Published (with minor modifications) in IEEE Trans. CAD, Vol. 19, No. 5, May 2000, pp. 534-549

Probability-Based Approaches to VLSI Circuit Partitioning*

Shantanu Dutt and Wenyong Deng

Department of EECS
University of Illinois at Chicago
Chicago, IL 60607-7053
dutt@eecs.uic.edu

Cadence Design Systems
San Jose, California 95134
wydeng@cadence.com

Abstract

Iterative-improvement 2-way min-cut partitioning is an important phase in most circuit placement tools, and finds use in many other CAD applications. Most iterative improvement techniques for circuit netlists like the Fiduccia-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 information. Krishnamurthy suggested a lookahead (LA) gain calculation method to ameliorate this situation; however, as we show, it leaves room for improvement. We present here a probabilistic gain computation approach called PROP (PRObabilistic Partitioner) that is capable of capturing the future implications of moving a node at the current time. We also propose an extended algorithm SHRINK-PROP that increases the probability of removing recently "perturbed" nets (nets whose nodes have been moved for the first time) from the cutset. This is necessary, since in a regular move process, the removal probabilities of most nets either remain unchanged or even decrease when their nodes are moved for the first time. Experimental results on medium- to large-size ACM/SIGDA benchmark circuits show that PROP and SHRINK-PROP outperform previous iterative-improvement methods like FM (by about 30% and 37%, respectively) and LA (by about 27% and 34%, respectively). Both PROP and SHRINK-PROP also obtain much better cutsizes than many recent state-of-the-art partitioners like EIG1, WINDOW, MELO, PARABOLI, GFM and GMetis (by 4.5% to 67%). We also show that the space and time complexities of PROP and SHRINK-PROP are very reasonable. Our empirical timing results reveal that PROP is appreciably faster than all recent techniques except GMetis—all other partitioners including ours work on flat netlists, while GMetis uses multilevel clustering, which is a paradigm orthogonal to basic partitioning, and can be used in conjunction with any partitioner. Further, PROP is only a little slower than FM and LA, both of which are very fast (but give sub-optimal results). SHRINK-PROP is about two times slower than PROP, but still faster than most recent partitioners. We also obtain results on the more recent ISPD-98 benchmark suite that show similar substantial mincut improvements by PROP and SHRINK-PROP over FM (24% and 31%, respectively). It is also noteworthy that SHRINK-PROP's results are within 2.5% of those obtained by hMetis, one of the best multilevel partitioners. However, the multilevel paradigm, as mentioned above, is orthogonal to SHRINK-PROP. Further, since it is a "flat" partitioner, it has advantages over hMetis in partition-driven placement applications.

Keywords: *clustering effect, iterative-improvement, min-cut partitioning, probabilistic gain, VLSI circuit*

*This work was supported partly by NSF grant MIP-9210049 and a grant from Intel Corp. This paper is a greatly extended version of [14].

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. 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. If we are interested in minimizing chip layout area, then at this point we know the approximate placement of nodes on it so that wiring and thus layout area is minimized. Other goals like timing minimization can also be achieved in a similar manner by assigning appropriate weights to different nets, and by changing the optimization metric for the partitioner (e.g. to min-max).

Formally, the k-way min-cut partitioning problem can be stated as follows. 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; generally the output of a node is connected to the inputs of several other nodes by a net. 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 q_n, the average number of nodes that a net connects by q_e, and the average number of neighbors of a node by $d = q_n(q_e - 1)$—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. 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

$$cost(P^k) = \sum_{i=1}^{k} c(n_t), \text{where } n_t \in E_{cut}$$

Here $c(n_t)$ 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 net n_t. If power minimization is the goal then $c(n_t) \propto$ the switching probability of the node driving n_t [10]. For timing minimization, a net is assigned weight proportional to the longest delay path through it to ensure that the length of critical or near-critical nets are kept as short as possible [12, 27, 21]; in this case, the optimization metrics proposed for partitioning are (weighted) mincut [27, 21] and weighted min-max [12].

Recursive 2-way partitioning is an efficient and popular approach to obtaining k-way partitions for $k > 2$ [9, 20, 30, 36, 35]. We will thus be concerned here with the 2-way min-cut partitioning problem. Since 2-way min-cut partitioning is NP-complete [19], a number of approximate schemes have been proposed. These include iterative improvement methods [16, 17, 23, 24, 29, 30], simulated annealing [31, 32] and clustering-based techniques [7, 5, 18, 20, 28, 29, 36, 35]. An excellent survey on partitioning techniques appears in [6]. In iterative improvement, 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 reducing the cutsize. Iterative improvement methods are also sometimes used as a preprocessing phase for clustering as in [7, 35, 36]. Thus mincut partitioning using iterative improvement is a fundamental tool in obtaining good VLSI cell placement.

A number of iterative min-cut methods for graph or hypergraph partitioning have been previously proposed [16, 17, 23, 24, 29, 30]. Kernighan and Lin proposed the well-known KL graph partitioning algorithm using pair swaps to improve a random initial 2-way partition [23]. Schweikert and Kernighan extended this algorithm to one (SK) that can handle netlists [30]. Fiduccia and Mattheyses gave a similar algorithm (FM) 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 [17]. 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 minimize power dissipation on interconnects or to minimize circuit delay [27, 21], then the partitioning process is much slower, since the bucket data structure of [17] can no longer be used for storing nodes ordered by their gains. Dutt improved the time complexity of the KL algorithm by discovering a result that established that only a particular subset of $O(d^2)$ node pairs need to be searched in order to determine the best node-pair to swap, where d is the maximum number of neighbors of any node [16]. This *neighborhood search* strategy was incorporated in the Quick_Cut (QC) algorithm, which has a worst-case time complexity of $\Theta(\max(ed, e \log n))$, and an average-case complexity of $\Theta(e \log n)$; the KL algorithm has a time complexity of $\Theta(n^2 \log n)$.

The node-gain calculations in all the above algorithms (KL, SK, FM and QC) use only local netlist information, and this quite often gives inaccurate indications of the potential improvement that can be obtained

by moving a node (or swapping a node pair), especially so in the case of hypergraphs. In [24], 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. Some pre-clustering techniques have been proposed in [34, 29] to remedy the weakness of FM and KL in getting trapped in local minima for large circuits; the techniques in [29] apply directly to graphs, though they may be extendible to hypergraphs. Very good results have been obtained at the cost of considerable CPU time increases and implementation complexities.

In this paper, we present a precise probabilistic gain calculation method PROP (for PRObabilistic Partitioner), and an enhanced algorithm SHRINK-PROP that capture 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 and ISPD-98 benchmark suites, which show that PROP performs an average of 30% better than FM and 27% better than LA, while SHRINK-PROP obtains about 37% and 34% better results than FM and LA, respectively. We also compare our methods to some of the more recent techniques like EIG1 [20], WINDOW [7], PARABOLI [28], MELO [5] and GMetis [4]. Results show that our new techniques also performs significantly better (by 4.5% to 67%) than these techniques.

The rest of this paper is organized as follows. In Sec. 2 we discuss two previous relevant iterative improvement methods FM and LA. Section 3 discusses the probability gain formulation of PROP and derives its complexities. Sec. 4, we present an enhanced algorithm SHRINK-PROP that attempts to obtain lower cutsizes by increasing the removal probability of recently "perturbed" nets. Section 6 presents the cutsizes obtained by PROP and SHRINK-PROP on medium- to large-size ACM/SIGDA and ISPD-98 benchmark circuits, and compares them to the results of various classical and state-of-the-art techniques such as FM, LA, WINDOW, EIG1, MELO, PARABOLI, GFM, GMetis and hMetis. Conclusions are in Sec. 7.

2 Previous Iterative-Improvement Methods

Here we briefly describe the netlist partitioning process and node-gain calculations used in the FM and LA algorithms. Assume that there are n nodes in the hypergraph G, and that it is being partitioned into $\{V_1, V_2\}$. The FM gain computation is as follows [17]. 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

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

This gain definition of a node is the 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. After all nodes are moved and locked in this manner, all prefix sums $S_k = \sum_{t=1}^{k} gain(u_t)$ are computed, and p is chosen so that the partial sum S_p is the 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 gain G_{max} obtained is 0. This solution represents a local minimum. Empirical evidence on what are currently considered small graphs (hundreds to thousands of nodes) shows that the number of passes required to achieve a local minima is two to four [23, 17]; for larger graphs (tens of thousands to more than a hundred thousand nodes) this number is about 8 for the FM algorithm and 10 to 12 for more sophisticated algorithms like PROP.

