Minimum Range Balanced Cuts via Dynamic Subset Sums

David Eppstein*

Department of Information and Computer Science
University of California, Irvine, CA 92717

Tech. Report 95-10

March 17, 1995

Abstract

We find the balanced cut in a graph that minimizes the maximum difference between edge lengths, in time $O(m + n^2 \log n)$, improving a previous $O(m+n^{2.5})$ bound. We use subroutines for solving a dynamic subset sum problem, in time $O(\ell \log \ell \log n)$ per operation in the fully dynamic setting, or in time $O(\ell \log n)$ per operation in the *semi-online* setting in which one can predict a superset of future deletions.

*Work supported in part by NSF grant CCR-9258355.

1 Introduction

Many graph optimization problems, such as shortest paths, minimum spanning trees, and minimum cuts, aim to find a subgraph satisfying certain requirements and minimizing the sum of its edge lengths. By contrast, a *minimum range* problem [1, 2, 6, 7, 8] instead seeks to find a subgraph minimizing the maximum difference between any two edge lengths; in other words the lengths in the subgraph should be as uniform as possible.

In this paper we study the *minimum range balanced cut* problem: partition the vertices of our input graph G into two equal-cardinality subsets A and B, minimizing the differences among lengths of edges with one endpoint in A and the other in B. Equivalently, we seek a set of edges in G, the removal of which leaves connected components which can be combined into two such sets A and B. The corresponding standard optimization problem, of finding a minimum balanced cut, is NP-complete, but the minimum range balanced cut can be found in polynomial time. Dai et al. [2] note that the minimum range balanced cut problem can be used to approximate the minimum cardinality balanced cut, using an approach previously studied by the same authors for the minimum cut problem [3]: if one assigns random weights to the edges of a graph, the cuts with few edges will likely have small weight ranges, so the minimum range balanced cut can give a good approximation to the minimum balanced cut.

We also study a dynamic *subset sum problem*: given a collection of integers n_i and a target integer ℓ, maintain $x_i \in \{0,1\}$ satisfying $\sum x_i n_i = \ell$ (or more generally maximizing $\sum x_i n_i \leq \ell$) as values are inserted and deleted in the collection. The connection of subset sums with minimum range balanced cuts is that, if we are given a weight range, we can test whether the edges in that range partition G into pieces that can be assembled into sets A and B of the appropriate sizes by using a subset sum algorithm.

2 New Results

We show the following results.

- We can insert or delete integers in a subset sum problem, and maintain the solution to the problem in $O(\ell \log \ell \log n)$ time per update and $O(n\ell)$ space. No fully dynamic algorithm was previously known.

- If we are given a limited ability to predict future deletions, in that for any k we can find a set of $O(k)$ values containing all values that could be deleted in the next k steps, we can improve our bounds to $O(\ell \log n)$ time per update and $O(n+\ell)$ space. This *semi-online* model was previously considered by Dai et al. [2] who gave an $O(\ell\sqrt{n})$ time bound for the semi-online subset sum problem.

- We can find the minimum range balanced cut in $O(m + n^2 \log n)$ time and $O(n)$ space. Previously $O(m + n^{2.5})$ time was known [2].

All our algorithms assume a random access model of computation in which arithmetic operations on $\max(\log n, \log \ell)$-bit integers can be performed in constant time. Some such restriction on the size of integers is necessary: if we were to allow arithmetic on unbounded (or even $n\ell$-bit) integers we could update the dynamic subset sum problem in constant time, and find the minimum range balanced cut in time $O(m + n \log n)$. For our balanced cut algorithm, the edge weights may be real numbers as long as we can compare any pair of weights or weight differences in constant time.

3 Reduction from Cuts to Subset Sums

We now show how to solve the minimum range balanced cut problem by using an algorithm for dynamic subset sums. In fact we give two reductions: to online, semi-online, and offline subset sums. Similar reductions are used by Dai et al. [2], however we are able to speed up the semi-online reduction by generalizing up our previous offline minimum spanning forest algorithm [4].

For any cut in our graph G, the minimum and maximum weight cut edges are part of the minimum and maximum spanning trees respectively. Therefore the range of the cut can be determined solely by considering edges in these two trees, and ignoring all other edges. Following Dai et al. [2], we perform a preprocessing step in which we compute these two trees, and remove all other edges from G. This can be done in time $O(m + n \log n)$, dominated by the other steps of our algorithm. Thus from now on we can assume without loss of generality that G has $O(n)$ edges.

