To appear in Proc. Design Automation Conf., 1996

A Probability-Based Approach to VLSI Circuit Partitioning[*]

Shantanu Dutt and Wenyong Deng
dutt@everest.ee.umn.edu and wydeng@lsil.com
Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455

Abstract

Iterative-improvement 2-way min-cut partitioning is an important phase in most circuit partitioning tools. Most iterative improvement techniques for circuit netlists like the Fidducia-Mattheyses (FM) method compute the gains of nodes using local netlist information that is only concerned with the immediate improvement in the cutset. This can lead to misleading gain calculations. Krishnamurthy suggested a lookahead (LA) gain calculation method to ameliorate this situation; however, as we show, it leaves considerable room for improvement. We present here a probabilistic gain computation approach called PROP[1] that is capable of capturing the global and future implications of moving a node at the current time. Experimental results show that for the same number of runs, PROP performs much better than FM (by about 30%) and LA (by about 27%), and is also better than many recent state-of-the-art clustering-based partitioners like EIG1, WINDOW, MELO and PARABOLI by 15% to 57%. We also show that the space and time complexities of PROP are very reasonable. Our empirical timing results reveal that it is appreciably faster than the above clustering-based techniques, and only a little slower than FM and LA, both of which are very fast.

1. Introduction

VLSI circuit partitioning is an important problem in design automation of VLSI chips and multichip systems. It is used to reduce VLSI chip area, reduce the component count and the number of interconnects in multiple-FPGA implementation of large circuits or systems, facilitate efficient parallel simulation of circuits, facilitate design of tests for digital circuits and reduce timing delays. A commonly used approach to solving the partitioning problem is to initially obtain a min-cut 2-way partition of the circuit in which it is partitioned into two subsets such that the number of nets connecting nodes in different subsets is minimized. Furthermore, there is generally a balance criterion with respect to the number of nodes or components that can be placed in any one subset; for example, equal number of nodes or components in both subsets, or no more than 55% of the nodes in any subset. Each subset is further partitioned into two smaller subsets with a minimum cut, and so forth until we have recursively partitioned the circuit into either a prespecified number k of subsets (thus obtaining a k-way partition), or until each subset has very few nodes, say, 2 or 3, in it.

[*]This work was supported partly by NSF grant MIP-9210049

[1]There is a paper in ICCAD-95 that presents a method also called PROP. Ours is a different method for a different purpose. We came up with the name PROP (for PRObabilistic Partitioner) much before we noticed the same name in the ICCAD-95 paper.

Let a circuit C be represented by a hypergraph or netlist $G = (V, E)$, where V is the set of nodes that represent components of the circuit and E the set of hyperedges that represent the nets of the circuit. Each hyperedge or net connects two or more nodes together. We will represent a net n_i as a set of the nodes that it connects. We denote the number of nodes in V by n, the number of hyperedges in E by e, the average number of nets that a node is connected to by p, the average number of nodes that a net connects by q, and the average number of neighbors of a node by $d = p(q-1)$—a node u is said to be a *neighbor* of another node v, if u and v are connected by a common net. A *k-way partitioning* of G is a set of subsets $P^k = \{V_1, V_2, \ldots, V_k\}$ of V such that each $v \in V$ belongs to exactly one V_i. Let r_1 and r_2 be two numbers between 0 and 1 such that $r_1 \leq r_2, r_1 \leq 1/k$ and $r_2 \geq 1/k$. Then, an *(r_1, r_2)-balanced k-partition* of G is defined as a k-partition in which $r_1 \leq |V_i|/n \leq r_2$ for each subset V_i of P^k. When $k = 2$, $r_1 = 1 - r_2$; however, for $k \geq 3$ there is no obvious relation between r_1 and r_2 (except $r_1 \leq r_2$). We assume that all nodes have unit size; the balance criterion is easily changed to reflect size constraints on the subsets when this is not the case. The *cutset* of a k-way partitioning is defined as

$$E_{cut} = \{n_t \in E \mid \exists u, v \in n_t \text{ s.t. } u \in V_i, v \in V_j, i \neq j\}$$

In other words, the cutset E_{cut} is the set of nets that connect nodes belonging to different subsets of P^k. The *cost* or *size* of the cutset of P^k is defined as $cost(P^k) = \sum_{i=1}^{k} c(n_t)$, where $n_t \in E_{cut}$, where $c(n_t)$ is the *cost* or *weight* of net n_t that depends on the criterion we are trying to optimize when partitioning a circuit. For example, if our goal is to minimize layout area of the circuit on a VLSI chip, then $c(n_t)$ is the width of n_t. On the other hand, if we are trying to minimize timing, then a critical net is assigned more weight than a non-critical one to ensure that the length of critical or near-critical nets are kept as short as possible [8]. The min-cut problem is to obtain a P^k so that its cost is minimized. Recursive 2-way partitioning is an efficient and popular approach to obtaining k-way partitions for $k > 2$ [3, 7, 13, 14]. We will thus be concerned here with the 2-way min-cut partitioning problem. Since the problem is NP-complete, a number of approximate schemes have been proposed. These include iterative improvement methods [3, 6, 9, 10], simulated annealing [12] and clustering-based techniques [1, 2, 7, 11, 13, 14]. In iterative improvement methods, we start with a random 2-way partition of the circuit, and iteratively improve it by either swapping pairs of nodes between the subsets, or moving one node at a time between them so that the cutset size is reduced. Clustering-based methods try to find natural clusters in the circuit and then assign them to the two subsets thereby automatically reducing the cutset size. Iterative improvement methods are also sometimes used as either pre- or post-processing phases for clustering as in [1, 14, 13]. Thus mincut partitioning using iterative improvement techniques is a fundamental tool for obtaining good VLSI cell placement.

