On Multilevel Circuit Partitioning

Sverre Wichlund
sverre.wichlund@alcatel.no
Alcatel Telecom Norway
(formerly at Norwegian University of Science and Technology)

Einar J. Aas
aas@fysel.ntnu.no
Norwegian University of Science and Technology

Abstract

Multilevel partitioning approaches for circuit partitioning has been shown to be powerful [14, 45, 1, 2]. In this paper we improve the excellent multilevel partitioning algorithm in [2] by taking edge-frequency information [44] into account during coarsening/uncoarsening, and to break ties. In addition, the uncoarsening phase is guided by an adaptive scheme which adds flexibility to the number of levels in the uncoarsening phase. We apply our algorithm to 13 benchmark circuits and achieve an improvement in min-cut and average min-cut values of up to 8.6% and 34.6% respectively, compared to the method in [2], at no extra runtime. Furthermore, our algorithm provides results of very stable quality. This positions our algorithm as current state of the art in multilevel circuit partitioning.

Keywords: partitioning, clustering, multilevel, heuristic.

1 Introduction

A graph $G = (V, E)$ consists of $n$ nodes and $m$ edges, a hypergraph $H = (V, E_h)$ consists of $n$ nodes and $m_h$ hyperedges, where a hyperedge $e$ in $E_h$ is defined by a subset of nodes $V_e$ in $V$. A graph is thus a special case of a hypergraph in which $|V_e| = 2$. Graph or hypergraph partitioning is the task of dividing the node set $V$ in $G$ or $H$ into sets of a predetermined size such that the sum of the weights on edges or hyperedges connecting nodes in different sets is minimized. Let the multiway partitioning problem be defined as follows [33]:

**Instance:** A hypergraph $H = (V, E_h)$ with $n$ nodes, a node weight function $w : V \rightarrow \mathbb{N}$, an edge cost function $c : E_h \rightarrow \mathbb{N}$, a number $k \in \mathbb{N}$ of modules, maximum module sizes $B(i) \in \mathbb{N}$, minimum module sizes $b(i) \in \mathbb{N}$, $i = 1, \ldots, k$

**Configurations:** All $k$-way partitions $P^k = \{V_1, \ldots, V_k\}$ of $V$, with $V_1 \subseteq V$, $V_i \neq \emptyset$, and $V_i \cap V_j = \emptyset$ for $i, j \in \{1, \ldots, k\}$, $i \neq j$ and $\bigcup_{i=1}^k V_i = V$

**Solutions:** All $k$-way partitions $P^k = V_1, \ldots, V_k$

such that $b(i) \leq w(V_i) \leq B(i)$ for $i = 1, \ldots, k$

Minimize: $c(P^k) = 1/2 \sum_{i=1}^k \sum_{e \in E_{ext,i}} c(e)$ where $E_{ext,i} = \{e \in E_h | e \cap V_i \neq \emptyset, e \setminus V_i \neq \emptyset\}$ is the set of external edges of module $V_i$

The cost $c(P^k)$ is called the cutsize, and the set of external edges is said to contribute to the cut. In the special case where $k = 2$ we have the bipartitioning problem.

The size constraints on modules are often formulated as $w(V_i) - (1 - r) \leq w(V_i) - \frac{w(V_i)}{2} - (1 + r)$, $i \in \{1, 2\}$, in this case where $r$ is a balance factor. The set of external edges is referred to as the cut and an edge in this set is said to be cut. Furthermore, if $|V|$ is even, the vertices are uniformly weighted and $w(V_i) = w(V_i)/2$, $i = 1, 2$, we have the bisection problem. In this paper we look at the bipartitioning problem. Other objective functions may be chosen as well. Define $\text{span}(e)$ to be $s$ if $e$ connects exactly $s$ modules, $s \geq 2$. 0 otherwise, then we minimize $|\{e \in E | \text{span}(e) \geq 2\}|$ [49, 40]. Note that for $k = 2$ the objective functions are equal.

