Edge Contraction in Global Min Cuts

April 7, 2003

1 Bucket Sorting Algorithm

Input: A set of numbers x_i, where $1 \le i \le n$ and $0 < x_i < 1, \forall i$. Also x_i are uniform i.i.d. (independent identically distributed).
Algorithm:

- Divide $[0, 1]$ into αn buckets ($\alpha < 1$) of equal size.
- For all x_i, compute the bucket number $(b(x_i)) = \lfloor \frac{x_i}{n\alpha} \rfloor$
- Sort each bucket separately using any sorting algorithm, e.g. insertion sort

Analysis: Let $N(B_j)$ denote the number of elements in the j^{th} bucket. Because of independence and uniformity, we have $E[N(B_j)] = 1/\alpha$. Hence, the expected time to sort denoted by $E[TS]$ is given by

$$\begin{aligned} E[TS] &= E[\textstyle\sum_{j=1}^{\alpha n} N(B_j)^2 + O(n)] && \text{(Since insertion sort takes } O(n^2)\text{)} \\ &\le 2n/\alpha + O(n) && \text{(using } E[x^2] = E[x]^2 + \sigma^2\text{)} \\ &= O(n) \end{aligned} \tag{1}$$

□

The analysis beats the lower bound of $O(n \log n)$ obtained by using comparisons even in the case where all permutations are equally likely. This can not be possible in general and the problem in the analysis lies in the computation model that uses the floor function and assumes constant time for it. We now show a general result that establishes that such a case (A randomized algorithm beating the lower bound) will never be possible unless the model of computations are different.

Theorem 1 *Yao's MinMax Theorem:*
Let $T_R(n)$ be the worst case expected running time of a randomized algorithm over an input of size n. Also, if $T_A(n)$ is the lower bound on the expected running time of an algorithm for a fixed distribution, then $T_R(n) = T_A(n)$.

Proof: We only proof the that $T_R(n) \geq T_A(n)$.
If a randomized algorithm R uses $r(n)$ random bits for inputs of size n and s_i is a bit string of length $r(n)$, then let $T_R(I, s_i)$ denote the run time of R for string s_i over input I. Also, let $\alpha(j)$ denote the input distribution and $\beta(i)$ denote the distribution of the random string s_i. We then compute $T_{j,i}$ as the running time of $R(I_j, s_i)$. We then have the following

$$\begin{aligned} T_A(n) &= \min_i \sum_j \alpha(j) T_{i,j} \\ T_R(n) &= \max_j \sum_i \beta(i) T_{i,j} \end{aligned}$$

Now, consider R for some string s. We have

$$\sum_j \alpha(j) T_{j,s} \geq T_A(n) \tag{2}$$

$$\sum_s \beta(s) \sum_j \alpha(j) T_{j,s} \geq T_A(n) \tag{3}$$

$$\sum_j \alpha(j) \sum_s \beta(s) T_{j,s} \geq T_A(n) \tag{4}$$

$$\exists j' \quad s.t. \sum_s \beta(s) T_{j',s} \geq T_A(n) \tag{5}$$

$$T_R(n) \geq \quad T_A(n) \tag{6}$$

$\square$

2 Computing Global Min Cuts

2.1 Related Work

The simplest way to obtain a global min cut is to use the duality between s-t max flow and s-t min cut. Hence, a natural strategy is to compute max flows for all s-t pairs and output the minimum of all those s-t flows to get the global min cut. Unfortunately, the algorithm is very expensive, i.e., $O(n^2)$ times Cost of flow Algorithm, which is $O(n^5)$.

Gomory-Hu Method uses flow computations and shows that all the $O(n^2)$ s-t flows need not be computed and only $O(n)$ computations would suffice. However, the method is fairly complex.

2.2 A simple Randomized Algorithm

Algorithm:

- Choose an edge (u, v) randomly and apply contract(u, v) to that edge.
- Repeat the above step $n - 2$ times.
- Output the edges that are still left as the min cut

contract(u, v): Merge vertices u and v into w such that all neighbours of u and v are now neighbours of w.
Analysis: Given a specific mincut C, we estimate the probability that C is not outputted. If C is not the output, then it means that none of the edges of C has ever been contracted. Let $E(i, C)$ denote the fact that an edge of C is contracted in i^{th} iteration. Also, $\tilde{E}(i, C)$ denote that no edge of C is contracted in any of the i iterations. We then have

$$\begin{array}{ll} \text{Prob}(E(1, C)) & \leq 2/n \\ \text{Prob}(E(i, C)|\tilde{E}(i-1, C)) & \leq 2/n_i \\ \text{Prob(no edge of C is contracted)} & \geq \Pi_{i=1}^{n-2} 1 - 2/n_i \\ & \geq 1/{}^{n}C_2 \end{array}$$

Hence, the probability that the min-cut would be the output is $O(1/n^2)$. Repeating the above algorithm $O(n^2)$ times would ensure that the min cut is expected to be the output at least once. Thus, the expected running time is $O(n^4)$ assuming that each step can be performed in $O(n)$ time. □

Theorem 2 *The time taken to pick an edge E randomly and contract the edge E is $O(n)$.*

Proof: We first present a method to pick an edge randomly in $O(n)$ time. The selection works as follows.

