process of converting eigenSNPs (which have real valued positive and negative coefficients) to discrete sets of SNPs is somewhat ad-hoc. Another issue is the fact that in the presence of multiple highly-correlated SNPs, it is possible that the SNP matrix is non-singular, and so the eigenvectors can be expressed as different linear combinations of the columns, resulting in both numerical instability and possible selection of a excessively large set of SNPS.

Other than the eigenSNP paper of [27], most of the other approaches [7, 37, 4] only consider pairwise correlation of the SNPs. For example, [7] uses the pairwise statistics D' and r^2 defined in [10] to characterize the pairwise LD between pairs of polymorphic sites, and then use this criterion to select SNPs. [4] propose a related measure which measures pairwise informativeness.

It is possible that a given SNP cannot be reconstructed to the desired degree of accuracy using any other single SNP, but can be reconstructed to this degree of accuracy using more than one SNP. The two algorithms we present in this chapter are based on information theoretic criteria that reflect the accuracy of reconstructing based on all the chosen SNPs (simultaneously). The first is a greedy selection algorithm to maximize the informativeness of SNPs (measured by entropy as suggested by [21, 3]) which can be shown to yield a 0.63 factor approximation algorithm. The second is an MDL based criterion described previously, which attempts to partition the set of SNPs into subsets of highly correlated SNPs and then selecting representative SNPs from each cluster.

6 Information Theoretic Approaches

[21, 3] use Shannon Entropy to measure the amount of diversity of a collection of SNPs. While a variety of techniques have been proposed to select a set of SNPs with the most entropy, we show that a simple greedy algorithm leads to a 0.63 factor approximation algorithm for the following problem:

Problem 2. *Given a collection of S of n SNPs, find a subset $T \subset S$ of size k with the largest possible entropy.*

This problem is NP-complete, but it can be shown that Algorithm 1 returns a 0.63 approximation algorithm for this problem. In fact it can also be shown that it is not possible for a polynomial time algorithm to have a better approximation guarantee unless P=NP.

$T_0 \leftarrow \phi$;
for $i := 1 \ldots k$ **do**
  Pick s_i so that $H\Big(T_{i-1} \cup \{s_i\}\Big)$ is maximized;
  $T_i \leftarrow T_{i-1} \cup \{s_i\}$;
end
return T_k;

Algorithm 3: A greedy algorithm to pick a set of SNPs with the most entropy

A different approach has been advocated in [4]. They suggest that it is more important to pick a subset that yields as much information about the remaining SNPs as possible. While they present an approach based on pair-wise predictability, Mutual Information generalizes this approach to allow the use of all the SNPs to predict the remaining SNPs. The mutual information between two sets T_1 and T_2 of SNPs is given by

$$I(T_1; T_2) = H(T_1) + H(T_2) - H(T_1 \cup T_2)$$

This measures the reduction in entropy (or uncertainty) in T given information about the SNPs in S. Haplotype blocks are generally considered to be blocks which have low diversity, and are more or less (statistically) independent of other haplotype blocks. Therefore, partitioning the SNPs into multiple blocks to minimize the mutual information across the blocks can be considered to be a good approximation of the optimal haplotype blocks. This is also directly related to the Minimum Description Length (MDL) criterion for partitioning. The entropy $H(S)$ measures the asymptotic description length using a universal code, and so the MDL criterion for partitioning the set of SNPs S into two parts T_1 and $T_2 = S \setminus T_1$ so that $H(T_1) + H(T_2)$ is minimized. In fact, this is equivalent to partitioning the SNPs into two parts to minimize the mutual information between the two parts:

$$\begin{aligned}
\arg\min_{S=T_1 \cup T_2} I(T_1; T_2) &= \arg\min_{S=T_1 \cup T_2} \Big[H(T_1) + H(T_2) + H(T_1 \cup T_2) \Big] \\
&= \arg\min_{S=T_1 \cup T_2} \Big[H(T_1) + H(T_2) + H(S) \Big] \\
&= \arg\min_{S=T_1 \cup T_2} \Big[H(T_1) + H(T_2) \Big]
\end{aligned}$$