3.1 The Matrix of Feasible Ranges

Given two edge weights α and β, we define the *range* $[\alpha, \beta]$ to consist of all edges with weight $\alpha \leq w \leq \beta$. A range is *feasible* for the balanced cut problem if there is a partition of $V(G)$ into two equal-cardinality subsets A and B such that all edges crossing the partition are in the range. A range is *minimal* if it is feasible and no other feasible range contains a subset of its edges. We will find the minimum balanced cut by listing all minimal ranges and finding the minimum difference $\beta - \alpha$ among these ranges.

Lemma 1. *We can test whether a range is feasible by solving a subset sum problem.*

Proof: We consider the graph G' formed by removing from G all edges in the range. G' has connected components with vertex sets of cardinalities n_i satisfying $\sum n_i = n$. The range is feasible iff there is some subset of the components with n_i summing to $n/2$; this can be tested as a subset sum problem with $\ell = n/2$. □

Sort the $m = O(n)$ edges of G by weight, producing a sequence of weights $w_1, w_2, \ldots w_m$. We define the $m \times m$ square $0-1$ matrix F to have the values $F_{i,j} = 1$ if range $[w_i, w_j]$ is feasible, and $F_{i,j} = 0$ otherwise.

Lemma 2. *The 0's and 1's in F form connected regions, separated from each other by a monotone path.*

Proof: This follows simply from the fact that as one decreases i or increases j, the range $[w_i, w_j]$ becomes larger, so it can not change from being feasible to being infeasible. □

The minimal ranges $[w_i, w_j]$ we seek are those for which $F_{i,j} = 1$ and $F_{i-1,j} = F_{i,j-1} = 0$. If we follow the path separating the 0' and 1's in F, we can find these ranges as the points at which the path turns a corner.

3.2 Reduction to Online Subset Sum

As discussed above, we have reduced the minimum range balanced cut problem to one of following a monotone path in a 0-1 matrix. We next discuss how to use dynamic subset sum algorithms to speed up the evaluation of values $F_{i,j}$ in such a path. We consider a dynamic process, in which after evaluating an entry $F_{i,j}$ we will next want to evaluate some $F_{i',j'}$ where i' and j' are within one step of i and j respectively.

Lemma 3. *Suppose we can solve the dynamic subset sum problem in time $f(n, \ell)$ per insertion or deletion. Then we can step from entry to adjacent entry of $F_{i,j}$ as described above, in time $O(\sqrt{n} + f(n, n))$ per step.*

Proof: Each step consists of adding or removing a single edge from a range, and hence deleting or inserting an edge in the graph G' described in Lemma 1. If we remove an edge from G', we may possibly split a single connected component in two; this causes one deletion and two insertions in the subset sum problem of Lemma 1. Similarly if we insert an edge we may connect two components of G' and cause two deletions and one insertion in the dynamic subset sum problem. We can keep track of the components of G' and their sizes by using a dynamic minimum spanning forest algorithm, in time $O(\sqrt{n})$ per update [5]. □

We can then simply follow the monotone path separating 0's and 1's in F, to find the corners of the path and hence (as discussed above) the minimal ranges, thereby solving the minimum range balanced cut problem.

Lemma 4 (Dai et al. [2]). *Assume we have an algorithm for performing a dynamic subset sum algorithm, in time $f(n,\ell)$ per insertion or deletion. Then we can solve the minimum range balanced cut problem in total time $O(m + n^{1.5} + nf(n,n))$.*

Proof: As noted earlier we use minimum and maximum spanning trees to remove unnecessary edges from G, reducing the number of edges to $O(n)$.

We use Lemma 3 to step from entry to entry of matrix F and find in each row i the minimum value of $j = j(i)$ for which $F_{i,j} = 1$. Note that, as discussed in Lemma 2, $j(i+1) \geq j(i)$. We begin at $(i,j) = (0,0)$. Then, whenever we have $F(i,j) = 0$, we increase j by one; whenever $F(i,j) = 1$ we instead increase i by one. The values $j(i)$ are those at which we found $F(i,j) = 1$. After at most $2m = O(n)$ steps we will have found all such values, so by Lemma 3 the time for this stage is $O(n^{1.5} + f(n,n))$.