As mentioned above, $gain(u)$ 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. However, it may be beneficial to give a node with higher potential gain but lower immediate gain priority over one with much lower potential but a higher immediate gain. Krishnamurthy developed a scheme that estimates the potential gain by using a "lookahead" gain vector for each node [24]. Consider a gain vector $gain(u)[k]$ 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 that belong to the cutset and to which i nodes in V_1 (including u) are connected minus the number of nets in the cutset connected to u that have $i-1$ nodes of V_2 connected to them. A gain vector $\mathbf{a}$ is said to be greater than a gain vector $\mathbf{b}$ if either $\mathbf{a}[1] > \mathbf{b}[1]$ or if there exits an $i < k$ such that $\mathbf{a}[j] = \mathbf{b}[j]$ for all $1 \leq j \leq i$ and $\mathbf{a}[i+1] > \mathbf{b}[i+1]$. 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 with even a small number of nodes with medium to large connectivities, it can become practically infeasible or very slow to use a lookahead of even 3. This is demonstrated by our experimental results (see Table 5 in Sec. 6) which show that the CPU time more than doubles when increasing the lookahead from 2 to 3, and also that this time is more than that of our more informed algorithms PROP and SHRINK-PROP.

Figure 1 illustrates the differences between FM, LA and the probability-based method PROP. Here, we discuss the gain function differences between FM and LA; the differences to PROP are described later. For simplicity, only nodes in V_1 are considered, and all nets are assumed to have unit cost. 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 nodes 1, 2 and 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

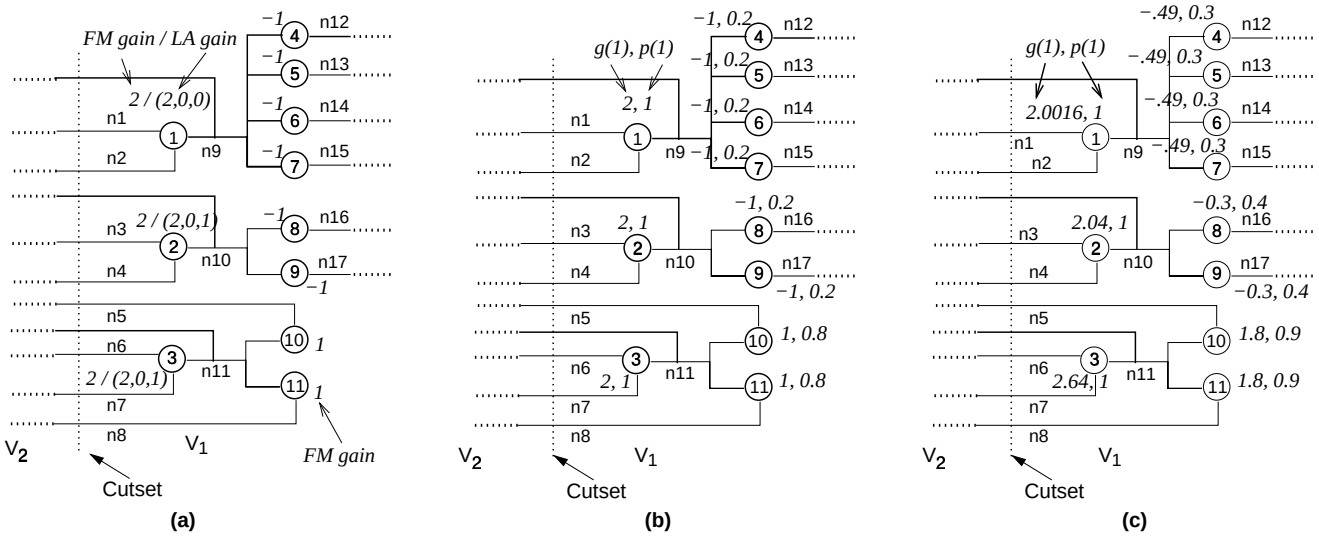


Figure 1: Illustration of the improvement of PROP over the LA and FM algorithms: (a) FM and LA-3 gains (the latter is only shown for nodes 1, 2 and 3). (b & c) Probabilistic gains and node-move probabilities after (b) the first, and (c) the second iteration of node gain and probability calculation.

cutset (i.e., nodes 2 and 3 have a better potential gain than node 1). The LA algorithm is able to do better then FM in this regard. Assuming a lookahead of 3 (LA-3), the node gain vectors for nodes 1, 2 and 3 are as follows: $gain(1) = (2, 0, 0)$, $gain(2) = (2, 0, 1)$ and $gain(3) = (2, 0, 1)$; 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 should also convince the reader that node 3 is a better candidate to move than node 2. The reason is that both nodes 10 and 11 (to which node 3 is connected 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 FM nor LA 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 that we have devised is able to see the likelihood of future events much better, and is described next. We will discuss the above example in the context of this method in Sec. 3.4.

3 The Probabilistic-Gain 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 event $M(u)$ that u will be *actually moved* to the other

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. (5) and (7), and $p(u)$ calculation $iter$ times
5. **Repeat** Steps 6-8 **until** all nodes are locked, or no more moves can be made to meet the balance criterion.
6. Select the node u with the best gain in either subset to be move 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 unlocked neighbors as described in Sec. 3.5.
9. Compute 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} corresponds to the pth move **then** actually make only the first p moves
    end
    else exit

End.

Figure 2: The probabilistic-gain based partitioning algorithm PROP

subset in the current pass of the partioning process[1]. Note that $M(u)$ is the event in which u belongs to the subset of actually moved nodes; it is not the event in which u is the only moved node. From $p(u)$, 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. An important question is how to obtain the node probabilities $p(u)$s in the first place. The obvious answer to this is that they should be computed from their respective (probabilistic) node gains—higher the gain, higher should be 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}. 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. 3.3). 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 can continue for a few iterations, though the best results are obtained for only one iteration; see Sec. 5.1 for an explanation of this phenomenon. After this

[1]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.

initial process is completed, we move nodes with the best gains between the two subsets just like in other iterative improvement methods. After each move, we update the node gains and probabilities as explained in Sec. 3.5. 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 into 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 some future moves will have large immediate gains. 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 immediate-gain based iterative improvement methods like FM and LA. The partitioning algorithm is described formally in Fig. 2.

3.1 The Sample Space of Node Moves

Before describing how node gains are computed from node probabilities, we establish that given any set of node probabilities, the sample space of events representing subsets of nodes that are actually moved in a pass is a valid probability space. All discussions in this and later sections relate to a single pass. Further, whenever we refer to the move of a node (and the probability of a move), we mean the actual move of the node, not its virtual move (all nodes are virtually moved in an iterative-improvement type approach.).

We first describe some concepts of conditional probabilities [33]. Let A and B be two events with probabilities $p(A)$ and $p(B)$, respectively, of occurring. The event AB is one in which both events A and B occur, and its probability is denoted by $p(AB)$. $p(A \mid B)$ is the *conditional probability* of event A occurring, given that event B has already occurred. If the occurrence of B raises (lowers) the likelihood of A occurring, then $p(A \mid B) > p(A)$ ($p(A \mid B) < p(A)$), and if the occurrence of B does not affect the likelihood of A's occurrence, then A is said to be *independent* of B, and $p(A \mid B) = p(A)$. In general, the probability $p(AB)$ of both events A and B occurring is given by $p(AB) = p(A \mid B) \cdot p(B)$; when they are independent, this is just $p(A) \cdot p(B)$.

Let the set of nodes be $V = V_1 \cup V_2 = \{u_1, \ldots, u_n\}$, and the probability that node u_i moves be $p(u_i), 0 \leq p(u_i) \leq 1$; thus $1 - p(u_i)$ is the probability that u_i is not moved in the current pass. Thus with any event, representing the subset of nodes actually moved in a pass, we can assign two possible outcomes for each node: either it is actually moved to the other side of the partition or it remains in its current side.

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* $E \cap F = \emptyset$.
(3) $P[\Omega] = 1$, *where* Ω *is the set of all possible events.*

Proof: See [11]. □

3.2 Computing Node Gains from Node Probabilities

We now describe how probabilistic node gains are computed from node probabilities. For each node u, we compute its gain components $g_{n_i}(u)$'s corresponding to its connected nets n_i. The total gain $g(u)$ of u is then obtained as $g(u) = \sum_{u \in n_i} g_{n_i}(u)$. 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. In the next two subsections, we determine formulations for $g_{n_i}(u)$ for the two cases: when n_i is in the cutset, and when it is not.