A number of iterative min-cut methods for graph or hypergraph partitioning have been previously proposed [3, 6, 9, 10]. Kernighan and Lin proposed the well-known KL graph partitioning algorithm using pair swaps to improve a random initial 2-way partition [9]. Schweikert and Kernighan extended this algorithm to netlists [3]. Fidducia and Mattheyses gave a similar algorithm for netlists that alternately moves single nodes between the two subsets of the partition as opposed to swapping node pairs at a time; this makes the process more time efficient [6]. They also proposed efficient data structures to obtain fast partitions. However, these data structures assume that nets have unit costs or weights. If this is not the case, as when a circuit is partitioned to reduce timing delays [8], then the partitioning process is much slower. The node gain calculations in both the Schweikert-Kernighan (SK) and Fidducia-Mattheyses (FM) algorithms use only local netlist information, and this quite often gives inaccurate indications of the potential improvement that can be obtained by moving a node. In [10], a lookahead (LA) gain calculation was employed to capture more global information. It gives better partitions than FM, but requires large amounts of memory, as will be discussed shortly, thus rendering it infeasible or very slow (due to frequent page swaps) for use on medium- to large-size circuits.

In this paper, we present a precise probabilistic gain calculation method PROP (for PRObabilistic Partitioner) that captures more global and futuristic information than FM or LA. We show by a simple example that PROP selects better nodes to move than either FM or LA. We also run tests on circuit netlists from the ACM/SIGDA benchmark, which show that PROP performs an average of 30% better than FM and 27% better than LA. Comparison of PROP to some of the more recent clustering-based techniques like EIG1 [7], WINDOW [1], PARABOLI [11] and MELO [2] show that PROP also performs significantly better (by 15% to 57%) than them.

The rest of this paper is organized as follows. In Sec. 2. we discuss two previous iterative improvement methods FM and LA, and thereby set the stage for discussing the PROP technique in Sec. 3., where we also derive its time and space complexities. Section 4. presents the cutsets obtained by PROP on standard circuit netlists. These results are compared to those obtained using FM, LA and WINDOW when the balance criterion is 50-50% ($r_1 = r_2 = 0.5$), and with EIG1, MELO and PARABOLI when the balance criterion is 45-55%. Conclusions are in Sec. 5..

2. Previous Iterative-Improvement Methods

Here we briefly describe the iterative improvement process, the node-gain calculations used in the FM and LA algorithms, and their shortcomings.

Assume that there are $n = 2l$ nodes in the hypergraph G, and that the initial partition is $\{V_1, V_2\}$ with $|V_1| = |V_2| = l$. When partitioning a hypergraph or netlist, the gain of a node is not as apparent as in the case of a graph. The FM netlist partitioner uses a simple extension of the Kernighan-Lin node gain calculation (used for graph partitioning) [6]. For each node u, let $I(u)$ be the set of nets to which u is connected that lie **entirely** in u's current subset, and $E(u)$ be the set of nets that belong to the cutset and for which u is the **only** node connected to them in u's partition. Then the *gain* of u is given by

$$\mathit{gain}(u) := \sum_{n_i \in E(u)} c(n_i) - \sum_{n_j \in I(u)} c(n_j) \qquad (1)$$

This gain definition of a node is the **immediate** decrease in the cutset cost if it is moved to the other subset. The partitioning process proceeds by determining the next best node u_i to move in the ith step as follows. The "unlocked" node (initially all nodes are unlocked) with the maximum gain in either subset is determined. If the balance criterion on the two subsets can be maintained after moving this node from its current subset to the other one, it is chosen as the node u_i. Otherwise, the unlocked node with the maximum gain in the other subset is chosen as u_i. Node u_i is then moved to the other subset and "locked", and the gains of all its neighbors are updated. The node gain $\mathit{gain}(u_i)$ is inserted in an ordered set S. After all nodes are moved and locked, all prefix sums $S_k = \sum_{t=1}^{k} \mathit{gain}(u_t)$ are computed, $1 \leq k \leq n$, and a p is determined for which the partial sum S_p is maximum. The set of nodes that are actually moved are then, $\{u_1, \ldots, u_p\}$. This whole process is called a *pass*. A number of passes are made until the maximum partial sum is 0 or negative. The resulting cutset cost is a local minima with respect to the initial partitions V_1 and V_2. Empirical evidence shows that the number of passes required to achieve this local minima is two to four [9, 6].

As mentioned above, $\mathit{gain}(u)$ is the immediate decrease (or increase if it is negative) in the cutset size that we will obtain on moving u to the other subset. There will very likely be a number of nodes with the same or similar gains, and ideally a tie between them should be broken by also considering the *potential* gain associated with each node, viz., the decrease in the cutset that is not immediate but has a good likelihood of occurring in the future. Krishnamurthy developed a scheme that estimates the potential gain by using a "lookahead" gain vector for each node [10]. Consider a gain vector $\mathit{gain}(u)[k]$ of node u with k elements, and assume that $u \in V_1$. The ith element of the vector is defined as the number of nets connected to u to which $i-1$ other nodes of V_1 are connected minus the number of nets connected to u that have $i-1$ nodes of V_2 connected to them. A gain vector **a** is said to be greater than a gain vector **b** if either $\mathbf{a}[1] > \mathbf{b}[1]$ or if there exists an $i < k$ such that $\mathbf{a}[j] = \mathbf{b}[j]$ for all $1 \leq j \leq i$ and $\mathbf{a}[i+1] > \mathbf{b}[i+1]$. A node u is said to have a larger gain than node v, if $\mathit{gain}(u) > \mathit{gain}(v)$. In practice, a lookahead value of $k = 2$ to 4 gives the best results, and consistently gives better results than FM. However, the memory requirement of the LA method is $\Theta(p_{max}^k)$, where p_{max} is the maximum number of pins on a node. Thus for circuits with medium to large connectivity for some nodes, it can become infeasible or very slow to even use a lookahead of 3.