We next list the minimal ranges as those values $(i, j(i))$ for which $j(i-1) < j(i)$. We compute $w_j - w_i$ for each such range, and find the minimum such difference. This gives the minimum range of a balanced cut; we then use a static subset sum algorithm to find the actual cut. □

3.3 Reduction to Semi-Online Subset Sums

Dai et al. [2] noted that there is a simple structure to the sequence of graph operations performed in Lemma 4: edges are inserted and deleted in increasing order of length. Only the order of insertions relative to deletions is dynamic. Since each graph operation gives rise to $O(1)$ subset sum operations, the predictability of the graph update sequence means we can also make some predictions about the subset sum update sequence. However the predicted structure is less simple: we cannot, for instance, predict the order in which deletions will occur in the subset sum problem.

In order to exploit the predictability of these two update sequences, Dai et al. introduce the following model. A dynamic algorithm in which elements are inserted or deleted from a set is said to be *semi-online* if we are given an oracle that can at any time perform the following prediction task: given an integer k, return a *prediction set* of size $O(k)$ with the property that any of the elements currently in the input set and deleted within the next k steps must belong to the prediction set.

This semi-online model generalizes other standard types of dynamic algorithms: *Incremental algorithms* (allowing insertions only) can be handled by returning an empty prediction set. *Offline algorithms* (in which the entire update sequence is known) and *semi-dynamic algorithms* (in which only the order of the deletions is known) can be handled by returning a prediction set consisting of the next k deletions in the known sequence.

The relevance of semi-online algorithms to our situation is, first, that the dynamic minimum spanning forest algorithm used to determine component sizes clearly fits this model (indeed, it also fits the more specialized semi-dynamic model discussed above). Second, the subset sum problem used in the reduction can also be solved with a semi-online algorithm:

Lemma 5 (Dai et al. [2]). *The sequence of updates to the dynamic subset sum problem of Lemma 4 is semi-online, with a prediction set oracle that takes time $O(k \log n)$.*

Proof: Given a value k, we examine the next k edges to be inserted and the next k edges to be deleted in G'. We return a prediction set consisting of the at most $4k$ values corresponding to components of G' containing an endpoint of one of these edges. These components can be found by representing the minimum spanning forest of G' by a dynamic tree data structure [9]. □

As a consequence, if we can perform semi-online subset sums in time $f(n, \ell)$ per update, we can solve the minimum range balanced cut problem in time $O(m + n^{1.5} + nf(n, n))$. We improve this somewhat by generalizing the offline dynamic minimum spanning forest algorithm of [4] to apply in the present situation.

Lemma 6. *We can solve the version of the dynamic minimum spanning forest problem in which the sequences of insertions and deletions are both* *known, but in which the way these two sequences are merged may vary online, in amortized time $O(\log n)$ per update.*

Proof: The algorithm of [4] uses the fact that, for any sequence of k updates to a graph, we can find a set S of $O(k)$ edges such that any change to the MST caused by one of the updates involves only edges in that set. S includes all edges updated in the given sequence. The remaining edges in S are found by examining the MST T of the graph G, performing all deletions of edges of T in the sequence and collecting the edges of $G - T$ added to replace them, and performing all insertions in the sequence and collecting all edges of T removed to replace them. Once we have calculated S, we can reduce the problem within this sequence to an equivalent one of size k by removing from G all edges in $G - T - S$ and contracting all edges in $T - S$.

By repeating this contraction process at $O(\log n)$ levels, with smaller and smaller values of k, one reduces the problem to graphs of size $O(1)$, for which it is trivial. This contraction process does not involve the ordering among the k updates, and works equally well if (as in the Lemma) we know a set of $2k$ updates such that any of the k actual updates is taken from this set. Thus the algorithm works not just for the offline problem considered in [4] but for the problem considered here.

We omit the details, and refer the reader to our original paper [4], as this result is not necessary for our eventual $O(m + n^2 \log n)$ bound. □

Lemma 7. *Assume we have an algorithm for performing a semi-online subset sum algorithm, which takes time $f(n, \ell)$ per insertion or deletion using a prediction set oracle with time bound $O(k \log n)$. Then we can solve the minimum range balanced cut problem in total time $O(m+n \log n+n f(n, n))$.*

4 Online Subset Sums

In this section we describe an algorithm for maintaining the solution to a subset sum problem, as values are inserted or deleted. Since we could solve a static problem by inserting its n values one at a time, and since the best static algorithm takes time $O(n\ell)$, the best we could hope for in a dynamic algorithm would be time $O(\ell)$ per update; our algorithm takes time $O(\ell \log \ell \log n)$, within a small polylogarithmic factor of optimal. We can either test whether there is a subset adding to a given value ℓ, or find the largest $\ell' < \ell$ for which such a subset exists, in the same amount of time.