3.2.1 Nets in the Cutset

For a subset S of nodes of one of the subsets, say, V_1, let $M(S)$ denote the event that all nodes in S are actually moved to the other subset, in this case, V_2, in the current pass; the probability of this event is denoted by $p(S)$. Note that $M(S)$ is not the event in which **only** the nodes of S are moved, but one which includes the moves of nodes of S. Similarly, we define $N(S)$ to be the event in which no node in S is moved to the other subset V_2, and its probability is denoted by $p(S^c)$. Whenever the meaning is clear, we will abbreviate $M(S)$ and $N(S)$ by S and S^c, respectively (or $[S]$ and $[S^c]$ for clarity, when they are being intersected or AND'ed with other events, e.g., $[S_1][S_2]$). Furthermore, singleton sets $\{u\}$ and $\{u\}^c$ are abbreviated by u and u^c, respectively.

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 $p(n_i^{1 \to 2})$ as

$$p(n_i^{1 \to 2}) = (\text{Probability of } n_i \text{ being removed from the cutset by moving all nodes in } n_{i,1} \text{ to } V_2)$$

and $p(n_i^{2 \to 1})$ as

$$p(n_i^{2 \to 1}) = (\text{Probability of } n_i \text{ being removed from the cutset by moving all nodes in } n_{i,2} \text{ to } V_1)$$

Note that $p(n_i^{1 \to 2})$ [$p(n_i^{2 \to 1})$] implicitly takes into account the probability that no node in $n_{i,2}$ ($n_{i,1}$) is moved to V_1 (V_2)—if this is not the case, n_i cannot be removed from the cutset. Since $n_i^{1 \to 2}$ and $n_i^{2 \to 1}$ are mutually exclusive events, the probability $p(n_i)$ of removing n_i from the cutset is

$$p(n_i) = p(n_i^{1 \to 2}) + p(n_i^{2 \to 1})$$

There are only two possibilities associated with a node u—it either is actually moved (before the max-prefix point) or not—in the final partitioning solution. We need to compute the potential effect in terms of net removal probabilities, of these two possibilities on each net n_i connected to u. The term $p(n_i^{1\rightarrow 2} \mid u)$ is this probability for n_i when u is moved, while $p(n_i^{2\rightarrow 1} \mid u^c)$ is this probability when u is not moved. Instead of considering these two terms separately for the two possibilities for u, we can subtract the second from the first to indicate whether it is better for u to move or not—a positive result indicates it is, while a non-positive result indicates it is not. It is this result that we compute as $g_{n_i}(u)$, which is u's "gain for moving" with respect to net n_i. Thus

$$g_{n_i}(u) = c(n_i)[p(n_i^{1\rightarrow 2} \mid u) - p(n_i^{2\rightarrow 1} \mid u^c)] \tag{2}$$

In [11], a detailed rationale is given for the following practical approximations of the terms $p(n_i^{1\rightarrow 2} \mid u)$ and $p(n_i^{2\rightarrow 1} \mid u^c)$:

$$p(n_i^{1\rightarrow 2} \mid u) \approx p(n_{i,1} \mid u) \approx \prod_{u_x \in (n_{i,1} - \{u\})} p(u_x) \tag{3}$$

$$p(n_i^{2\rightarrow 1} \mid u^c) \approx \prod_{u_y \in n_{i,2}} p(u_y) \tag{4}$$

From Eqns. 2, 3 and 4, we obtain

$$g_{n_i}(u) \approx c(n_i)[\prod_{u_x \in (n_{i,1} - \{u\})} p(u_x) - \prod_{u_y \in n_{i,2}} p(u_y)] \tag{5}$$

3.2.2 Internal Nets

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. No additional formulation is required for such nets—Eqn. 2, reiterated below, also provides the gain contributions of such nets as follows.

$$g_{n_i}(u) = c(n_i)[p(n_i^{1\rightarrow 2} \mid u) - p(n_i^{2\rightarrow 1} \mid u^c)]$$

The $p(n_i^{2\rightarrow 1} \mid u^c)$ component in Eqn. 2 for such nets is 1, since it is already in V_1. We thus obtain

$$g_{n_i}(u) = -c(n_i)[1 - p(n_i^{1\rightarrow 2} \mid u) \approx -c(n_i)(1 - p(n_{i,1} \mid u)$$

It is also interesting to try to derive $g_{n_i}(u)$ for internal nets from "first principles". Net n_i will be introduced into the cutset when u is moved from V_1 to V_2. Thus, $g_{n_i}(u)$ should be negative, but less negative than $-c(n_i)$, since even though n_i will be introduced into the cutset, there is a likelihood that it will subsequently be removed from the cutset by future moves. Thus $g_{n_i}(u)$ should be

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

$$= -c(n_i)(\text{Probability that not all nodes in } n_{i,1} - \{u\} \text{ will be moved} \mid M(u))$$

$$= -c(n_i)(1 - p(n_i^{1\rightarrow 2} \mid u)) \approx -c(n_i)(1 - p(n_{i,1} \mid u)) \quad (6)$$

We thus obtain the same expression from first principles as by applying the general formulation of Eqn. 2 for this case.

Again proceeding as in the derivation of $p(n_{i,1} \mid u)$ in Eqn. 3 (see [11]), we obtain

$$g_{n_i}(u) \approx -c(n_i)(1 - \prod_{u_x \in n_{i,1} - \{u\}} p(u_x)) \quad (7)$$

The total gain $g(u)$ of node u is then computed as

$$g(u) = \sum_{u \in n_i} g_{n_i}(u) \quad (8)$$

3.3 Computing Node Probabilities

After the gains of every node has been computed 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 $f(g(u))$ of their gains. This function takes the maximum gain g_{max} and minimum gain g_{min} into account. Functions that we have experimented with are *linear* and *semi-Gaussian,* and are specified below.

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 $p_{max} \leq 1$ and a minimum probability $p_{min} > 0$ within which interval all node probabilities lie[2]. 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, will be moved to the other subset with very high likelihood, and those will very low gains, say less than -1, will most likely not be actually moved in the current pass. Thus g_{up} and g_{lo} should replace g_{max} and g_{min}, respectively, in the above probability functions. These functions are completely specified below, where we assume that $p_{min} > 0$.

Linear probability function:

$$f(g(u)) = \begin{cases} p_{max} & \text{when } g(u) > g_{up} \\ p_{min} + (p_{max} - p_{min})(g(u) - g_{lo})/(g_{up} - g_{lo}) & \text{when } g_{lo} \leq g(u) \leq g_{up} \\ p_{min} & \text{when } g(u) < g_{lo} \end{cases}$$

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

Semi-Gaussian probability function:

$$f(g(u)) = \begin{cases} p_{max} & \text{when } g(u) > g_{up} \\ p_{min} + (p_{max} - p_{min}) e^{(-\frac{1}{2}(g_{up} - g(u))^2 / \frac{1}{4}(g_{up} - g_{lo})^2)} & \text{when } g(u) \leq g_{up} \end{cases}$$

3.4 An Example

To illustrate the improvement offered by the probability-based node gain calculation over deterministic ones like 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 1(b) shows the initial gains and probabilities $g(u), p(u)$ for each node; note that the node probabilities are consistent with a monotonically increasing probability function. We assume for simplicity of exposition that for each net n_1 to n_{11} in the cutset, their $p(n_i^{2\rightarrow 1})$ terms are equal; thus the difference in the node gains $g(u)$s will only depend on their $p(n_i^{1\rightarrow 2} \mid u)$ terms (see Eqns. 5 and 7). In the second iteration, the node gains are calculated as follows using Eqn. 5 and 7.
$g_{n_1}(1) = g_{n_2}(1) = 1, g_{n_9}(1) = (0.2)^4 = 0.0016$, 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. 1(c). 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. 3.3); however, node selection is based on their gains.

3.5 Node Updates

When net n_i is locked in V_2, then from Eqn. 3

$$p(n_i^{1\rightarrow 2}) \approx \prod_{u_x \in n_{i,1}} p(u_x) \tag{9}$$

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, $p(n_i^{2\rightarrow 1}) = 0$, since $p(u) = 0$ for a locked node u. Equation 9 does not apply, however, when n_i is not locked in V_2, i.e, when this probability is not conditioned on some node move to V_2. From Eqns. 3 and 9, it follows that when n_i is locked in V_2, then for an unlocked node $u_x \in n_{i,1}$,

$$p(n_i^{1\rightarrow 2} \mid u_x) = p(n_i^{1\rightarrow 2})/p(u_x) \tag{10}$$

From Eqns. 5 and 10, and the fact that $p(n_i^{2\to1}) = 0$, $g_{n_i}(u_x)$ for an unlocked node $u_x \in n_{i,1}$, where n_i is locked in V_2, is simply given by

$$g_{n_i}(u_x) = c(n_i) \cdot p(n_i^{1\to2} \mid u_x) = c(n_i) \cdot p(n_i^{1\to2})/p(u_x) \tag{11}$$

Also, for u_y an unlocked node in $n_{i,2}$, where n_i is locked in V_2, we obtain using Eqn. 5 and the fact that $p(n_i^{2\to1}) = 0$,

$$g_{n_i}(u_y) = -c(n_i) \cdot p(n_i^{1\to2} \mid u_y^c) = -c(n_i) \cdot p(n_i^{1\to2})) \tag{12}$$

Similar expressions hold when n_i is locked in V_1, with nodes $u_y \in n_{i,2}$ on the unlocked side and nodes $u_x \in n_{i,1}$ on the locked side interchanging their roles. These equations are very useful for efficient node updation after a move, as discussed next.