This approach does not consider the distance between the SNP positions, just their statistical relationship. This may be a better approach to constructing sets with high LD than considering just locality because studies have shown that there can be high LD even across large distances. It turns out that there is an algorithm which when given a set of SNPs, can find the optimal partition of SNPs to minimize the amount of mutual information across the two parts (in other words, it partitions the SNPs into parts which are as statistically independent as possible). This algorithm can be applied recursively in a simple dynamic programming framework to pick the optimal set of haplotype blocks. This approach is closely related to the approaches of [37, 7], which cluster the SNPs on the basis of the similarities of the pairwise LD measures, and then select one SNP per cluster.

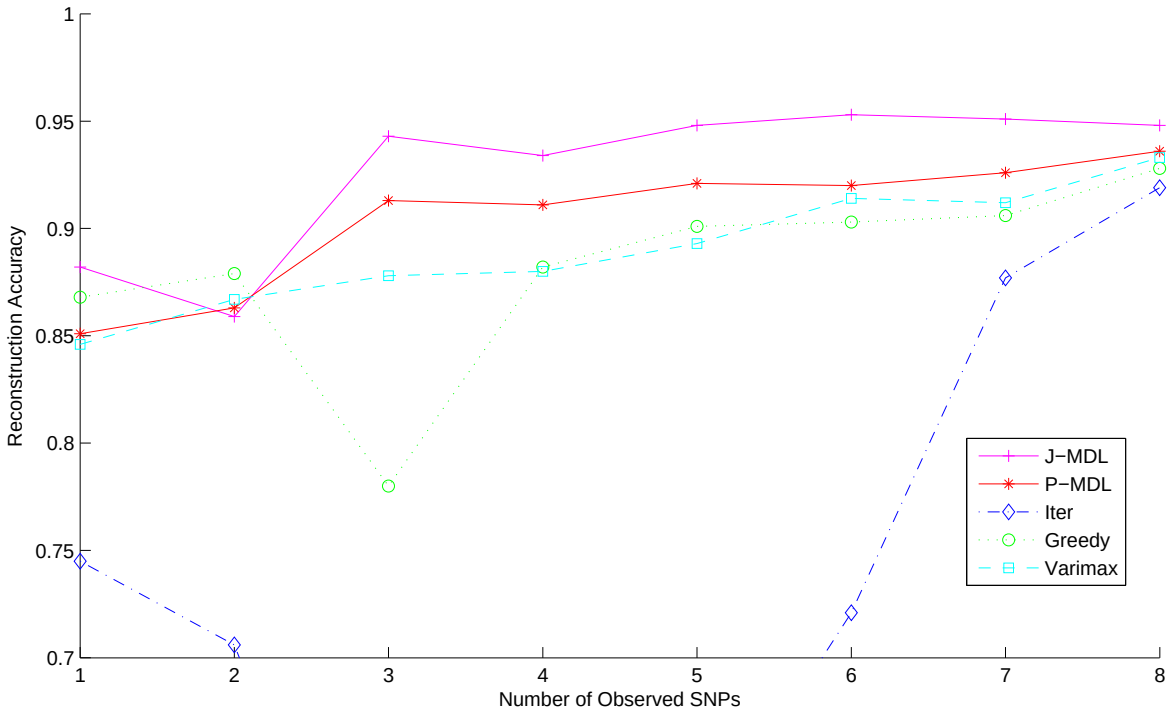

Figure 9: Leave-1-Out CV on the ACE Gene

7 Experimental Results

Figures 9, 10, and 11 compare various algorithms for reconstructing all the alleles from a small subset of alleles. Besides the SNP selection algorithm presented in this chapter, we evaluate the Varimax, Greedy and Iterative Selection algorithms of [28]. In [28], it is shown that these algorithms substantially outperform other algorithms, and so these three are used as the baseline. While the MDL clustering algorithm presented in this chapter can be used to select the SNPs that are to be observed, different algorithms can be used to predict the remaining SNPs. We use two techniques for reconstruction. The first is to predict an unknown SNP based on the known SNP from the same cluster as the unknown SNP, and this is denoted as 'MDL' in the figures. The second is to use all the observed SNPs (from all the custers) to generate a classifier for each unknown SNPs. This is denoted 'IMDL' in the figures.

