*Paolo Codenotti*

# Repeated Min-Cut

## The Problem

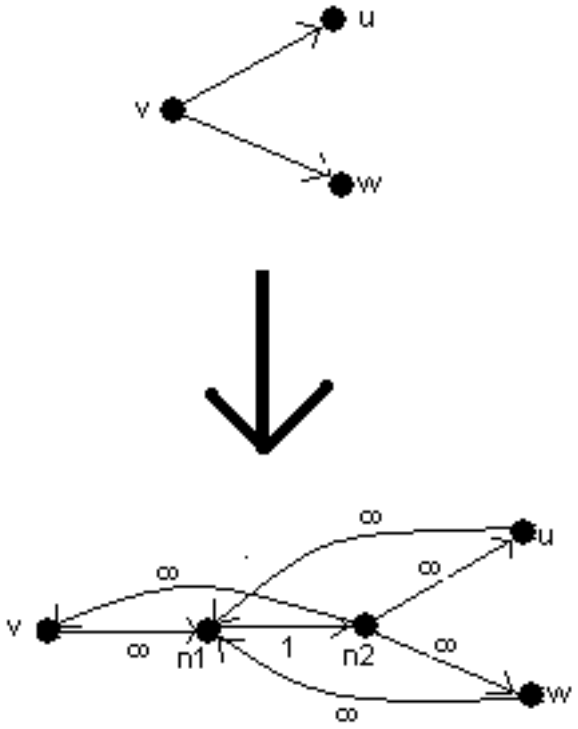
We are considering the problem of partitioning a graph into partitions of a given size, such that the interconnect between partitions be minimal. This problem is known to be NP-Hard, even in the case of only two partitions. However, there are known heuristics which give approximate solutions to the balanced bipartitioning problem. Most heuristics are based on using the min-cut max-flow theorem repeatedly, in clever ways (see[HB97], [YW94], [HHK95], [Kri84]). Since we realized that the max-clique heuristic was not fast enough on large inputs, and is not very effective if these are sparse, we implemented a repeated min-cut based heuristics to apply to the original graph, which is often sparse. So we can apply max-clique to smaller, denser subgraphs, where the algorithm is more efficient, both in terms of time, and in terms of finding an optimal solution. The min-cut algorithm we implemented is based on the one described in the paper [YW94]. The technique used in this paper is to construct a network flow from the input graph, such that a min-cut in the network flow will correspond to a min-net cut in the input graph. Altough a min-net-cut does not correspond to a min-cut, it will be a cut with little interconnect, especially if the graph is sparse and the degree of each vertex is bounded.

## The construction

In order to use the max-flow algorithm, in [YW94] they create a weighted directed graph $G'$ from the original graph $G$, such that a min-cut partition in $G'$ corresponds to a min-net-cut partition in $G$.

From the input directed graph we construct a flow network as follows (see fig. 1): for each net $net(n)$ of node $n$, we add two additional nodes $n1$ and $n2$ in the new graph. connections are made as follows: there is an edge of weight $\infty$ from $n$ to $n1$, and edge of weight 1 from $n1$ to $n2$, edges of weight $\infty$ from $n2$ to each of the outneighbors of $n$ in the original graph, and from each of these to $n1$, plus an edge of weight $\infty$ from $n2$ to $n$. One can show that the graph $G\prime$ is strongly connected, it has size at most constant times the size of the original graph, and a min cut in $G\prime$ corresponds to a min-net-cut in $G$.

Fig. 1




This figure shows how the graph $G\prime$ is constructed from $G$.

## The algorithm

Therefore, our algorithm will work as follows:

`Procedure 1:`

1. `Randomly choose s and t source and target from the set of vertices`
2. `Construct the flow network as described above`
3. `Find a min-net-cut from s to t, which defines partitions` $X$ `and` $\overline{X}$ , where "$s\epsilon X$, and $t\epsilon\overline{X}$
4. `If` $(1-\varepsilon)PS \leq w(X) \leq (1+\varepsilon)PS$ `then:`
   (a) `return X as one of the partitions;`
   (b) `remove X from G;`
   (c) `goto 1;`

5. `If` $(1-\varepsilon)PS \leq w(\overline{X}) \leq (1+\varepsilon)PS$ `then:`
   (a) `return` $\overline{X}$ `as one of the partitions;`
   (b) `remove` $\overline{X}$ `from G;`
   (c) `goto 1;`
6. `Otherwise :`
   (a) `Let` $G = X$ `, and repeat the algorithm;`
   (b) `Let` $G = \overline{X}$ `, and repeat the algorithm;`

Where PS is the target Partition Size we're trying to achieve. We also have to be careful to simply return one partition: the whole graph in case the input graph has weight $\leq (1+\varepsilon)PS$

Once we have a min-net-cut partition of our graph $G$, we have two cases: either one of the partitions is within a factor of epsilon of the right partition size, or neither are. In the first case, we simply output the partition which is close enough to the desired partition size, and then we repeat the algorithm on the rest of the graph. Otherwise, the sizes of the two parts are both greater than the target partition sizes, we simply run the algorithm on $X$ and $\overline{X}$ separately.

Step 3 is implemented by using the construction described above to construct a flow network $G\prime$, then finding a max-flow in $G\prime$, and the relative cut of minimum capacity, which corresponds to a min-net-cut in $G$. However, this can be done incrementally, as we reduce the size of the graph. More explicitly:

`Procedure 2:`

1. `While there exists an additional augmenting path from s to t`
2. `Increase the flow value along the augmenting path; (end while)`
3. `Mark all nodes reachable from s through an augmenting path;`
4. `Return the marked nodes as` $X$;

The max-flow algorithm works by increasing the flow along the edges until there are no more augmenting paths. Then, we let $X$ be the set of edges reachable from s, and $\overline{X}$ the complement of X. It is easy to see that $s \epsilon X$, and $t \epsilon \overline{X}$.

The max-flow algorithm has time complexity $|V||E|$ over a graph $G(V, E)$, where $V$ is the set of vertices, and $E$ is the set of edges. The construction of $G\prime$ from $G$ only increases the size by a constant factor. If we use `Procedure 2` for step 3, it is shown in [YW94] that the time complexity is $O(|V||E|)$ if we repeat the algorithm only on one of $X$ or $\overline{X}$. By doing it on both, we at most increase the time complexity by a fact of $|V|/PS$ (recall, PS is the target partition size). Thus at most we get a time complexity of $O(|V|^2|E|)$, since PS is a constant.

**References**


HB97 S. Hauck, G. Borriello, "An evaluation of Bipartitioning Techniques", *IEEE Trans. on CAD*, Vol. 16, No. 8, pp. 849-866, August 1997.

YW94 H. Yang, D. F. Wong, "Efficient Network Flow based Min-Cut Balanced Partitioning", *International Conference on Computer-Aided Design*, pp.50-55, 1994.

HHK95 L. Hagen, J. H. Huang, A. B. Kahng, "On implementation choices for Interactive Improvement Partitioning Algorithms", *European Design Automation conference*, pp. 144-149, 1995.

Kri84 B. Krishnamurthy, "An improved Min-Cut Algorithm for partitioning VLSI Networks", *IEEE Transactions on Computers*, Vol. C-33, No.5, pp. 438-446, May 1984.