Many problems in computer science are conveniently formulated as a graph or hypergraph partitioning problem. Electrical circuits with multiple-pin nets are readily modelled as hypergraphs. In VLSI placement a divide and conquer approach is taken where the circuit is hierarchically divided into smaller components using graph partitioning [33]. The formulation also finds applications in rapid prototyping where the goal typically is to partition to a minimal number of FPGAs (Field Programmable Gate Arrays) under different constraints such as available number of pins and routing resources [30, 31, 32, 13, 47, 10]. Another application area is design for testability of VLSI circuits where the circuit is partitioned into smaller parts to facilitate testing [43, 42]. Unfortunately the problem is computationally intractable, which, when taking its numerous applications into account, makes it the subject of substantial research efforts. Clever heuristics have been developed which provide suboptimal solutions in reasonable computing time given that the problem is generally NP-hard [33]. The methods may be classified in several ways. From an algorithmic viewpoint, one simple classification may be as follows based on whether we already have a solution (of unacceptable solution quality) to our problem or not:

- **Constructive methods** that constructively build a partition of our input graph. Techniques based on geometric embeddings of the circuit as in [9, 5, 3, 34] belong to this category. In [35] a framework for approximation
algorithms is proposed that guarantees polylogarithmic time optimal solutions to graph and hypergraph partitioning problems. The much celebrated max-flow min-cut theorem of Ford and Fulkerson [19] will provide the optimal solution to partitioning a graph into two subsets of unconstrained sizes in polynomial time if applied directly, but the resulting subsets tend to be of very uneven sizes [11]. However the theorem may be used in combination with other methods as in [48, 34].

- **Iterative methods** that iteratively try to improve an existing partition of our input graph. One well known bi-partitioning technique in this category is due to Kernighan & Lin [28] which was extended to multi-pin nets in [41]. The method iterates over all node pairs in two equal size partitions and a pairwise swapping of nodes takes place. In [18, 15] the method is further refined by using clever data structures. Krishnamurthy [29] takes into account lookahead information concerning subsequent node moves between the two partitions. In [40] the last method is extended to multiway partitioning. In [6] the multiway partitioning problem is approximated and solved as a transportation problem.

We may also apply well known methods often used in conjunction with combinatorial optimization like Simulated Annealing, Genetic Algorithms, and Taboo Search, see [38] for an overview.

Due to its popularity (fast and easy to implement) the original FM-algorithm [18] has been subject to improvements [16, 17, 29, 40, 21]. The method has been successfully incorporated into other approaches constituting hybrids as in [20, 36, 8]. To improve the performance of the FM-algorithm, it is frequently combined with clustering as a preprocessing step [12] often referred to as the two-phase approach, and extended to the multilevel approaches in [22, 1, 14, 23, 24, 27, 45, 2]. In this paper we improve the excellent multilevel partitioning algorithm in [2] by

1. guiding the coarsening phase by augmenting their connectivity function to take the edge-frequency\(^1\) into account. We also break ties in a similar manner.

2. Not following the cluster hierarchy directly during uncoarsening; instead we select a fraction of the clusters at each level for uncoarsening according to the connectivity calculated in the coarsening phase. This is controlled by an adaptive scheme [46].

2 Motivation

The iterative improvement methods easily get trapped in local minima as the size of the input circuit grows. In [28] they empirically found the probability of terminating with the optimal solution to be approximately \(2^{-n/30}\) for their test circuits where \(n\) is the number of nodes in the input circuit. One solution to the problem is to coarsen the input circuit as a preprocessing step prior to these methods, thus reducing problem complexity and keeping strongly connected components unseparated during partitioning [42, 29, 14, 4]. This may be accomplished by applying clustering to the input circuit before partitioning. We actually partition our input circuit represented by a hypergraph \(H(V, E)\) into \(c\) clusters, \(P^c = \{C_1, \ldots, C_c\}\), \(c \gg k\). Then we build a constructed circuit represented by a coarsened hypergraph \(H'(V', E')\) with the clusters as nodes, i.e. \(V' = \{C_1, \ldots, C_c\}\) and \(E' = \bigcup_{e \in E} \{C_i | |e| > 1 \text{ and } e \cap C_i \neq \emptyset\}\). This coarsened hypergraph reduces the problem size from \(n\) to \(c\). Besides, it will be characterized by a higher density and larger minimal node degree than our original hypergraph representation \(H(V, E)\) of the circuit [4]. This is known to improve the performance of partitioning heuristics based on the method in [18]. Lengauer [33] conjectures that for graphs with high density and large minimal node degree, the solution space will contain few local optima that are not global optima.