Figure 1 illustrates the differences between the probability-based method, and the LA and FM algorithms in computing node gains. For simplicity, only nodes in V_1 are considered, and all nets have cost 1. FM will give nodes 1, 2 and 3 a gain of two, 10 and 11 a gain of one, and all the other nodes shown a gain of -1. Since node 1, 2, 3 have the same gain, FM can very well choose to move node 1 first. However, it is easy to see from the figure that both nodes 2 and 3 are better candidates to move first, since moving either of them will make it easier for either 8 and 9 or 10 and 11 to be moved later and thus obtain a greater reduction in the cutset (i.e., nodes 2 and 3 have a better potential gain than node 1). The LA algorithm is able to do better than FM in this regard. Assuming a lookahead of 3 (LA-3), the node gain vectors for nodes 1, 2 and 3 are as follows: $\mathit{gain}(1) = (2,0,0)$, $\mathit{gain}(2) = (2,0,1)$ and $\mathit{gain}(3) = (2,0,1)$; see Fig. 1(a). Thus by this gain calculation, node 2 and 3 are correctly portrayed as being better than node 1. A little thought will also convince us that node 3 is a better candidate to move than node 2. This is because both nodes 10 and 11 that node 3 is connected to via net n_{11} are themselves better candidates for moving than nodes 8 and 9 that are connected to node 2 via net n_{10}—moving nodes 10 and 11 (after node 3 has been moved) reduces the cutset by three nets (n_5, n_8 and n_{11}), while moving nodes 8 and 9 (after node 2 has been moved), results in a cutset reduction by only one net n_{10}. Thus, neither the FM nor the LA methods are able to completely distinguish between nodes 1, 2 and 3 as shown by their gain values in Fig. 1(a) (increasing the lookahead of LA beyond 3 does not change this) in spite of the fact that intuitively the distinction between them is obvious. The primary reason for this is that neither method is able to accurately predict the future state of a net. The probability-based method PROP is able to see the likelihood of future events much better, and is described next.

3. The Probabilistic-Gain Based Partitioner

The probability-based method PROP determines the best node to be moved at any point in the partitioning stage using much more global and futuristic information than LA or FM. We associate with each node u a probability $p(M(u))$ (abbreviated as $p(u)$) of the

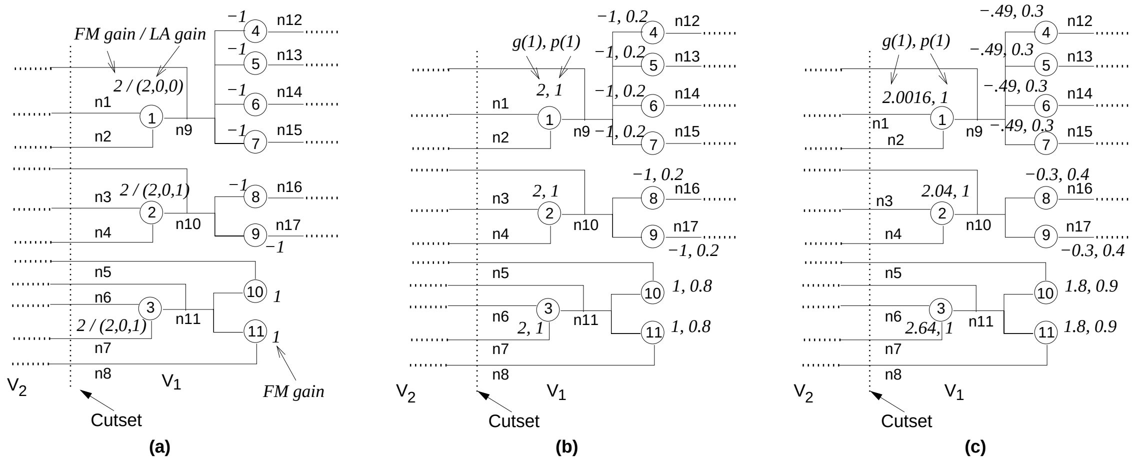

Figure 1: Illustration of the improvement of PROP over the LA and FM algorithms: (a) FM and LA-3 gains (the latter is only shown for nodes 1, 2 and 3). (b & c) Probabilistic gains and node-move probabilities after (b) the first, and (c) the second iteration of node gain and probability calculation.

event $M(u)$ that u will be *actually moved* to the other subset in the current pass of the partitioning process[2]. From this probability, we compute the probabilistic (or potential) gains (hereafter only termed gain) $g(u)$s of the nodes, which gives us an accurate indication of the benefit of moving them to the other subset. The obvious question is how to obtain the node probabilities $p(u)$s in the first place, to which the answer is that they are computed from their respective (probabilistic) node gains—higher the gain, higher is a node's probability of being actually moved to the other side. However, we need to start off this process of chicken-and-egg interdependency between gains and probabilities somewhere, and we do so by first determining a rough estimate of the $p(u)$s in one of two ways. In the first method, at the beginning of a pass, we "blindly" assign all nodes the same probability p_{init} of, say, 0.8. In the second method, we first determine the *deterministic gains gain*(u)s of nodes as given by Eqn. 1 for the FM method. From these deterministic gains, we determine the initial probabilities of the nodes (functions for determining probabilities from gains are discussed in Sec. 3.2.). This method gives us reasonable first-cut probability estimates.