After moving a node u, say, from V_1 to V_2, we first update, $p(n_i^{1\to2})$ and $p(n_i^{2\to1})$ 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. 11 or 12. We then set $p(u) = 0$, to represent the fact that u is locked. In the following update steps, we assume that the most recent move (of node u) was from V_1 to V_2; a move from V_2 to V_1 is handled in a symmetric fashion. For each net n_i connected to u, the updating of the gains of unlocked neighbors is performed in steps 1 and 2.

1. Update $p(n_i^{1\to2})$ and $p(n_i^{1\to2})$ for all $n_i \ni u$ as

   $$p(n_i^{1\to2}) = p(n_i^{1\to2})/p(u), \quad p(n_i^{2\to1}) = 0$$

   As discussed above, when n_i is not locked in V_2, then the expression of Eqn. 9 does not hold for $p(n_i^{1\to2})$. However, since we do not really use the value of $p(n_i^{1\to2})$ unless it is conditioned on a node move or after a node move, for implementation efficiency we compute the RHS of Eqn. 9 as its initial value. Then, after the first node move from $n_{i,1}$ to $n_{i,2}$, the above update of $p(n_i^{1\to2})$ yields its correct (approximate) new value.

2. Update the gain of all the unlocked neighbors of node u on net n_i. First consider u's neighbors $u_x \in V_1$, i.e., $u_x \in (n_{i,1} - \{u\})$. $g_{n_i}(u_x)$ is updated using Eqn. 11 and the above updated value of $p(n_i^{1\to2})$. Note that it will never be the case that n_i is currently not in the cutset, so that Eqn. 7 does not need to be used for updates. Similarly for u's neighbors $u_y \in V_2$, we update $g_{n_i}(u_y)$ by using Eqn. 12, and the updated value of $p(n_i^{1\to2})$.

   Finally, the total gains of the neighbors of u are recomputed by subtracting the previous gain components corresponding to net n_i and adding the corresponding updated gain components.

3. Recompute the probabilities of all unlocked neighbors of u.

4. Set $p(u) = 0$.

3.6 Time and Space Complexities

Recall that n is the number of nodes, e the number of nets, q_n the average number of pins per node, i.e., the number of nets it is connected to, q_e the average number of pins on a net, i.e., the number of nodes a net is connected to, $d = q_n(q_e - 1)$ is the average number of neighbors per node. We define $p = nq_n = eq_e$ as the total 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. We first consider the time complexities of the different stages of PROP.

Setting Up Data Structures: We have the standard adjacency list for nodes and nets, and also store that nets either in a bucket data structure and/or in a balanced binary AVL tree [1] using the nodes' gains as the keys. This takes $\Theta(p)$ time. It also follows from this that the space complexity is $\Theta(p)$.

Initial Node Probability and Gain Computations: Once we have the initial first-cut probabilities, we compute $p(n_i^{2\rightarrow 1})$ and $p(n_i^{1\rightarrow 2})$ for each net n_i according to Eqn. 9 [3]; if q_i is the number of pins on n_i, then this takes $\Theta(q_i)$ time, and hence for all nets this process takes $\Theta(\sum_{i=1}^{e} q_i) = \Theta(p)$ time. We then compute the $g_{n_i}(u)$ of each node by either Eqns. 5 or 7 and Eqns. 11 or 12 in constant time. Also, computing the probability of a node from its gain is a constant time operation. Thus computing $g(u)$ for node u takes $\Theta(q_n)$ time, and thus a total of $\Theta(nq_n) = \Theta(p)$ time over all nodes. Once we have the gains of all nodes, recomputing their probabilities takes a total of $\Theta(n)$ time. Since we have only 2 or 3 iterations in the initial phase, the initial phase's time complexity is $\Theta(p)$.

Choosing the Best Move: In the case of single moves and the AVL tree structure, it takes $\Theta(\log n)$ time to find the best node to move. In the case of a bucket data structure it takes a constant time to find the best node, since we only look at the topmost nonempty bucket in which there are typically very few nodes. Thus over the entire pass, the time complexity of this phase is $\Theta(n \log n)$ for single moves using AVL tree storage, and $\Theta(n)$ time using bucket storage.

Node Updation: The number of entities (nodes and nets) updated are q_n nets of the moved node u and d neighbors of u. We see from Sec. 3.5 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. This reinsertion time is on the average $\Theta(d)$ for the bucket data structure. Hence the updating process takes a total of $\Theta(nd \log n)$ time for the AVL tree or $\Theta(nd)$ time for the bucket structure for the entire pass.

[3] As noted in Sec. 3.5, even though these are not the correct values of these probabilities initially, we use these values for implementation efficiency. These probabilities are only used when conditioning them on a node move or to update these probabilities after a node move (using Eqns. 10, in which case the correct values are obtained.

Thus the time complexity of PROP for an entire pass is either $\Theta(nd) = \Theta(pq_e)$ (using the bucket data structure) or $\Theta(nd \log n) = \Theta(pq_e \log n)$ (for the AVL tree data structure). For VLSI circuits, q_e is a small constant like 4 and thus the time complexities are $\Theta(p)$ and $\Theta(p \log n)$, respectively, for the bucket and AVL tree data structures. Since the empirical evidence is that only a constant number of passes are required to obtain a min-cut 2-way partition, the time complexity of PROP is the same as that for a single pass. Finally, as mentioned above its space complexity is $\Theta(p)$.

4 A Step Further: SHRINK-PROP

An undesirable phenomenon generally occurs related to net removal probabilities when for the first time a net's node is moved: the net's removal probability either decreases or remains unchanged. This causes a lack in focus in removing this net from the cutset as the gains of its nodes on its unlocked side are not appropriately increased, and those of its nodes on it locked side are not appropriately reduced. Thus fewer than possible nets are removed from the cutset in every pass. SHRINK-PROP is an enhancement of PROP designed to counter this phenomenon and thereby increase the number of nets removed from the cutset in any pass.

We classify nets in the cutset into three categories: (1) Nets that have some nodes moved to V_1, (2) nets that have some nodes moved to V_2, and (3) *unperturbed* nets, none of whose nodes have yet been moved; the first two categories of nets are *perturbed* nets. We ignore *locked* nets whose nodes have moved to both V_1 and V_2, as they cannot be removed from the cutset in the current pass. A net in category (1) can only be moved to V_1, while one in category (2) can only be moved to V_2. Unperturbed nets can be moved to either V_1 or V_2.

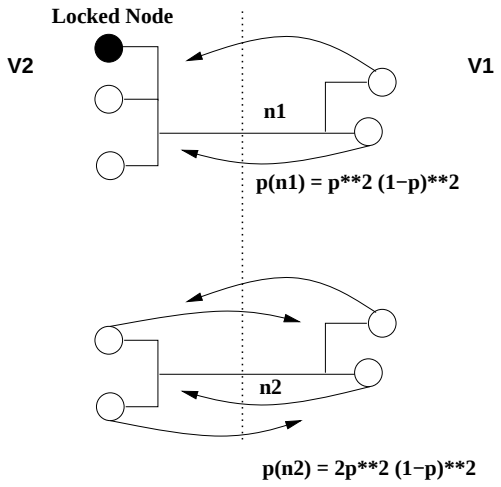


Figure 3: Movement probabilities of perturbed and unperturbed nets.

Consider Fig. 3, which shows two similarly configured nets in the cutset (each net has two unlocked

nodes in either subset). Assume for simplicity of exposition that every unlocked node has a probability p of being moved. The movement probability of the perturbed net n_1 is $p(n_1) = p^2(1-p)^2$, while that of the unperturbed net n_2 is $p(n_2) = 2p^2(1-p)^2$, since it can move to either V_1 or V_2 with probability $p^2(1-p)^2$. Note also that when n_1 was an unperturbed net, its removal probability was $p'(n_1) = p^3(1-p)^2 + p^2(1-p)^3 = p^2(1-p)^2$, which is the same as its removal probability $p(n_1)$ right after it is perturbed. In general, a newly perturbed net's removal probability either remains unchanged or decreases for the most interesting cases [4] (see [11]). Further, n_1's removal probability will be less than that of a similarly configured unperturbed net. This runs counter to a productive node-move process wherein removal probabilities of perturbed nets will be higher than those of similarly configured unperturbed nets. This enables the initial node moves made for a cut net to be proportionately rewarded by the ultimate removal of that net from the cutset.