Rather than partitioning the contracted circuit represented by \(H'\) followed by a final run on the original circuit by expanding \(H'\) without altering the partition, which is denoted as the two-phase approach, more sophisticated schemes exist. In the multilevel approaches we build a hierarchy of successively coarser hypergraphs \(H_i(V_i, E_i), i \in \{0, \ldots, m\}\), by applying the principle above on our original netlist hypergraph \(H_0(V_0, E_0)\), then on \(H_1(V_1, E_1)\) ...

The coarsening is halted when the topmost hypergraph \(H_m(V_m, E_m)\) contains a suitable number of nodes (say \(|V_m| \approx 50\)). Then an iterative improvement heuristic is applied to \(H_m(V_m, E_m)\) and the obtained solution is projected back (uncoarsened) to the next level \(H_{m-1}(V_{m-1}, E_{m-1})\) by replacing each node in \(V_m\) by its constituent nodes to obtain \(V_{m-1}\) and restoring the edges (note that the actual partition is not altered). This process continues until partitioning on \(H_0(V_0, E_0)\) is performed. The multilevel approach has several advantages over the two-phase approach. With many levels of uncoarsening we allow the refinement process to proceed more slowly, improving the partitioning at each level\(^2\). In addition, in early stages of the algorithm many nodes are moved. Bad node moves during these stages may be corrected later, when successively fewer nodes are moved at a time. Interestingly enough, in [39] they proceed in the opposite direction, beginning with the original graph and successively coarsening and partitioning towards the coarsest graph. Their motivation is that coarsening should not proceed too fast, comparing the process to simulated annealing\(^3\). An analysis of multilevel graph partitioning applied to a restricted class of graphs may be found in [25].

From the scientific computing community, two efficient multilevel partitioning packages have emerged, namely the Chaco package [24] and the Metis package [20]. The latter is based on a greedy weighted matching algorithm on which the work in [1, 2] is based. In [2] the authors achieve excellent results by introducing a matching ratio \(\alpha, 0 < \alpha \leq 1\), where only a fraction \(\alpha\) of the nodes at each level is matched. In this way the height of the cluster tree is increased making the uncoarsening more smooth and leaving more room for improvement, see [2] for details. This seems promising, especially on the larger circuits.

We adapted this scheme in our approach. However, we take the edge-frequency [44] into account when matching by augmenting the connectivity function used during matching. The concept of edge-frequency is to keep track of the number of times each edge appears in the cut for solutions with a cut size below a certain threshold. In our implementation we store the number of times each edge has been cut in the \(h\) best solutions encountered so far\(^4\). Then we favour

\(^{1}\)The concept of edge-frequency was originally proposed in [44].

\(^{2}\)Note that a local optimum at level \(i\) not necessarily is a local optimum at level \(i-1\) since the uncoarsening change the structure of the problem, i.e. the local optimum may not be a sink in the search graph after uncoarsening, see [37, 33] for necessary definitions.

\(^{3}\)Their coarsening is based on information gained during the sequence of node moves performed by the improvement heuristic.

\(^{4}\)This is easily implemented by storing a bit-vector of \(h\) bits for
matching of nodes which are connected by low-frequency edges. The motivation is that edges of high frequency should most likely appear in the cut of high quality solutions and thus not matched. We are motivated by the work in [7] where they observe that the best local optima tends to be “close” to the other local optima for several instances of the TSP (Travelling Salesman Problem) and the graph bisection problem. This is also the motivation behind the work in [1] where the authors devise a genetic framework to improve the Metis performance. In the event of breaking ties we select the move which results in the largest gain $g_e$ defined as $g_e = \sum_{i \in N} f(e) - \sum_{i \in N} f(e_i)$, where $N_i$ are nets that will appear in the cut and vice versa and $f(e)$ is the frequency of edge $e$. During matching the nodes are visited in random order and for each node $v_{\pi(i)}$ we select an unmatched node $v$ which has the highest connectivity to $v_{\pi(i)}$ according to the following connectivity function

$$
connect(v, u) = \frac{1}{w(v)w(u)} \sum_{e \in \{e: v \in E(e), u \in E(e)\}} \exp(-\gamma f(e)) \tag{1}
$$