Once we have this initial probability (by either of the above two techniques), we compute the (probabilistic) gains of nodes as explained shortly. From these gains, we recompute the node probabilities, and from these we obtain more accurate node gains. This cycle continues for a few iterations (we have used 2 iterations in our implementations) and we obtain more and more refined node probabilities and gains. After this initial process is completed, we move nodes with the best gains between the two subsets as in other iterative improvement methods. After each move, we update the node gains and probabilities as explained in Sec. 3.4.. Also, with each move we note the *immediate gain* achieved, which is the number of nets that are removed from the cutset minus the number of nets that are introduced in it on that move. At the end of the current pass, we actually make moves to that point which gives the maximum prefix immediate gain, as in FM and LA. Note that the probabilistic gain is useful in determining which nodes to move that will ultimately yield the most improvement in the cutset, though the immediate gain of that move might be small or even negative; due to moving such a node at the present time, we expect that a future move will have a large immediate gain. It is this determination of probabilistic gains of nodes, initially, and after every move, that is the key to obtaining much better performance than previous deterministic- or

[2]Note that a node is *actually moved* from its original subset to the other one in an iterative-improvement scheme like KL, FM and PROP, if its "virtual" move lies within the range of the maximum prefix gain that is computed after all nodes are (virtually) moved. Nodes beyond this range are not actually moved and revert back to their original subset.

```
**Procedure** PROP($G$);
/* $G$ is the hypergraph to be partitioned */
**Begin**
  1. Either randomly or using some clustering techniques partition $G$ into
     two equal (or almost equal) sized subsets $V_1$ and $V_2$.
  2. **Repeat** the following steps (each iteration of these steps is a *pass*)
     **until** the gain $G_{max}$ obtained after a pass is less than or equal to 0.
  3. For each node $u$, either let $p(u) = p_{init}$, or determine the $p(u)$s from
     the nodes' deterministic gains.
  4. For each node, iterate through the gain calculation steps using Eqns. (3)
     and (4), and $p(u)$ calculation two times
  5. **Repeat** Steps 6-8 **until** all nodes are locked, or no more moves can be
     made to meet balance criterion.
  6. Select the node $u$ with the best gain in either subset to be moved
     to the other subset if the $(r_1, r_2)$-balance condition is not violated.
     Otherwise, move the best-gain node $u$ from that subset for which the
     balance condition is not violated.
  7. Note the immediate gain of the current move.
  8. Lock all nodes moved in the current iteration, and update their un-
     locked neighbors as described in Sec. 3.4..
  9. Calculate the prefix sums of the immediate gains of all moves made
     and note the maximum $G_{max}$ of these sums.
 10. **If** $G_{max} > 0$ **then begin**
       **If** $G_{max}$ correspond to the $p$th move **then** Actually
       make only the first $p$ moves
     **end**
     **else exit**
**End**.
```

Figure 2: The probabilistic-gain based partitioning algorithm PROP.

immediate-gain based iterative improvement methods like FM and LA. The partitioning algorithm is described formally in Fig. 2. We state the following theoretical result proved in [5].

Theorem 1 *Given any set of node probabilities, the sample space of events representing subsets of nodes that are actually moved in a pass of an iterative-improvement process is a valid probability space, i.e., for any event E in this space (1) $P[E] \geq 0$; (2) $P[E \cup F] = P[E] + P[F]$ if $E \cap F = \emptyset$; and (3) $P[\Omega] = 1$, where Ω is the set of all possible events.*

3.1. Calculating Probabilistic Node Gains

We now describe how probabilistic node gains are calculated from node probabilities. For each node, we calculate its gains corresponding to each net that it is connected to; its total gain is the sum of these net gains. We first define the following concepts for a net. A net n_i is said to be *locked* in V_1 (V_2) if any node in it is locked in V_1 (V_2). A net is *locked in the cutset*, if it is locked in both V_1 and V_2. For node-gain calculation, there are two cases depending on whether the net in question is currently in the cutset or not.

3.1..1 Net in Cutset

Let $u \in V_1$ be connected to net n_i, which belongs to the cutset. We denote the set $n_i \cap V_r$ by $n_{i,r}$, $r = 1, 2$. Let $n_i^{1\to 2}$ ($n_i^{2\to 1}$) be the event in which net n_i is removed from the cutset by moving all nodes in $n_{i,1}$ ($n_{i,2}$) to V_2 (V_1). We define

$p(n_i^{1\to 2}|u) =$ (Probability of n_i being removed from the cutset by moving all nodes in $n_{i,1}$ to V_2 given that u has been moved)

$p(n_i^{2\to 1}|u^c) =$ (Probability of n_i being removed from the cutset by moving all nodes in $n_{i,2}$ to V_1 given that u is not moved)

Then the gain $g_{n_i}(u)$ of u corresponding to net n_i is defined as

$$g_{n_i}(u) = c(n_i)[p(n_i^{1\to 2}|u) - p(n_i^{2\to 1}|u^c)] \qquad (2)$$

The rationale for the negative term in the above expression is that moving u precludes the event $n_i^{2\to 1}$ from occurring, and thus eliminates the possibility of removing n_i from the cutset in that manner. Using conditional probabilities and the fact that most nets in a VLSI circuit have few connections (an average of about 4 over our suite of benchmark circuits), we obtain $p(n_i^{1\to 2}|u) \approx \prod_{u_x \in (n_{i,1}-\{u\})} p(u_x)$, $p(n_i^{2\to 1}|u^c) \approx \prod_{u_y \in n_{i,2}} p(u_y)$, and thus arrive at the following approximation of $g_{n_i}(u)$; see [5] for details.