Figure 4(a) shows the probability distribution of unperturbed nets of similar configurations, while Fig. 4(b) shows the distribution when some nets are perturbed thus decreasing their relative probabilities. As a result, the average number of nets removed from the cutset will be lower than in a process where these probability decreases are minimized, and better still, reversed. It is thus desirable that the absolute probability of a net increases when it is first perturbed. The desired probability distribution of perturbed nets relative to unperturbed ones is shown in Fig. 4(c). A simple solution to achieve this effect is to increase the weight of a net n_i by some factor when it transits from an unperturbed state to a perturbed one. This will substantially increase the gains and probabilities of nodes on n_i on its unlocked side, and decrease the gains and probabilities of its nodes on the locked side (see Eqn. 5). As a result the net's removal probability increases compared a process in which its weight remains unchanged when it enters the perturbed state.

Finally, we note that in a regular move process (one in which net weights remain unchanged), a perturbed net's removal probability will always increase (by a factor of $1/p$, where p is the probability of the moved node) whenever a node on it is moved from its unlocked to its locked side. Thus it is only necessary to take care of the case in which a net moves from an unperturbed state to a perturbed one, by increasing its weight once at that juncture.

Procedurally, we use the following efficient method for achieving weight magnification for newly perturbed nets. These steps replace the last iteration of the initial gain calculation stage in PROP (steps 2-4 in Fig. 2).

1. Multiply all node gains $g(u)$s as well as component gains $g_{n_i}(u)$s by a shrink factor f_s, $0 < f_s < 1$.
   Effect: This has the effect of decreasing the weights of all nets (initially unperturbed) by a factor of f_s.

[4] The decrease in the removal probability of newly perturbed nets occurs in iterative-improvement partitioners in which nodes are locked after being moved once in a pass. Node locking was invented in the classical Kernighan-Lin partitioner to eliminate the possibility of nodes thrashing from one side to the other [23].

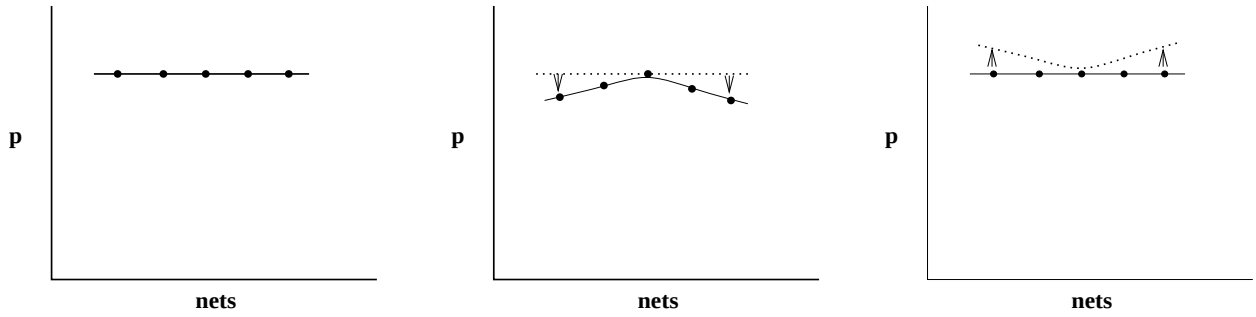


Figure 4: Removal probabilities of similarly configured nets when (a) they are initially unperturbed, (b) some of them are perturbed in a "regular" iterative-improvement process, and (c) some of them are perturbed in a "weighted" iterative-improvement process.

2. Update node probabilities of the neighbors of the moved node (keeping all net weights unchanged).
   Effect: The first time a net n_i becomes perturbed, the component gains $g_{n_i}(u)$s of its nodes are also updated for the first time, and computed according to n_i's original (non-shrunk) weight. These component gains thus experience a magnification by a factor of $1/f_s$ compared to the regular process in which all gains are not initially shrunk. For subsequent updates of these component gains, no further magnification is obtained. All this has the effect of increasing n_i's weight once by a factor of $1/f_s$ when it transits from the unperturbed to the perturbed state.

The rest of the steps are exactly the same as in Fig. 2. The complexities of SHRINK-PROP are the same as that for PROP; $\Theta(nd) = \Theta(pq_e)$ (using the bucket data structure) or $\Theta(nd \log n) = \Theta(pq_e \log n)$ (for the AVL tree data structure) time complexity, and $\Theta(p)$ space complexity.

Finally, we present a qualitative comparison of the CLIP algorithm [15] to SHRINK-PROP as both increase the weights of perturbed nets. In spite of this similarity there are key differences between these two algorithms. In CLIP, the weight increase of a perturbed net is infinite, while in SHRINK-PROP it is a finite value of $1/f_s$. More importantly, they differ in the rationale that led to these weight increases. In CLIP the rationale was enabling cluster movement across the cutline, while in SHRINK-PROP the rationale is to make the "reward" function (in terms of removal probabilities of connected nets) of the "energy" expended in node moves monotonically increasing. Of course, the SHRINK-PROP strategy implicitly also causes efficient cluster removal by causing blocks of node moves to be focused on removing clusters that span the cutline.

5 Effect of Parameter Values on Solution Quality

The major parameters in PROP are the initial probability, p_{init}, upper and lower thresholds g_{up}, g_{low} and the number of initial iterations *iter*. A classification and understanding of the effect of various value ranges for these parameters is needed for a determination of "good" parameter values that will yield high solution

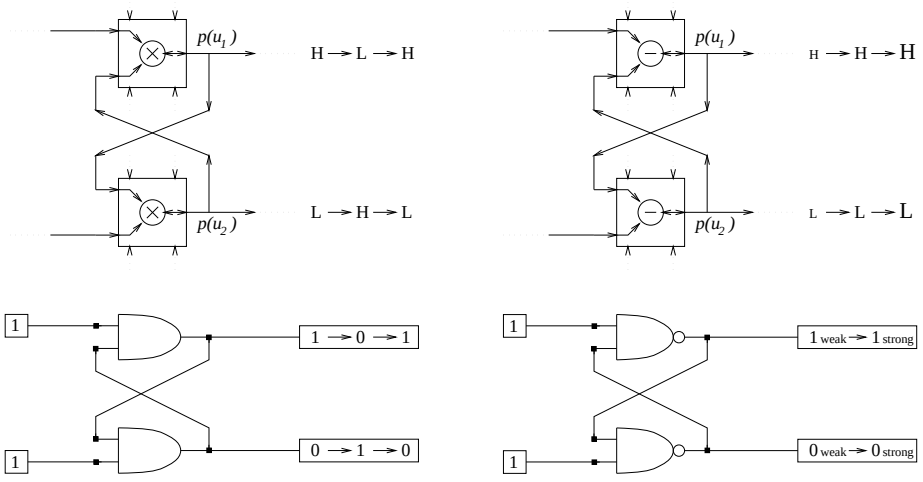
qualities for most circuits. With this in mind, we performed some experiments to categorize and understand these effects. We should note here that these experiments were done on a version of the PROP program that determined strict *non-deviated* balance-ratios, i.e., if the balance-ratio was specified to be 45-55%, then this program obtained solutions in which one subset size was strictly 45% ($\pm 0.5\%$) and the second subset size was strictly 55% ($\pm 0.5\%$) of the total node size. This is unlike results obtained in all past partitioning work as well as the final results for PROP and SHRINK-PROP presented in Sec. 6 for the best parameter values, in which a 45-55% balance ratio is interpreted as a range within which the subset sizes have to fit; thus a 48-52% ratio of subset sizes is an acceptable solution. Thus the PROP results presented in this section do not match, and are somewhat larger than those in Sec. 6, and are not comparable to those of other work. However, the conclusions drawn from the experiments of this section and the relative goodness of various parameter values hold in the "regular" balance-ratio case, since the fundamentals of correct node-move selection are the same in both cases.

5.1 Probability Convergence and the Flip-Flop Effect

The first experiment was to determine if the $p(u)$s converged over some number of initial iterations. The conclusion we reached is that the probabilities do not converge well all the time, and sometimes probabilities of nodes on certain nets completely reverse their orderings—this is called the *flip-flop* effect. It also occurs during node updates. The flip-flop effect leads to inconsistent decisions in choosing the next node to move, as a prior node choice u followed by node updates can cause a high-probability neighbor to become a somewhat lower probability one (when in fact, its probability should become even higher). This can lead to a next move of a node w that is outside u's neighborhood. Thus the partitioner can lose focus and jump from one net to the next without removing them from the cutset.