Figures 9, 10, and 11 compares the algorithms for reconstruction accuracy on the ACE gene, the IBD gene and the ABCB gene respectively.

8 Conclusions

In this chapter, we showed that several natural clustering criterion are either submodular, or reducible to submodular criteria. As a result, we can use Queyranne's algorithm to find the optimal 2-clustering with respect to these criterion. For k-clustering ($k > 2$), we can use the Gomory-Hu tree produced by Queyranne's algorithm to produce clusterings guaranteed to be within $2(1 - 1/k)$ of the optimal. Because the MDL criterion is submodular, we can use Queyranne's algorithm for MDL clustering. We applied MDL clustering to the problem of determining an optimal set of haplotype tagging SNPs, and this results in substantially better results than the algorithms currently used for this purpose.

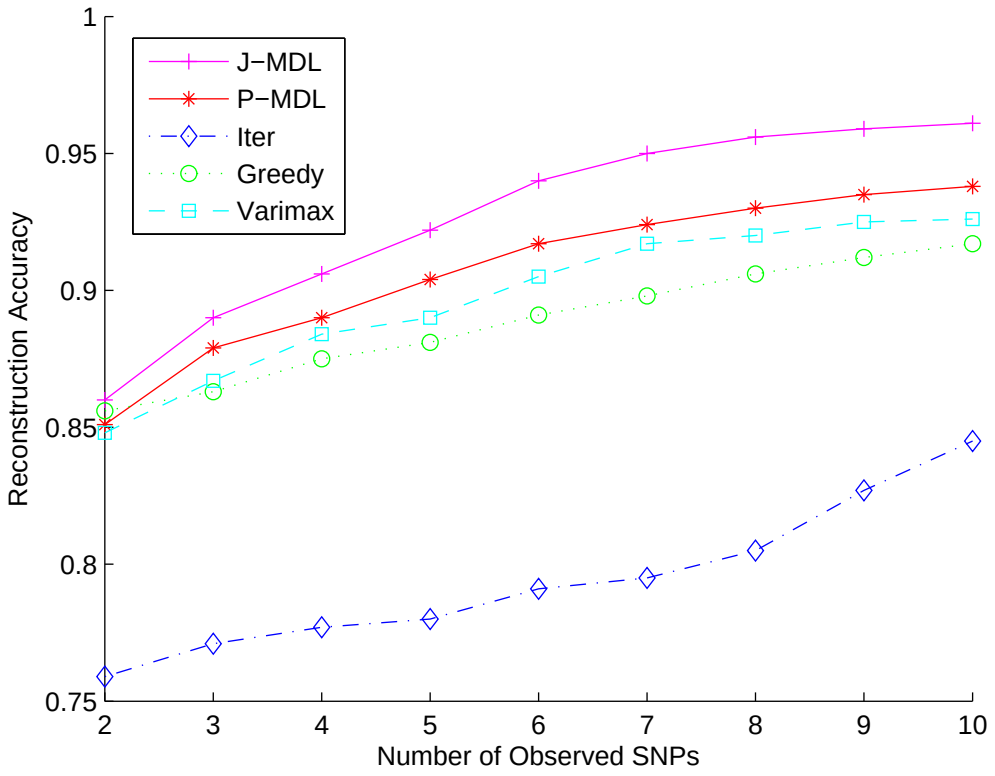

Figure 10: Leave-1-Out CV on the IBD Gene

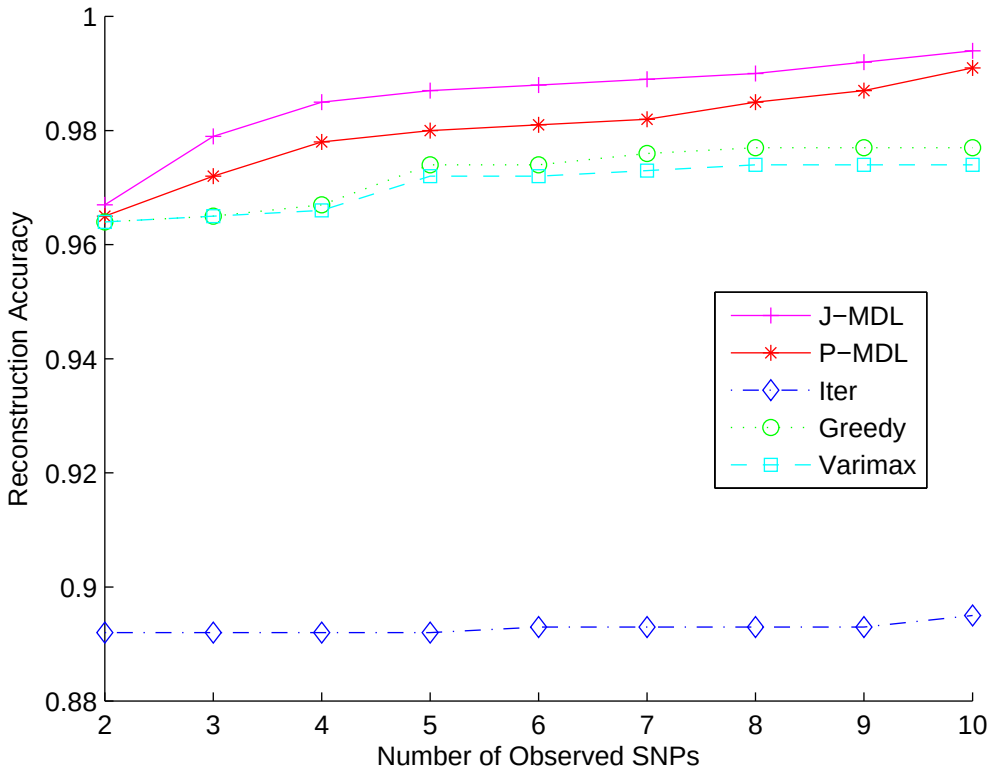

Figure 11: 10-fold CV on the ABCB Gene

References

[1] C. J. Alpert and A. B. Kahng. Recent developments in netlist partitioning: A survey. *Integration: The VLSI Journal*, 19(1):1–81, 1995.

[2] D. Author and S. Vassilvitskii. How slow is the k-means method? In *Proceedings of the 2006 symposium on computational geometry*, 2006.

[3] H. I. Avi-Itzhak, X. Su, and F. M. De La Vega. Selection of minimum subsets of single nucleotide polymorphisms to capture haplotype block diversity. In *Proceedings of Pac. Symp Biocomput*, pages 466–477, 2003.

[4] V. Bafna, B. V. Halldorsson, R. Schwartz, A. G. Clark, and S. Istrail. Haplotypes and informative snp selection algorithms: don't block out information. In *Proceedings of the Annual Conference on Research in Computational Molecular Biology, Berlin*, 2003.

[5] I. Borg and P. Groenen. *Modern multidimensional scaling: theory and applications*. Springer-Verlag New York, 1997.

[6] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Improved algorithms for hypergraph bipartitioning. In *ASPDAC*, pages 661–666, 2000.

[7] C. S. Carlson, M. A. Eberle, M. J. Rieder, Q. Yi, L. Kruglyak, and D. A. Nickerson. Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. *American Journal Human Genetics*, 74:106–120, 2004.