$$g_{n_i}(u) \approx c(n_i)[\prod_{u_x \in (n_{i,1}-\{u\})} p(u_x) - \prod_{u_y \in n_{i,2}} p(u_y)] \qquad (3)$$

3.1..2 Net Not in Cutset

We now consider the gain contributed to u by a net n_i that is not currently in the cutset and is not locked in the subset, say, V_1, that it lies in. Then, it will be introduced into the cutset when u is moved from V_1 to V_2. Thus intuitively, $g_{n_i}(u)$ should be negative, and is given by

$g_{n_i}(u) = -c(n_i)$(Probability that n_i remains in the cutset after u is moved)

$= -c(n_i)$(Probability that not all nodes in $n_i \cap V_1 - \{u\}$ will be moved given u has been moved)

$= -c(n_i)(1 - p(n_i^{1\to 2}|u)) = -c(n_i)(1 - p(n_{i,1}|u))$

Again proceeding as in the net-in-cutset case, we obtain

$$g_{n_i}(u) = -c(n_i)(1 - \prod_{u_x \in n_{i,1}-\{u\}} p(u_x)) \qquad (4)$$

The total gain of node u is then $g(u) = \sum_{u \in n_i} g_{n_i}(u)$.

3.2. Calculating Node Probabilities

After the gains of every node has been computed once by either using a first approximation probability of p_{init} for each node, or by first computing their deterministic gains (Eqn. 1), their probabilities are computed using a suitable monotonically increasing function $p(u) = f(g(u))$ of their gains.

There are two caveats with probability calculation. One is that, since there are no certainties in node moves, it seems reasonable to establish a maximum probability $p_{max} < 1$ and a minimum probability $p_{min} > 0$ within which interval all node probabilities lie[3]. The second caveat is to establish upper and lower gain thresholds g_{up} and g_{lo}, such that all nodes with gains greater than or equal to g_{up} will get probability p_{max}, while those with gains lower than g_{lo} will get probability p_{min}. The rationale for establishing thresholds is that nodes with high gains, say, greater than 2, are going to be ultimately moved to the other subset no matter what, and those with very low gains, say less than -1, will most likely not be actually moved in the current pass. One obvious example of such functions is a linear probability function between $p(u)$ and $g(u)$ when $g_{lo} \leq g(u) \leq g_{up}$.

[3]Actually, it is not unreasonable to have $p_{max} = 1$, but p_{min} definitely needs to be greater than 0.

3.3. An Example

To illustrate the improvement offered by the probability-based node gain calculation over deterministic ones as in FM and LA, let us go back to the example of Fig. 1. In this example, we use the method of obtaining the initial deterministic gains of nodes (Eqn. 1) and their probabilities (using some monotonically increasing function f of the gains). Figure 1b shows the initial gains and probabilities $g(u), p(u)$ for each node. We assume for simplicity of exposition that for each net n_1 to n_{11} in the cutset, their $p(n_i^{2\to 1})$ terms are equal; thus the difference in the node gains $g(u)$s will only depend on their $p(n_i^{1\to 2}|u)$ terms (see Eqns. 3 and 4). In the second iteration, the node gains are calculated as follows using Eqn. 3 and 4.

$g_{n_1}(1) = g_{n_2}(1) = 1, g_{n_9}(1) = (0.2)^4 = 0.0016$, thus $g(1) = 2.0016$;

$g_{n_3}(2) = g_{n_4}(2) = 1, g_{n_{10}}(2) = (0.2)^2 = 0.04$, thus $g(2) = 2.04$;

$g_{n_6}(3) = g_{n_7}(3) = 1, g_{n_{11}} = (0.8)^2 = 0.64$, thus $g(3) = 2.64$.

Similarly, we obtain $g(10) = g(11) = 1.8$. To obtain the gains of nodes 4 to 9, we assume that nets n_{12} to n_{17} that are not in the cutset are each connected to one other node (not shown) of probability 0.5. We then get $g(8) = g(9) = -0.3$ and the gains of nodes 4 to 7 as -0.49. These gain values and their corresponding probabilities are shown in Fig. 1c. We now clearly see that node 3 has the highest gain and is thus the best node to move as we had concluded intuitively from Fig. 1! Note that the $p(u)$s of nodes 1, 2 and 3 are all 1 (e.g., when $g_{up} = 2$; see Sec. 3.2.); however, node selection is based on their gains.

3.4. Node Updates

When net n_i is locked in V_2, then as seen earlier in the derivation of Eqn. 3, $p(n_i^{1\to 2}) \approx \prod_{u_x \in n_{i,1}} p(u_x)$, since this probability is implicitly conditioned on the previous move(s) from V_1 to V_2 of the node(s) currently locked in $n_{i,2}$. Also in this case, $p(n_i^{2\to 1}) = 0$, since $p(u) = 0$ for a locked node u. Hence it follows that when n_i is locked in V_2, then for an unlocked node $u_x \in n_{i,1}, p(n_i^{1\to 2}|u_x) = p(n_i^{1\to 2})/p(u_x)$, and thus

$$g_{n_i}(u_x) = c(n_i) \cdot p(n_i^{1\to 2}|u_x) = c(n_i) \cdot p(n_i^{1\to 2})/p(u_x) \qquad (5)$$

Also, for u_y an unlocked node in $n_{i,2}$, where n_i is locked in V_2, we obtain using Eqn. 3 and the fact that $p(n_i^{2\to 1}) = 0$,

$$g_{n_i}(u_y) = -c(n_i) \cdot p(n_i^{1\to 2}|u_y^c) = -c(n_i) \cdot p(n_i^{1\to 2}) \qquad (6)$$

Similar expressions apply when n_i is locked in V_1. These equations are very useful for efficient node updating after a move, as discussed next.