The flip-flop effect is depicted in Fig. 5, where it is modeled as two (or more) cross-coupled monotonic functions whose inputs and outputs are probabilities of neighbors. Figure 5(a) shows the effect which occurs between neighboring cells on the same side of a partition (when multiple iterations or updates after node moves occur). This effect is similar to that of cross-coupled AND gates. If the current probability of u_1 is higher than u_2 (e.g., after the first iteration at the beginning of a pass, $g(u_1) > g(u_2)$), the new probability of u_2 could become higher than u_1's because of the high probability fed by u_1 into the gain calculation of u_2 and/or low current probability of u_2 fed into u_1's gain. The probabilities and gains of the two cells will then continue to oscillate with each new iteration (of gain-probability computation) or neighbor update after a node move.

One way to minimize the flip-flop effect is to reduce the node probability-range. This limits the absolute probability changes due to the flip-flop effect, and thus minimizes its negative impact. Empirical studies showed that the best probabilities range for PROP is $[0.4, 1.0]$. Table 1 compares the results obtained for this probability range to those of the wider range $[0.1, 1.0]$—almost 8% better mincuts are obtained for the



u_1 and u_2 are on the same side

(a)

u_1 and u_2 are on opposite sides

(b)

Figure 5: Models of the flip-flop effect for (a) cells in the same side which is similar to a cross-coupled AND gates, and (b) cells in the opposite side which is similar to a cross-coupled NAND gates.

narrower range.

For neighboring cells on opposite sides of a partition [Fig 5(b)], however, their probabilities negatively affect each other's probabilities, creating an overall positive feedback on themselves. Thus instead of oscillations with new iterations, the high/low probability values are further strengthened—high values become higher and low values lower, as seen in Fig 5(b).

From these discussions, we can formulate the following parameter values for obtaining good solutions:

- The number of initial iterations *iter* can be kept at 1 in order to minimize the flip-flop effect. A reasonable discrimination between node gains is also obtained by $iter = 1$.
- However, since the flip-flop effect also manifests itself during the update process, the probability range of nodes should be kept small to minimize flip-flops during updation; an initial probability (p_{init}) value that accomplishes this naturally will lead to better solution quality (all other effects of p_{init} choice being equal).

Test Case	$\text{PROP}_{0.1-1.0}$		$\text{PROP}_{0.4-1.0}$	
	Min	**Avg**	**Min**	**Avg**
balu	27	33.2	27	32.9
p1	47	60.6	47	62.8
bm1	47	59.8	47	60.5
t4	56	70.2	52	62.2
t3	58	72.6	57	77.1
t2	91	100.2	91	101.5
t6	68	85.9	67	88.8
struct	36	49.1	36	45.6
t5	83	98.7	73	96.9
19ks	107	140.6	108	130.4
p2	145	189.0	147	196.8
s9234	45	62.1	46	67.3
biomed	84	116.5	85	111.2
s13207	74	111.2	74	112.4
s15850	66	99.2	65	95.0
industry2	223	321.2	204	313.8
industry3	278	360.1	272	384.1
s35932	73	80.9	59	87.9
s38584	74	103.5	63	103.4
avq_small	208	416.1	211	396.8
s38417	95	129.2	58	119.8
avq_large	337	441.4	251	457.2
Total	2322	3201.3	2140	3204.4
% Improvement Over PROP	-	-	7.84	-0.09

Table 1: Comparisons of cutsizes for the **non-deviated** $45-55\%$ balance criterion and 20 runs produced by PROP with probability ranges from $0.1-1.0$ and $0.4-1.0$. The **Min** column is the best result from all the runs, the **Avg** column shows the average cutsize over those runs.

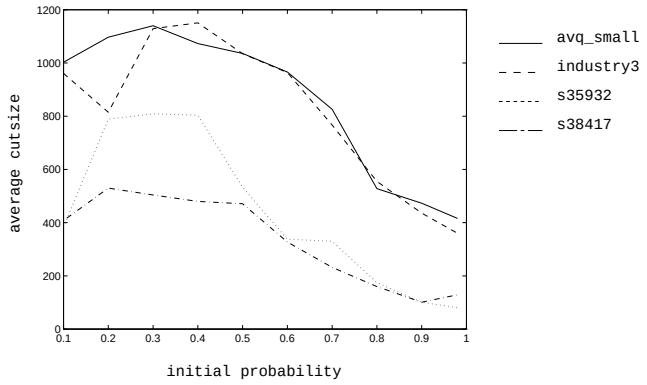


Figure 6: Average cutsize produced by four different circuit with various initial cell probabilities for 20 runs.

5.2 Thresholds

Generally, a range of [1, 1.5[for g_{up} is not discriminating enough for many circuits, i.e., many marginal nodes get a probability of 1 leading to an overly optimistic probability distribution and incorrect decisions. Similarly, g_{lo} values in the range]-1.5, -1] creates an overly pessimistic probability distribution for many circuits and thus bad node choices. The situations are reversed for $g_{up} > 2$ (this causes a pessimistic probability distribution) and $g_{lo} < -2$ (this causes a too optimistic distribution). Empirical results that bolster these arguments show that the best results are obtained for $g_{up} \in [1.5, 2]$ and $g_{lo} \in [-2, -1.5]$.

5.3 Initial Probability and the Clustering Effect

The initial probability value p_{init} has the greatest impact on solution quality. In general, the solution quality goes on improving with increasing p_{init} that is less than 1. Figure 6 shows the plot of average cutsize over various initial probabilities for four different circuits. The results were obtained for 20 runs of PROP with a thresholds of ± 1.5. It is clear that much better results are obtained for higher initial probabilities.

The reasons for this trend are:

1. Larger p_{init}'s lead to narrower probability ranges that minimizes the flip-flop effect. If the initial probability is between 0.9 and 0.98, the differences in nets' removal probabilities $p(n_i^{1\to 2})$ or $p(n_i^{2\to 1})$ will be small (e.g., $0.98^2 = 0.96$, while $0.98^4 = 0.92$). Thus node gains are small and in a narrow range, and thus the probability range is also narrow initially.

   Figure 7 shows several plots of probability distribution for circuit `s35932`; the x-axis is the probability values from 0 to 1 and the y-axis is the numbers of cells with those probabilities (the minimum

probability is set to 0.1). The plots show the probability distribution at every 10% of cell moves up to 50% which is about the maximum prefix point, for various initial probabilities—0.2, 0.5, 0.7 and 0.98. These plots clearly indicate that larger initial probabilities result in narrower probability distributions.

2. Larger p_{init}'s lead to small $g(u)$s initially (with $iter = 1$) and thus small $p(u)$s. This leads to a desired magnification (reduction) of neighbor gains on the same side (opposite side) when a node u is moved—we call this the *clustering effect* as it causes the node move process to focus on removing a cluster of tightly connected nodes straddling the cutline before nodes outside the cluster are moved. Consider for example a net n_i with 3 nodes u, u_1, u_2 in V_1 and one u_3 in V_2 shown in Fig. 8(a). For $p_{init} = 0.98$, the n_i-related gains of u, u_1, u_2 is $0.98^2 - 0.98 = -0.02$, while that for u_3 is $1 - 0.98^3 = 0.06$. Empirical results do show that for $p_{init} = 0.98$, most $g_{n_i}(u)$s are in the range of ± 0.05, and cell gains are mostly in the order of one-tenth of 1. These values are very small compared to the gain threshold, and thus cell probabilities are confined within a very small range (between 0.4 and 0.5; see Fig. 7). Figure 8(a) shows a probability distribution of the four nodes consistent with this. Note that the gain and probability computations proceed as:

$$p_{init} \rightarrow g_{n_i}(u) \rightarrow g(u) \rightarrow p(u)$$

At the end of this computation, $p(n_i^{1\rightarrow 2}) = p(u) \cdot p(u_1) \cdot p(u_2) = 0.09$ and $p(n_i^{2\rightarrow 1}) = p(u_3) = 0.4$. When node u is moved, the new g_{n_i} values become $g_{n_i}(u_1) = p(n_i^{1\rightarrow 2})/(p(u) \cdot p(u_1)) = 0.4$, $g_{n_i}(u_2) = p(n_i^{1\rightarrow 2})/(p(u) \cdot p(u_2)) = 0.45$, $g_{n_i}(u_3) = -p(n_i^{1\rightarrow 2})/p(u) = -0.18$. Thus the g_{n_i} gains of u_2, u_3 have increased greatly from their previous value of -0.02, while that of u_3 has decreased significantly from its previous value of 0.06. The resulting probabilities are shown in Fig. 8(b), and the new $p(n_i^{1\rightarrow 2}) = 0.56$. These effects will promote the removal of n_i from the cutset by making it very likely for u_1 and u_2 to be among the next few move choices.