We assume that we can perform arithmetic on $\max(\log n, \log \ell)$-bit integers; this many bits are required simply to represent the input values. Note that some such assumption is needed for the problem to be nontrivial: if we were allowed unlimited precision we could maintain the value $P = \prod(1 + 2^{nn_i})$ in constant time per update, and recover the number of solutions to the subset sum problem for a given ℓ as $P/2^{n\ell} \bmod 2^n$.

Given a set N of values n_i, define $B_i(N)$ to be 1 if some subset of the values has sum i, and 0 otherwise. We let $B(N)$ denote the vector of ℓ such values, for i in the range $1, 2, \ldots, \ell$.

Lemma 8. *Given two collections of values S and T which are disjoint (in the sense that no element of S belongs to T or vice versa, although elements in different collections may have the same value) we can compute $B(S \cup T)$ from the two vectors $B(S)$ and $B(T)$, in time $O(\ell \log \ell)$.*

Proof: For notational convenience we extend the length of the vectors by one, and define $B_0(S) = B_0(T) = 1$ (the empty subset of any set sums to zero). Let $A_i = \sum_{j+k=i} B_j(S)B_k(T)$. Then A_i counts the number of ways i may be formed as the sum of values j and k that can respectively be formed as sums of subsets of S and T. We then let $B_i(S \cup T) = 1$ if $A_i > 0$, and $B_i(S \cup T) = 0$ otherwise. A_i is simply a vector convolution, so it can be computed in time $O(\ell \log \ell)$ by the fast Fourier transform algorithm. It is then trivial to compute B_i from A_i in time $O(\ell)$. □

Theorem 1. *We can solve the online subset sum problem in $O(\ell \log \ell \log n)$ time per update, using $O(n\ell/\log \ell)$ space.*

Proof: Our data structure consists of a balanced binary tree, with n leaves corresponding to the n input values. For each internal node v we store the vector $B(N(v))$, where $N(v)$ denotes the set of leaves descending from v. The solution to the subset sum problem can then be found simply by looking at the stored value for $B_\ell(N(r))$ where r is the root of the tree.

After each update we recompute the vectors at $O(\log n)$ tree nodes. The vector $B(\{x\})$ at a leaf can be constructed easily: $B_x(\{x\}) = 1$ and all other values of $B(\{x\})$ are 0. At each internal node v with children u and w we compute $B(N(v)) = B(N(u) \cup N(w))$, in time $O(\ell \log \ell)$ by Lemma 8.

The vector of $O(\ell)$ bits at each of $O(n)$ tree nodes can be compressed into $O(\ell/\log \ell)$ integer values, giving the claimed space bound. □

Note that the steps in which we produce the vectors B_i by "throwing away" information from the A_i are necessary: if we left them out, and instead stored at each node the convolution of the vectors at its children, the bits per value would exceed the $O(\log \ell)$ bound.

By using this data structure with Lemma 4 we can solve the minimum range balanced cut problem in time $O(m+n^2 \log^2 n)$ and space $O(n^2/\log n)$. However, we can get better time and space bounds, and avoid the complexity of the FFT-based online algorithm, by instead using the semi-online algorithm to be described next.

5 Semi-Online Subset Sums

We solve the semi-online subset sum problem using a recursive blocking technique similar to that in our previous offline minimum spanning forest algorithm [4]: we partition the sequence of update events that we will be handling into a sequence of *blocks.* Each block contains a number of events that is a power of two, and is partitioned into two smaller blocks, its *children.* Thus the overall sequence of blocks has the structure of a complete binary tree. The *length* $L(b)$ of a block b is defined to be the number of update operations occurring within it. If n denotes the total number of operations performed in the algorithm, the sum of the lengths of the blocks at a given level of the tree is exactly n, and since there are $O(\log n)$ levels the sum of the lengths of all blocks is $O(n \log n)$. In what follows, we let b' denote the parent of any block b, i.e. the unique block containing b with $L(b') = 2L(b)$.