After moving a node u, say, from V_1 to V_2, we first update, $p(n_i^{1\to 2})$ and $p(n_i^{2\to 1})$ of every net n_i that u is connected to. We follow this by updating the gains of all nodes connected to each such n_i (i.e., the neighbors of u) according to Eqn. 5 or 6. We then set $p(u) = 0$, to represent the fact that u is locked. Finally, we update the gains of a few, say, five, of the top ranked nodes in each subset using Eqn. 3 or 4; this is needed since some of these top nodes can be neighbors of the neighbors of u, whose probabilities have been updated. Note that potentially we could carry on the updating process for the neighbors of the neighbors of u, their nets, and so forth until all nodes and nets have been updated. However, the benefit of doing such a complete updating is minimal at best and it is very time consuming. Since, it is really the top few nodes that are in contention for the next move, the above update process is all that is really necessary.

3.5. Time and Space Complexities

Recall that n is the number of nodes, e the number of nets, p the average number of pins per node, i.e., the number of nets it is connected to, q the average number of pins on a net, i.e., the number of nodes a net is connected to, $d = p(q-1)$ is the average number of neighbors per node. We define $m = pn = qe$ as the total

Test Case	# Nodes	# Nets	# Pins	Test Case	# Nodes	# Nets	# Pins
balu	801	735	2697	19ks	2844	3282	10547
bm1	882	903	2910	biomed	6514	5742	21040
p1	833	902	2908	industry2	12637	13419	48404
p2	3014	3029	11219	t2	1663	1720	6134
s13207	8772	8651	20606	t3	1607	1618	5807
s15850	10470	10383	24712	t4	1515	1658	5975
s9234	5866	5844	14065	t5	2595	2750	10076
struct	1952	1920	5471	t6	1752	1541	6638

Table 1: Benchmark circuit characteristics.

Test Case	Cut Size				Improvement %		
	MELO	Paraboli	EIG1	PROP	MELO	Paraboli	EIG1
balu	28	41	110	27	3.6	34.1	75.5
bm1	48		75	50	-4.0		33.3
p1	64	53	75	47	26.6	11.3	37.3
p2	169	146	254	143	15.4	2.1	43.7
s13207	104	91	110	75	27.9	17.6	31.8
s15850	52	91	125	65	-20.0	28.6	48.0
s9234	79	74	166	41	48.1	44.6	75.3
struct	38	40	49	33	13.2	17.5	32.7
19ks	119		179	105	11.8		41.3
biomed	115	135	286	83	27.8	38.5	71.0
industry2	319	193	525	220	31.0	-12.3	58.1
t2	106		196	90	15.1		54.1
t3	60		85	59	1.7		30.6
t4	61		207	52	14.8		74.9
t5	102		167	79	22.5		52.7
t6	90		295	76	15.6		74.2
Total	1554	864	2904	1245	19.9	15.0	57.1

Table 3: Comparisons of cutset sizes on ACM/SIGDA benchmark circuits for the 45-55% balance criterion produced by PROP, MELO [2], PARABOLI [11] and EIG1 [7]; the results of PARABOLI and EIG1 are also given in [2].

number of pins in the circuit. Even if the average and maximum values of pins per node, pins per net and neighbors per node are very different, the time and space complexities determined below will hold, since these costs are amortized over all nodes and nets with varying number of pins. The time complexities of the different stages of PROP are as follows (details are in [5]. (1) We have the standard adjacency list for nodes and nets, and also store nodes, according to their gains, in a balanced binary AVL tree. This takes $\Theta(m)$ time. It also follows from this that the space complexity is $\Theta(m)$. (2) Computing $p(n_i^{2\to 1})$ and $p(n_i^{1\to 2})$ over all nets n_i as specified in Sec. 3.4. takes $\Theta(m)$ time, as does the computation of $g_{n_i}(u)$ over all nodes and nets. Computing node probabilities takes $\Theta(n)$ time. (3) It takes $\Theta(\log n)$ time to find the best node to move using an AVL data structure; thus total time is $\Theta(n \log n)$. (4) The number of entities (nodes and nets) updated are p nets of the moved node u, d neighbors of u, $\Theta(p)$ nets of the constant number of top nodes that are also updated. We see from Sec. 3.4. that each update step for nets and neighbors connected to u takes constant time. In the AVL tree data structure, it takes $\Theta(\log n)$ time to delete and reinsert a node; thus it takes $\Theta(d \log n)$ time to reinsert all updated nodes per move. Hence the updating process takes a total of $\Theta(nd \log n)$ time over one pass.

From the above discussion, the time complexity of PROP for an entire pass is $\Theta(nd \log n) = \Theta(mq \log n)$. For VLSI circuits, q is a small constant like 4 and thus the time complexity is $\Theta(m \log n)$. Finally, as mentioned above PROP's space complexity is $\Theta(m)$.