The factor $\exp(-\gamma f(e))$ disfavors nodes incident to edges of high frequency. The damping factor $\gamma$ is experimentally determined, see next section. During uncoarsening we select a fraction $\alpha$ of the clusters at the current level for uncoarsening. The cluster with the lowest connectivity between its constituents according to equation 1 is split. This process is controlled by an adaptive scheme based on the number of previous passes performed by the iterative improvement heuristic, see appendix A. In figure 1 we have depicted our algorithm, call it $MLAF$. The function $BuildHierarchy()$ builds a cluster hierarchy using matching on the input hypergraph with parameters $\alpha$ and $T$ as in [2].

$RandomBipartition()$ constructs a random initial bipartition. The function $FMPartition()$ applies the FM-algorithm on the current partition $P_{level,i}^2$ and returns a refined partition in $P_{level,i}^2$. We use the FM-algorithm [18] with a LIFO data structure [21]. The function $adapt()$ adjusts $\alpha$ based on the number of passes performed by $FMPartition()$ in the previous step, see appendix A for details. $topLevel()$ returns the topmost level in the cluster hierarchy that will result after uncoarsening. $Uncoarsen()$ uncoarsens the current hypergraph by selecting a fraction $\alpha$ of the clusters at the current level for splitting. $Project()$ projects the current solution $f_{level,i}^2$ back onto the uncoarsened hypergraph $H_{level,i}$. Last, if the best solution found so far $P_{best}$ is updated, the function $FrequencyUpdate()$ is called to update the edge-frequency of all nets cut.

3 Experimental results

The algorithm $MLAF$ in figure 1 was implemented in C under Solaris 2.5.1 on a Pentium Pro. We selected 13 benchmark circuits from the CBL (Collaborative Benchmarking Laboratory) for comparison of $MLAF$ to the method in [2]. The results reported are bipartitioning results with size constraints on partitions given by $w(V_i) \leq w(V_0) \leq \frac{w(V_i)(1-r)}{w(V_i)(1+r)}$, $i \in \{1, 2\}$ for $r \leq 0.1$ and unit area on nodes.

In table 1 we have listed the min-cut and average min-cut values obtained by $MLAF$ for various values of $h$ and $\gamma$. $\alpha$ was set to 0.5, and $max\_runs$ to 100 runs. Average min-cut values are of interest, because an improvement in the average value implies a better quality if only a small no. of runs is affordable. We also observed a lower variance.

The results for $MLF$ and $MLC$ are taken from [2]. We compare our results to $MLF$ and $MLC$ [2] where in $MLC$ the FM partitioner uses the CLIP scheme [17] which is shown to improve $FM$ by 18% on the average (using LIFO buckets).

We do not report CPU-times. We did not implement the CPU-time reducing modifications in [24] in our algorithm $MLAF$. The authors’ clever method of avoiding recomputing gains was alone reported to reduce the overall CPU-time by 30% to 95% on their test circuits. However, taking this into account and estimating the relative performance between the hardware in [2] and ours turns out to give $MLAF$ the same CPU-time performance as $MLC$ [2]. In some situations a minor improvement in cut-size is critical, while for others the CPU-time may be the main issue. There is a tradeoff between the amount of CPU-time spent and the quality of the resulting partition. For a cost-value analy-
sis of the subject, refer to [45]. From table 1 we observe that the best overall performance of $ML_{AF}$ was achieved for $(h, \gamma) = (5, 0.5)$. Except for the industry circuits and 19ks we improve the min-cut values of all circuits compared to $ML_I$ and $ML_c$ with up to 6.8% for $s15850$.

for $\alpha = 0.5$ as well. However, for the large benchmarks (industry3 and golem3) the best results are achieved for $\alpha = 0.33$. And as the benchmarks get larger, the best results seem to be achieved for a relatively smaller $\alpha$. In table 3 we have listed the min-cut and average min-cut values obtained by $ML_{AF}$ when the adaptive scheme is disabled, $\alpha$ was 0.5 and $(h, \gamma) = (5, 0.5)$.