On the other hand, the situation for $p_{init} = 0.5$ is shown in Figs. 8(c-d). As can be seen, the initial g_{n_i} values (Fig. 8(c)) are much larger in magnitude and this leads to a wider probability distribution ([0.1,1]; see Fig. 7), and such a distribution is shown in Fig. 8(c). Due to the much larger $p(u)$ value of 0.8 in this case, the $p(n_i^{1\rightarrow 2})$ value, which increases by a factor of $1/p(u)$ after u is moved, does not increase as much as in the $p_{init} = 0.98$ case. Since $p(n_i^{1\rightarrow 2})$ contributes positively to the g_{n_i} gains and probabilities of u_1 and u_2, and negatively to those of u_3, the latter do not increase as much, while the former do not decrease as much, as in the $p_{init} = 0.98$ case; see Fig. 8(d). Hence u_1 and u_2 may not have a high likelihood of being among the next few nodes chosen to be moved, and thus n_i's

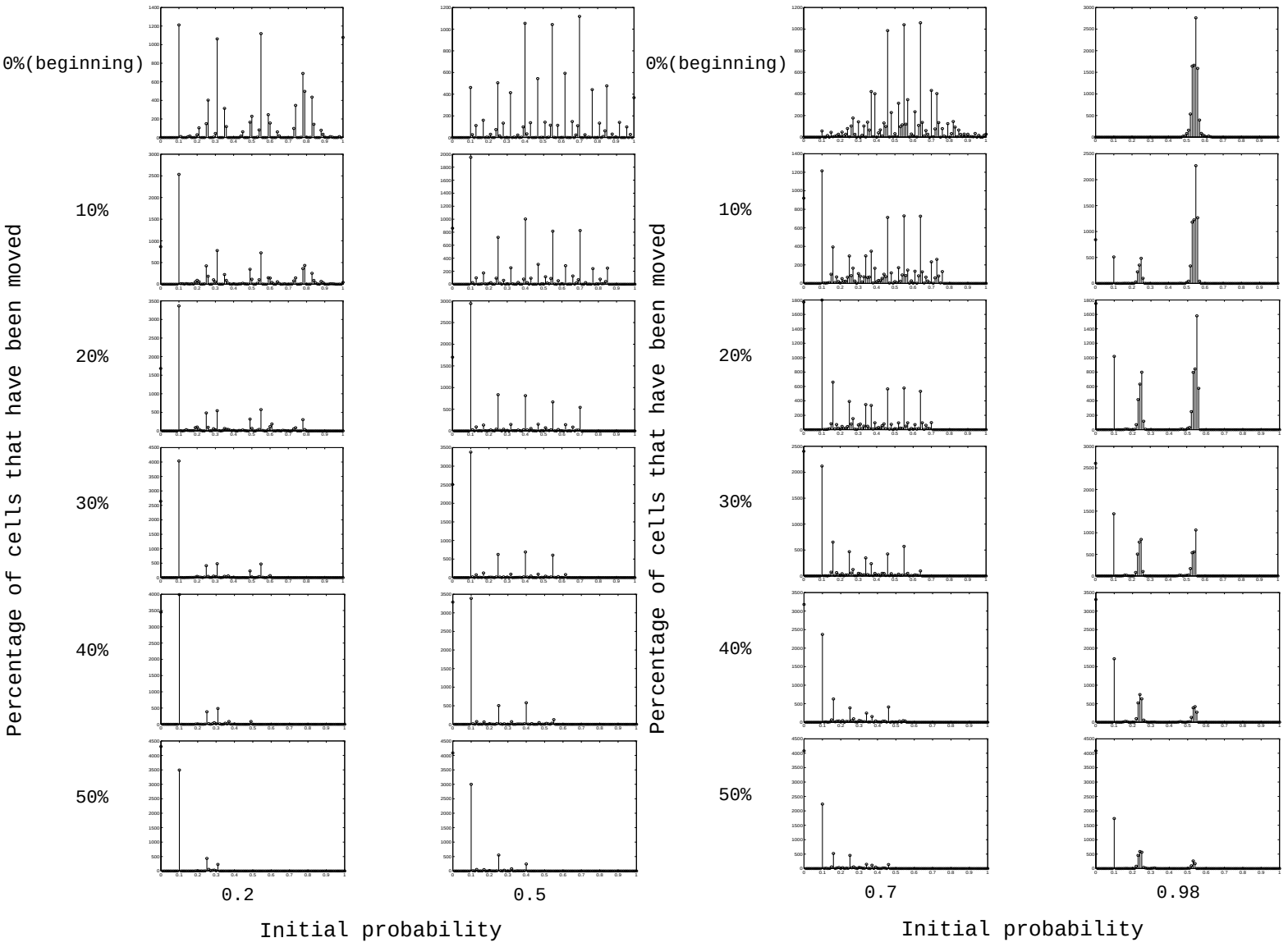


Figure 7: Probability distribution of circuit s35932 at four different stages during the partition (beginning, 10%, 20% and 30% of cells moved), and various initial probabilities (0.2, 0.5, 0.7, 0.98).

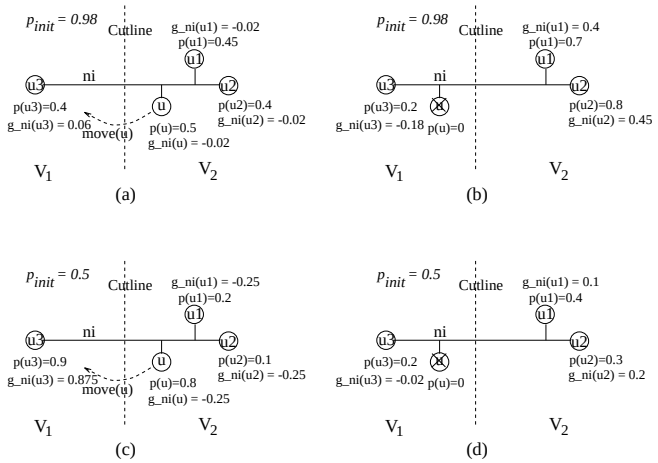


Figure 8: Effect of p_{init} value on the correct promotion and demotion of neighbor probabilities on moving node u: (a-b) Right promotion and demotion for $p_{init} = 0.98$; (c-d) Depressed promotion and demotion for $p_{init} = 0.5$.

removal from the cutset may not be correctly promoted after u's move.

5.4 Parameter Values for SHRINK-PROP

Although a high p_{init} works best for PROP, a small p_{init} of 0.2 or 0.3 works best for SHRINK-PROP along with similar threshold (g_{up}, g_{lo}) values as for PROP. The reason for this is that the SHRINK-PROP gain and probability computation is performed as:

$$p_{init} \rightarrow g_{n_i}(u) \rightarrow g(u) \rightarrow shrink(g(u)) \rightarrow p(u)$$

With a small p_{init}, we will initially obtain a wider spread of g_{n_i} values as explained in Sec. 5.3. However, the subsequent shrinking of gain values results in a narrower spread similar to that for PROP with $p_{init} = 0.98$. The resulting probability distribution is thus also similarly narrow. Thus similar positive effects of reduction of the flip-flop effect, and correct promotion and demotion of neighbor probabilities after a node move (the clustering effect) is obtained. For a high p_{init} for SHRINK-PROP, the pre-shrunk $g(u)$s will start with a narrow distribution which will be made overly narrow after shrinking—this has the undesirable effect of very small discrimination among node gains. Further, coupled with the explicit magnification and reduction by a factor of $1/f_s$, a high p_{init} also causes a disproportionate neighbor-gain magnification and reduction after a node move. This makes the gain/probability distributions so skewed that no other factors besides being neighbors of moved nodes, on the right or wrong side, have a measurable effect on node move selection. A good gain function is one which can strike a good balance between the total non-weighted gain and the weighted gain components corresponding to moved neighbors.

6 Final Experimental Results

We have obtained results on two benchmark suites, the ACM/SIGDA and ISPD-98 [2] suites, for PROP and SHRINK-PROP, as well as FM and LA for comparisons. We also compare PROP and SHRINK-PROP to various other prior and current state-of-the-art partitioners like WINDOW, EIG1, MELO, PARABOLI, GFM, GMetis and hMetis.