- Select a vertex v at random with probability $= \frac{deg(v)}{\sum_{u \in V} deg(u)}$
- Select an edge (v, w_i) at random with probability $= \frac{\#edges(v, w_i)}{\sum_{j=1}^{k} \#edges(v, w_j)}$

where $\#edges(u, v)$ denotes the number of edges between u and v and k is the number of neighbours of v.
Hence, the probability of choosing any edge (v, w_i) is given by

$$\begin{aligned} &= \frac{\#edges(v, w_i)}{\sum_{u \in V} deg(u)} + \frac{\#edges(w_i, v)}{\sum_{u \in V} deg(u)} \\ &= \frac{\#edges(v, w_i)}{\#edges} \end{aligned}$$

Thus, the above method picks edges randomly with equal likelihood. Also, note that using an adjacency matrix M for storing the graph where M_i, j denotes the number of edges between v_i and v_j allows us to merge vertices i and j in O(n) time. $\square$

2.3 An efficient algorithm for global min cut

One may note that the probability that the min cut survives $O(n/c)$ contractions is $O(1/c^2)$. However, the probability falls rapidly after the n/c contractions. The original algorithm is then modified as follows.

- Perform n/c contraction steps to get a partial result R'.
- Make a copy of R'.
- Recursively compute the min cut on both the copies independently.

Analysis: Taking the value of $c = \sqrt{2}$, we have the following recurrence for the running time of a run of the algorithm.

$$T(n) = 2T(n/\sqrt{2}) + O(n^2) \tag{7}$$

Solving the recurrence would yield that $T(n) = \tilde{O}(n^2)$, where $\tilde{O}$ denotes a *poly* log factor. We show that the bound of $O(n^2 \log_{\sqrt{2}} n)$ satisfies the induction. Let us assume that $T(n/\sqrt{2}) \leq c(n/\sqrt{2})^2 \log(n/\sqrt{2})$. We now verify that $T(n) \leq cn^2 \log n$.

$$\begin{aligned} T(n) &\leq 2cn^2/2 \log n/\sqrt{2} + n^2 \\ &= cn^2 \log n - cn^2 \log(\sqrt{2}) + n^2 \\ &\leq cn^2 \log n \end{aligned} \tag{8}$$

Note that that taking that log with base $\sqrt{2}$ allows us to make the last assertion directly.

Similarly, the probability $P'(n)$ that the min cut would survive in any of the copies of the run is given by

$$P'(n) = 1/2(1 - (1 - P'(n/\sqrt{2}))^2) \tag{9}$$

Again solving the recurrence would yield $P'(n) = 1/\log n$. We can again check the result using induction. We assume that $P'(n/\sqrt{2}) \geq \frac{1}{c \log n/\sqrt{2}}$ and show that $P'(n) \geq \frac{1}{c \log n}$.

$$\begin{aligned} P'(n) &\geq 1/2(1 - (1 - 1/(c\log(n/\sqrt{2})))^2) \\ &= 1/2(1 - 1 - \frac{1}{c^2 \log^2(n/\sqrt{2})} + \frac{2}{c\log(n/\sqrt{2})} \\ &= 1/2(\frac{2(c\log n + c\log\sqrt{2})}{c^2\log^2 n - c^2 \log^2\sqrt{2}} - \frac{1}{c^2\log^2(n/\sqrt{2})} \\ &= \frac{1}{c\log n} + 1/2(\frac{2c\log\sqrt{2}}{\log(\sqrt{2}n)\log(n/\sqrt{2})c^2} - \frac{1}{c^2\log^2(n/\sqrt{2})}) \\ &\geq 1/(c\log n) \text{ if } 2c > \frac{(\log_{\sqrt{2}} n)+1}{(\log_{\sqrt{2}} n)-1} \end{aligned} \tag{10}$$

Hence, we only need $\log n$ runs of the algorithm and expect to see the min cut. Thus, the running time is $\tilde{O}(n^2)$. $\square$