We also improve the average min-cut values of almost all circuits by at most 25% for circuit $s38417$ compared to $ML_c$ and by 22% for circuit $s13207$ compared to $ML_f$. For average min-cut, $ML_{AF}$ was slightly inferior with respect to $golem3$ compared to $ML_c$ and $ML_f$. Results for $(h, \gamma) = (5, 1.0)$ are not listed in table 1. In this case, we saw 98.22(103.18).

For $s35532$ the average min-cut obtained by $ML_{AF}$ is the same as the min-cut obtained by $ML_c$ and better than the min-cut obtained by $ML_f$. For $s35534$ and industry2 the average min-cut obtained by $ML_{AF}$ is equal to the min-cut obtained by $ML_f$. Again, for avgml the average min-cut is better than the min-cut obtained by $ML_c$.

Table 3: Min and average cut for our algorithm obtained for 100 runs with respect to $biomed$ and $golem3$ compared to $ML_f$ and $ML_c$.

Here we observe a drop in performance for $ML_f$ with respect to $biomed$ and $golem3$ compared to $ML_f$ and $ML_c$.

Table 2: Min and average cut for our algorithm obtained for 100 runs for various values of $\alpha$, $(h, \gamma) = (5, 0.5)$, adaptive=on. Improvement is over $ML_f$ and $ML_c$ for $\alpha = 0.33$.

In table 2 we have listed the min-cut and average min-cut values obtained by $ML_{AF}$ for $\alpha = 1.0$, 0.5 and 0.33, and $ML_f$ and $ML_c$ for $\alpha = 0.33$. Here we observe that $ML_{AF}$ is inferior to $ML_f$ and $ML_c$ by 29.7% and 25.5% accordingly with respect to average min-cut for $biomed$. In turn, for industry2 and s38417, $ML_{AF}$ is better than $ML_c$ by 34.6% and 25% accordingly with respect to the average min-cut. For $s13207$ the min-cut obtained by $ML_{AF}$ is better than the min-cut obtained by $ML_f$ and $ML_c$ by 8.6%. For avgml, the average min-cut obtained by $ML_{AF}$ is better than the min-cut obtained by $ML_f$.

With respect to min-cut, from table 1 and 2 we observe that the overall performance is best for $\alpha = 0.5$. When average min-cut is compared, the best performance is achieved causing a drop in overall performance compared to adaptive=on, see table 1. When it comes to min-cut values, it seems that the circuits 19ks, $biomed$, and industry3 have some properties that are not captured by $ML_{AF}$. One possible explanation may be that the “good” solutions for these circuits do not tend to be particularly “close” in the sense described in [7]. Thus equation 1 confines the search to a possibly “average good” region in the solution space, having the effect of rather limiting the search for the best solution. This is further evidenced from table 4 where we have listed the results obtained by $ML_{AF}$ for edge-frequency information used in coarsening/uncoarsening and breaking ties ($f_{bi} = 1, f_{ci} = 1$), information used to break ties only ($f_{bi} = 1, f_{ci} = 0$) and no information used ($f_{bi} = 0, f_{ci} = 0$). We observe that $ML_{AF}$ performs better on 19ks and industry3 when it comes to min-cut for $f_{bi} = 0, f_{ci} = 0$ (which is equivalent to $\gamma = 0$) compared to $f_{bi} = 1, f_{ci} = 1$. From table 1, column 6 $(h, \gamma) = (10, 0.1)$, the same situation...
is observed ($\gamma = 0.1$ means less damping of frequent edges than $\gamma = 0.5$).

\[
\text{Table 4: Min and average cut for our algorithm obtained for 100 runs. } \alpha = 0.5, (h, \gamma) = (5, 0.5), \text{ adaptive} = \text{on}. \text{ Edge frequency used in the coarsening/uncoarsening phase and to break ties. Improvement is edge frequency information used in both cases over no information used, i.e. column 2 over column 4.}
\]

We have not included the GM approach [1] in the tables presented. However, with respect to column 3 in table 1, $M_{Af}$ improved the min-cut and average min-cut values overall by 19.9% and 30.0% respectively compared to GM.