4. Experimental Results

Tables 2 and 3 give cutset results for the 50-50% and 45-55% balance criterion, respectively, of many ACM/SIGDA benchmark circuits whose number of nodes, nets and pins are given in Table 1. These are the same circuits used in the MELO technique [2] to compare their results to those of PARABOLI and EIG1. For both the 50-50% and 45-55% balance criterion, PROP uses the following parameters: single moves, AVL tree data structure, p_{init} = 0.95, p_{max} = 0.95, p_{min} = 0.4, the linear probability function, g_{up} = 1, and $g_{lo} = -1$. Table 2 compares cutset sizes for the 50-50% balance criterion produced by FM20, FM40, FM100, (these are FM run on 20, 40 and 100 initial random partitions, respectively), WINDOW [1] (which uses FM20 as a final phase) and PROP using 20 runs. We see that PROP cutsets are on the average 30% better than FM20 and 22% better than FM100. These percentage improvements are calculated as (cutset improvement/larger cut set)$\times$100. Note also that as we increase the number of runs for FM, we start getting diminishing returns; thus we probably cannot do much better if we increase the number of FM runs beyond 100. PROP is also 27% better than LA-2 (20 runs), 16.2% better than LA-2 (40 runs), and 17% better than LA-3 (20 runs). It also performs 26% better than WINDOW. In Table 3, we compare the performance of 20 runs of PROP to some recent state-of-the-art clustering-based methods, viz., EIG1 [7], PARABOLI [11] and MELO [2] for the 45-55% balance criterion. PROP performs better than all of these methods, by 57% over EIG1, by 20% over MELO and by 15% over PARABOLI. Thus used as a stand-alone partitioner, PROP gives better results than the best recent clustering-based partitioners.

FM has a complexity of $\Theta(nd)$ if all net costs are assumed to be 1, as is the case in our implementation, and it is thus very fast. PROP is about 4.6 times slower than FM per run. If the assumption of unit net cost cannot be made, as in the case when circuits are partitioned to minimize timing [8], then FM, like PROP, will need a binary tree data structure, and its time complexity will be $\Theta(nd \log n)$. However, PROP will have the same complexity ($\Theta(nd \log n)$) under non-uniform net costs. In order to demonstrate how PROP compares to both versions of FM, i.e., with bucket (FM-bucket) and tree (FM-tree) data structures, we have implemented both of these and tabulated times per run for different SIGDA benchmark circuits in Table 4. This table also gives the run times of these circuits for all the other algorithms compared here. Note that the actual times for FM100, LA-2, LA-3 and PROP (these are the 45-55% case run times for PROP; those for the 50-50% case are a little lower) are obtained by multiplying the per run times by 100, 40, 20 and 20, respectively. This is what we have done in the last row of Table 4 in which the total times over all the circuits are given. Note that it is more appropriate to compare PROP with 20 runs to FM100 and LA-2 with 40 runs instead of to FM20 and LA-2 with 20 runs, respectively, since, PROP's cutset improvement margins are lesser compared to the former two methods than to the latter two. From Table 4, it is easy to see that PROP is among the fastest partitioners. It is very comparable to FM100-bucket and LA-2 (though it obtains 22.3% and 16.2% better cutsets, respectively, than these methods), and slightly slower (by 37%) than EIG1—but then it obtains 57% better cutsets than EIG1. Assuming that the Sun Sparc 5, Sparc 10 and the DEC 3000 are comparable in speed , PROP is 3.15 times faster than FM100-tree, 3.9 times faster then PARABOLI, about 2.2 times faster than LA-3 and MELO, and at least 1.5 times faster than WINDOW.

5. Conclusions

We have presented a probabilistic-gain based approach PROP to iterative-improvement type min-cut partitioning. The methodology is based on probability and conditional probability theory. Results run on a suite of ACM/SIGDA benchmarks show that we outperform other iterative-improvement methods like FM and LA by wide margins, and that we also outperform recent clustering-based partitioners by significant margins. The run times of PROP also compare very favorably with those of the iterative-improvement and clustering techniques; it is comparable to those of FM and LA, and much faster than those of most clustering-based methods. It thus seems that PROP can do very well as a stand-alone partitioner, and we believe that in conjunction with a clustering initial phase it will yield a high-quality partitioning tool. The probabilistic-gain approach opens up a number of exciting possibilities, for example, k-way partitioning, multiple-FPGA partitioning, and partitioning

Test Case	**Cut Size**							**Improvement %**					
	FM100	FM40	FM20	LA-2	LA-3	WINDOW	PROP	FM100	FM40	FM20	LA-2	LA-3	WINDOW
balu	32	32	32	31	31		32	0.0	0.0	0.0	-3.1	-3.1	
bm1	55	57	65	58	55	70	54	1.8	5.3	16.9	6.9	1.8	22.9
p1	57	57	59	59	55	60	59	-3.4	-3.4	0.0	0.0	-6.8	1.7
p2	236	238	245	215	183	258	154	34.7	35.3	37.1	28.4	15.8	40.3
s13207	92	101	101	81	89		83	9.8	17.8	17.8	-2.4	6.7	
s15850	112	120	120	122	75		73	34.8	39.2	39.2	40.2	2.7	
s9234	53	59	59	57	58		55	-3.6	6.8	6.8	3.5	5.2	
struct	45	47	52	45	45		38	15.6	19.1	26.9	15.6	15.6	
19ks	142	150	150	141	153	136	120	15.5	20.0	20.0	14.9	21.6	11.8
biomed	83	83	83	122	91	164	88	-5.7	-5.7	-5.7	27.9	3.3	46.3
industry2	428	501	501	492	378	392	254	40.7	49.3	49.3	48.4	32.8	35.2
t2	115	115	115	124	105	105	91	20.9	20.9	20.9	26.6	13.3	13.3
t3	72	72	72	78	90	67	58	19.4	19.4	19.4	25.6	35.6	13.4
t4	86	88	97	94	88	61	58	32.6	34.1	40.2	38.3	34.1	4.9
t5	97	97	149	109	96	101	82	15.5	15.5	45.0	24.8	14.6	18.8
t6	71	71	71	70	63	70	81	-12.3	-12.3	-12.3	-13.6	-22.2	-13.6
Total Cuts	1776	1888	1971	1898	1655	1484	1380	22.3	26.9	30.0	27.3	16.6	25.9