[8] M. F. Cox and M. A. A. Cox. *Multidimensional scaling*. Chapman and Hall, 2001.

[9] M. Dellattre and P. Hansen. Bicriterion cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(4), 1980.

[10] B. Devlin and N. Risch. A comparison of linkage disequilibrium measures for fine-scale mapping. *Genomics*, 29:311–322, 1995.

[11] C. Ding and X. He. K-means clustering via principal components analysis. In *Proceedings of the international conference on machine learning*, pages 225–232, 2004.

[12] G. W. Flake, R. E. Tarjan, and K. Tsioutsioulikis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4), 2004.

[13] L. E. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[14] M. X. Goemans and V. S. Ramakrishnan. Minimizing submodular functions over families of sets. *Combinatorica*, 15:499–513, 1995.

[15] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the Association of Computer Machinery*, 35:921–940, 1988.

[16] R. E. Gomory and T. C. Hu. Multiterminal network flows. *SIAM Journal on Applied Mathematics*, 9:165–174, 1961.

[17] J. Hao and J. B. Orlin. A faster algoirthm for finding a minimum cut in a graph. In *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 165–174, 1992.

[18] S. Hauck and G. Borriello. An evaluation of bipartitioning techniques. *IEEE Transactions on Computer Aided Design*, 16(8):849–866, 1997.

[19] S. Haykin. *Neural networks - A comprehensive foundation*. Prentice-Hall, 1999.

[20] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[21] R. Judson, B. Salisbury, J. Schneider, A. Windemuth, and J. C. Stephens. How many snps does a genome-wide haplotype map require? *Pharmacogenomics*, 3:379–391, 2002.

[22] R. Kannan, S. Vempala, and A. Vetta. On clusterings: good, bad and spectral. In *Proceedings of the fourty-first symposium on the foundations of computer science*, 2000.

[23] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of DAC*, pages 343–348, 1999.

[24] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Tech Journal*, 49:291–307, 1970.

[25] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. Som_pak: The self-organizing map program package, report a31. Technical report, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland, 1996.

[26] P. Kontkanen, P. Myllymäki, W. Buntine, J. Rissanen, and H. Tirr. An mdl framework for data clustering, 2004.

[27] Z. Lin and R. B. Altman. Finding haplotype tagging snps by use of principal components analysis. *American Journal of Human Genetics*, 75:850–861, 2004.

[28] Z. Lin and R. B. Altman. Finding haplotype tagging snps by use of principal components analysis. *American Journal of Human Genetics*, 75:850–861, 2004.

[29] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[30] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967.

[31] M. Meila and J. Shi. A random walks view of spectral segmentation. In *Proceedings of AISTATS*, 2001.

[32] Z. Meng, D. V. Zaykin, C. F. Xu, M. Wagner, and M. G. Ehm. Selection of genetic markers for association analyses using linkage disequilibrium and haplotypes. *American J. human Genetics*, 73:115–130, 2003.

[33] R. Rizzi. On minimizing symmetric set functions. *Combinatorica*, 20:445–450, 2000.

[34] H. Saran and V. V. Vazirani. Finding k-cuts withing twice the optimal. *SIAM Journal of Computation*, 24(1):101–108, 1995.

[35] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[36] M. Stoer and F. Wagner. A simple min-cut algorithm. In *Proceedings of the 1994 European Symposium on Algorithms*, 1994.

[37] X. Wu, A. Luke, M. Rieder, K. Lee, E. J. Togh, D. Nickerson, X. Zhu, D. Kan, and R. S. Cooper. An association study of angiotensinogen polymorphisms with serum level and hypertension in an african-american population. *J. Hypertens*, 21:1847–1852, 2003.

[38] K. Zhang, M. Deng, T. Chen, M. S. Waterman, and F. Sun. A dynamic programming algorithm for haplotype partitioning. *Proc. Nat. Acad. Sci.*, 99:7735–7339, 2002.