It was observed that the cluster sizes (the node sizes in $H_n$) were quite unbalanced for most of the circuits, especially the circuits $19k$, $biomed$, $industry2$ and $industry3$. We believe that the cluster sizes should be relatively balanced to achieve the best performance. Unbalanced cluster sizes imply that some clusters may be relatively large compared to others. Moving these large clusters will most likely violate size constraints on partitions. Thus unbalanced cluster sizes may lead to relatively fewer possible moves during partitioning of the coarser hypergraphs, having a negative impact on performance. By modifying the connectivity function in such a way that matching of smaller nodes is further emphasized, the nodes in $H_n$ should be more balanced. Specifically, the connectivity function is now:

\[
\text{connectivity function: } \sum_{v \in E} \exp(-\gamma f(v))
\]

In equation 2, $\omega > 1$ is chosen to give preference to nodes with relatively smaller sizes. Experiments performed for $\omega > 1$ indeed demonstrated more balanced node sizes in $H_n$.

\[
\text{Table 5: Min and average cut for our algorithm obtained for 100 runs for various values of } \omega, (h, \gamma) = (5, 0.5), \text{ adaptive} = \text{on}. \alpha = 0.5. \text{ Improvement is over } M_{Ic} \text{ and } M_{Af} \text{ for } \omega = 1.2. (f_{A} = 0, f_{cl} = 1).
\]

$w = 1.2$ together with the devised adaptive scheme during uncoarsening. In this case, $M_{Af}$ produced results of very stable quality at no extra CPU-time compared to $M_{Ic}$ and $M_{Ic}$. The overall improvement in min-cut and average min-cut values was 21.2% and 12.3% respectively compared to $M_{Ic}$ and 2.0% and 11.7% respectively compared to $M_{Ic}$.
This positions algorithm $ML_{AF}$ as current state of the art in multilevel circuit partitioning. In future work it would be interesting to establish a relation between structural properties of the input circuit and the values to use for $h, \gamma$. Furthermore, we would like to incorporate the CLIP scheme [17] into $ML_{AF}$ to see if any further improvement would result.

A. Adaptive adjustment of $\alpha$

In [2] it was observed that a slower coarsening, i.e. more levels of uncoarsening significantly improved solution costs for the larger benchmarks. Stated another way, the speed of coarsening apparently depends on problem size. This is also studied in [46]. By selecting a fraction of the clusters for uncoarsening at each level controlled by an adaptive scheme, we hope to achieve a better overall performance by adjusting the number of uncoarsening levels to match the problem size. In this way we should also avoid superfluous runs of the partitioning heuristic caused by an $\alpha$ too small for the problem size. This could make the current local optimum a local optimum on the next level of uncoarsening by improper restructuring of the search graph. We propose the following simple adaptive adjustment of the uncoarsening speed:

Let $\alpha$ depend on the number of passes $p_i, i = 1, \ldots, t$ done by the partitioning heuristic during the last $t$ levels of uncoarsening, such that $\alpha$ is increased if the number of passes performed during the last $t$ levels in some sense is insufficient. More formally, define a set of weights $\Gamma = \{\gamma_1, \ldots, \gamma_t\}$ and a function

\[ \tau(i) = \begin{cases} 0 & p_i > 1 \\ 0.1 & p_i = 1 \\ 1 & p_i = 0 \end{cases} \]  

for $i = 1, \ldots, t$.

where $p_i$ is the number of passes performed by the partitioning heuristic $i$ levels of uncoarsening earlier, and finally we define $\alpha$ to be

\[ \alpha = C_1 \sum_{i=1}^{t} \gamma_i \cdot \tau(i) + C_0 \]  

The constants $C_0$ and $C_1$ have to be chosen such that $0 < \alpha \leq 1.0$. By selecting proper values for $t, \Gamma, C_0$ and $C_1$, this scheme may adjust $\alpha$ in such a way that we achieve the proper degree of restructuring of the search graph regardless of the level of uncoarsening, leading to more effective use of the CPU time. We implemented the proposed scheme, with the algorithm $ML_{AF}$, and performed experiments on the benchmark circuits with $t = 5, \gamma_i = 0.2, i = 1, \ldots, t$, $C_0 = 0.01$ and $C_1 = 0.99$. Thus $\alpha$ is a weighted sum of the passes completed by the partitioning heuristic during the last $t$ levels of unclustering.

References