Table 2: Comparisons of cutset sizes on ACM/SIGDA benchmark circuits for the 50-50% balance criterion produced by three versions of FM (with 20, 40 and 100 runs), LA-2 and LA-3 (each with 20 runs), PROP (with 20 runs) and WINDOW [1] in which clustering is followed by 20 runs of FM. On executing LA-2 with 40 runs (this makes the times for LA-2 and PROP almost equal—see Table 4) instead of 20, a total cutset of 1647 is obtained; this represents a cutset improvement of 16.2% for PROP.

Test Case	**Sun Sparc-5**					**DEC 3000 Model 500 AXP**		**Sun Sparc-10**	
	FM-bucket x100	FM-tree x100	LA-2 x40	LA-3 x20	PROP x20	EIG1	Paraboli	MELO	WINDOW
balu	0.21	0.42	0.42	0.94	0.78	6.2	15.5	7	
bm1	0.24	0.58	0.56	1.12	1.02			4	
p1	0.24	0.57	0.51	1.10	0.96	3.1	18.3	8	
p2	1.28	3.53	2.87	4.95	6.94	17.6	137.4	89	> 115.7
s13207	3.39	10.23	7.14	10.36	8.87	43.5	2060.4	710	
s15850	3.39	10.60	8.55	13.09	14.55	78.4	2730.9	1197	
s9234	2.22	6.64	4.71	7.04	6.93	24.2	490.3	516	
struct	0.46	1.15	1.55	2.14	2.08	6.9	35.2	38	
19ks	1.15	3.12	2.66	5.27	4.33			79	
biomed	2.96	7.24	5.55	8.74	12.49	521.2	710.9	496	> 421
industry2	7.27	24.18	17.56	37.05	43.34	706.6	1367.3	1855	> 1385
t2	0.46	1.18	1.45	40.99	3.20			29	
t3	0.56	1.33	1.14	23.32	2.56			27	
t4	0.41	1.02	1.22	39.57	2.46			24	
t5	0.81	2.13	2.27	69.84	4.86			67	
t6	0.50	1.09	0.87	1.33	3.77			31	
Total	2555 (all ckts., $\times 100$)	7501 (all, $\times 100$)	2361.2 (all, $\times 40$)	5331 (all, $\times 20$)	2383/1939/1255 (all/9/3, $\times 20$)	1408 (9 ckts.)	7567 (9 ckts.)	5177 (all)	> 1922 (3 ckts.)

Table 4: Comparisons of CPU times in secs per run, and total times over all circuits and all runs made for various algorithms.

for timing minimization, that we will explore in the future.

References

[1] C.J. Alpert and A.B. Kahng, "A general framework for vertex orderings, with applications to circuit clusterings", *Proc. IEEE/ACM International Conference on CAD*, Nov. 1994, pp. 63-67.

[2] C.J. Alpert and S-Z Yao, "Spectral Partitioning: The more eigenvectors the better", *Proc. Design Automation Conf.*, 1995, pp. 195-200.

[3] D. G. Schweikert and B. W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits", *Proc. 9th Design automation workshop*, 1972, pp. 57-62.

[4] S. Dutt, "New faster Kernighan-Lin-type graph-partitioning algorithms", *Proc. IEEE/ACM International Conference on CAD*, Nov. 1993.

[5] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", Tech. Report, EE Dept., Univ. of Minnesota, 1996—available at ftp site ftp-mount.ee.umn.edu in file /pub/faculty/dutt/vlsi-cad/papers/dac96-ext.ps.

[6] C.M. Fidducia and R.M. Mattheyses, "A linear-time heuristic for improving network partitions", *Proc. Nineteenth Design Automation Conf.*, 1982, pp. 175-181.

[7] L. Hagen and A. Kahng, "Fast spectral methods for ratio cut partitioning and clustering", *Proc. Int'l. Conf. Computer-Aided Design*, 1991, pp. 10-13.

[8] M.A.B. Jackson, A. Srinivasan and E.S. Kuh, "A fast algorithm for performance driven placement", *Proc. IEEE/ACM International Conference on CAD*, 1990, pp. 328-331.

[9] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", *Bell System Tech. Journal*, vol. 49, Feb. 1970, pp. 291-307.

[10] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks", *IEEE Trans. on Comput.*, vol. C-33, May 1984, pp. 438-446.

[11] B.M. Riess, K. Doll and F.M. Johannes, "Partitioning very large circuits using analytical placement techniques", *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 646-651.

[12] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer, B.V., Deventer, The Netherlands.

[13] Y.C. Wei and C.K. Cheng, "Towards efficient hierarchical designs by ratio cut partitioning", *Proc. Int'l. Conf. Computer-Aided Design*, 1989, pp. 298-301.

[14] Y.C. Wei and C.K. Cheng, "A two-level two-way partitioning algorithm", *Proc. Int'l. Conf. Computer-Aided Design*, 1990, pp. 516-519.