When we perform an update operation, we will do some computations within each of the blocks for which that update is the first operation, in order from longer blocks to shorter blocks. These operations compute an *active set* $X(b)$ for each block b, with the property that any deletion in b is in the active set. This is essentially just the *prediction set* for which the semi-online problem gives us an oracle, so $|X(b)| = O(L(b))$. However we make the minor restriction that $X(b) \subset X(b')$. We can safely remove any values from the prediction set of b that are not in $X(b')$, since we know they can not actually be deleted in block b.

We let $N(b)$ denote the set of all subset sum values present in the problem before the first operation of b that are not members of $X(b)$. For each block b we will maintain a vector $B(N(b))$ (with the same definition of $B(S)$ as in the previous section).

Lemma 9. *We can compute $B(N(b))$ from $B(N(b'))$ in time $O(L(b) \cdot \ell)$.*

Proof: Let $S = N(b) - N(b')$; then S consists of $X(b') - X(b)$, together with those values inserted in the other child of b' (if b is the second child) and not part of $X(b)$. Thus $|S| = O(b)$. Let the values in S be denoted $v_1, v_2, \ldots v_k$. We compute the vectors $B^1, B^2, \ldots B^k$, where $B^i = B(N(b') \cup \{v_1, v_2, \ldots v_i\}$. Each B^i can be calculated in time $O(\ell)$ from B^{i-1} and v_i since B^i_j is one iff either B^{i-1}_j or $B^{i-1}_{j-v_i}$ is one. The desired vector $B(N(b)) = B^k$. □

Theorem 2. *We can solve the semi-online subset sum problem, with a prediction oracle that takes time $kp(n)$, in amortized time $O((p(n)+\ell)\log n)$ per update, using $O(\ell)$ space.*

Proof: We compute the values $B(N(b))$ for each block b as described above. At the leaves of the tree of blocks (those blocks containing only a single operation), $|X(b)| = O(1)$ and we can compute $B(S)$ for the actual set of values in the problem at that time by computing $B(N(b) \cup X(b))$ in time $O(\ell)$ using the same technique as Lemma 9.

The algorithm performs $O(L(b) \cdot \ell)$ work computing $B(N(b))$ for each block, and $O(L(b)p(n))$ work constructing $X(b)$. Over a sequence of n operations it takes $O((p(n) + \ell) \sum L(b)) = O((p(n) + \ell)n \log n)$ total time.

The space bound follows since at any point in time we need to remember the values of $O(\log n)$ vectors $B(N(b))$, each of which can be stored as $O(\ell/\log n)$ integer values. □

Theorem 3. *We can solve the minimum range balanced cut problem in time $O(m + n^2 \log n)$.*

Proof: We apply Lemma 7, using $f(n,n) = O(n \log n)$ by Theorem 2 (with $p(n) = O(\log n)$). □

References

[1] P. M. Camerini, F. Maffioli, S. Martello, and P. Toth. Most and least uniform spanning trees. *Discrete Applied Math.* 15 (1986) 181–187.

[2] Y. Dai, H. Imai, K. Iwano, and N. Katoh. How to treat delete requests in semi-online problems. *Proc. 4th Int. Symp. Algorithms and Computation*, Springer-Verlag LNCS 762 (1993) 48–57.

[3] Y. Dai, H. Imai, K. Iwano, N. Katoh, K. Ohtsuka, and N. Toshimura. A new unifying heuristic algorithm for the undirected minimum cut problem using minimum range cut algorithms. *Discrete Applied Math.*, to appear.

[4] D. Eppstein. Offline algorithms for dynamic minimum spanning tree problems. *J. Algorithms* 17 (1994) 237–250.

[5] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Proc. 33rd IEEE Symp. Foundations of Computer Science* (1992) 60–69.

[6] Z. Galil and B. Schieber. On finding most uniform spanning trees. *Discrete Applied Math.* 20 (1988) 173–175.

[7] N. Katoh and K. Iwano. Efficient algorithms for minimum range cut problems. Networks 24 (1994) 395–407.

[8] S. Martello, W. R. Pulleyblank, P. Toth, and D. de Werra. Balanced optimization problems. *Operations Research Lett.* 3 (1984) 275–278.

[9] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comp. Sys. Sci.* 24 (1983) 362–381.