6.1 ACM/SIGDA Benchmarks

Cutsize Results: Tables 3 and 4 give cutsize results for the 50-50% and 45-55% balance criterion, respectively, for medium- to large-size ACM/SIGDA benchmark circuits whose number of nodes, nets and pins are given in Table 2. We assume that all node sizes are unity; all previous methods to which we compare our results make the same assumption. Furthermore, the balance condition given is strictly adhered to for every move in a pass (no violating moves are allowed at any point) as specified in Fig. 2. For both the 50-50% and 45-55% balance criterion, PROP uses the following parameter values: $p_{init} = 0.98$, $p_{max} = 1.0$, $p_{min} = 0.4$, the linear probability function, $g_{up} = 1.75$, and $g_{lo} = -1.75$. SHRINK-PROP uses the parameter values: $p_{init} = 0.3$ or 0.2 that is dynamically chosen based on whether the circuit is large ($>$ 10,000 nodes) and relatively dense (i.e., the average number of pins per cell is greater then 3.5 or much greater then the average number of pins per net), or not, respectively, $p_{max} = 1.0$, $p_{min} = 0.1$, the linear probability function, $g_{up} = 1.75$, and $g_{lo} = -1.75$, and the shrink factor $f_s = 0.1$. The cutsets generated by SHRINK-PROP were further refined by PROP (using the above PROP parameters); the times reported for SHRINK-PROP include the post-processing time for PROP. Percentage improvements are calculated as (mincut improvement/larger mincut)$\times 100$.

Table 3 compares cutset sizes for the 50-50% balance criterion produced by FM_{20}, FM_{40}, FM_{100} (FM run with 20, 40 and 100 initial random partitions, respectively), WINDOW [7] (which uses FM_{20} as a final phase), and PROP and SHRINK-PROP, both using 20 runs. The circuits for which these results are shown

Test Case	# of Cells	# of Nets	# of Pins	Test Case	# of Cells	# of Nets	# of Pins
balu	801	735	2697	s9234	5866	5844	14065
p1	833	902	2908	biomed	6514	5742	21040
bm1	882	903	2910	s13207	8772	8651	20606
t4	1515	1658	5975	s15850	10470	10383	24712
t3	1607	1618	5807	industry2	12637	13419	48404
t2	1663	1720	6134	industry3	15406	21924	68290
t6	1752	1541	6638	s35932	18148	17828	48145
struct	1952	1920	5471	s38584	20995	20717	55203
t5	2595	2750	10076	s38417	23949	23843	57613
19ks	2844	3282	10547	golem3	103048	144949	338419
p2	3014	3029	11219				

Table 2: Benchmark circuit characteristics.

Test Case	Cut Sizes: 50-50% Balance							
	Previous Methods						Probability-Based Methods	
	FM_{100}	FM_{40}	FM_{20}	LA-2_{20}	LA-3_{20}	WIN-DOW	$PROP_{20}$	SHRINK-$PROP_{20}$
balu	32	32	32	31	31		32	31
p1	57	57	59	59	55	60	59	54
bm1	55	57	65	58	55	70	54	52
t4	86	88	97	94	88	61	58	54
t3	72	72	72	78	90	67	58	58
t2	115	115	115	124	105	105	91	90
t6	71	71	71	70	63	70	81	66
struct	45	47	52	45	45		38	34
t5	97	97	149	109	96	101	82	76
19ks	142	150	150	141	153	136	120	116
p2	236	238	245	215	183	258	154	146
s9234	53	59	59	57	58		55	49
biomed	83	83	83	122	91	164	88	83
s13207	92	101	101	81	89		83	72
s15850	112	120	120	122	75		73	69
industry2	428	501	501	492	378	392	254	199
Total Cuts	1776	1888	1971	1898	1655	1484	1380/1099 (all/11 ckts)	1249/994 (all/11 ckts)

PROP-Based Methods	Cut Size Improvements							
$PROP_{20}$	22.3%	26.9%	30.0%	27.3%	16.6%	25.9%	-	-
SHRINK-$PROP_{20}$	29.7%	33.8%	36.6%	34.2%	24.5%	33%	9.5%	-
CPU Time (secs)	2555 (b) 7501 (t)	1022 (b) 3000 (t)	511 (b) 1500 (t)	1181 (b)	5331 (b)	> 1922 (3 ckts.)	2383/1255 (t) (all/3 ckts)	4433/2610 (t) (all/3 ckts)

Table 3: Comparisons of cutset sizes of 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), WINDOW [7] in which clustering is followed by 20 runs of FM, and PROP and SHRINK-PROP, each with 20 runs. For PROP and SHRINK-PROP, total cuts are shown for all circuits and the 11 circuits for which results have been reported for WINDOW, and the total CPU times are given for all circuits and for the 3 circuits whose times were reported for WINDOW. In the CPU Time row, a '(b)' indicates a bucket data structure and a '(t)' a tree data structure, and is used to differentiate the effect of the data structure from the rest of the algorithm's time complexity—the bucket data structure, which is more time-efficient, can only be used if all net costs are assumed to be uniform, which is not the case for some objectives of partition-driven placement like performance optimization.

Test Case	Cut Sizes: 45-55% Balance							
	Previous Methods					Probability-Based Methods		
	MELO	Paraboli	EIG1	GFM	GMetis	$PROP_{20}$	$PROP_{80}$	$SHRINK\text{-}PROP_{20}$
balu	28	41	110	27	27	27	27	27
p1	64	53	75	47	47	47	47	47
bm1	48		75		48	47	47	47
t4	61		207		49	50	50	48
t3	60		85		62	58	57	57
t2	106		196		95	88	88	88
t6	90		295		94	63	63	60
struct	38	40	49	41	33	34	33	33
t5	102		167		104	74	74	72
19ks	119		179		106	107	105	104
p2	169	146	254	139	142	143	143	143
s9234	79	74	166	41	43	45	45	41
biomed	115	135	286	84	102	84	84	83
s13207	104	91	110	66	74	80	67	68
s15850	52	91	125	63	53	69	52	58
industry2	319	193	525	211	177	213	181	186
industry3		267	399	241	243	249	244	243
s35932		62	105	41	57	59	49	46
s38584		55	76	47	53	62	52	49
s38417		49	121	81	69	58	58	58
golem3		1629	5379		2111	1465	1425	1374

Total Cut Sizes								
13 circuits				1129		1170	1082	1082
14 circuits		2926				2635	2507	2456
16 circuits	1554					1229	1163	1162
21 circuits			8984		3789	3122	2991	2932

PROP-Based Methods	Cut Size Improvements							
$PROP_{20}$	20.9%	9.95%	65.2%	-3.5%	17.6%	-	-	-
$PROP_{80}$	25.2%	14.3%	66.7%	4.2%	21.1%	4.2%	-	-
$SHRINK\text{-}PROP_{20}$	25.2%	16.0%	67.4%	4.2%	22.6%	6.1%	2.0%	-

Table 4: Comparisons of cutset sizes of ACM/SIGDA benchmark circuits for the 45-55% balance criterion produced by our new algorithms, PROP (20 and 80 runs) and SHRINK-PROP (20 runs), to previous state-of-the-art techniques, MELO [5], PARABOLI [28], EIG1 [20], GFM [26] and GMetis [4].

are the same as those for which results were reported in [14], and are a subset of the circuits given in Table 2. PROP and SHRINK-PROP cutsizes are on the average 30% and 36.6% better than FM_{20}, respectively, and 22% and 29.7% better than FM_{100}, respectively. 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 and SHRINK-PROP also yield 27% and 34% better results than LA-2_{20}, 16.2% and 24% better cutsizes than LA-2_{40}, and about 17% and 24.5% better results than LA-3_{20}, respectively. They also perform 26% and 33% better than WINDOW, respectively. SHRINK-PROP yields 9.5% better results than PROP, at a cost of being only a factor of two slower; see last row of Table 3.

In Table 4, we compare the performance of $PROP_{20}$, $PROP_{80}$ and $SHRINK\text{-}PROP_{20}$ to recent state-of-the-art methods like EIG1 [20], PARABOLI [28], MELO [5], GFM [26] and GMetis [4] for the 45-55% balance case. The probability-based methods yield much better cutsizes than these other methods. $PROP_{20}$ is 65% better than EIG1, 21% better than MELO, 10% better than PARABOLI, and 17.6% better than GMetis. It is, however, a little worse (by 3.5%) than GFM, which is about 8.5 times slower than $PROP_{20}$, assuming Sparc 5 and Sparc 10 machines have comparable speeds; see footnote 6. We thus also obtained results for $PROP_{80}$, and it performs 4.2% better than GFM, while still being twice as fast. $SHRINK\text{-}PROP_{20}$ is 67.4% better than EIG1, 25% better than MELO, 16% better than PARABOLI, 22.6% better than GMetis, 4.2% better than GFM, and 6.1% better than $PROP_{20}$. It is, however, only a factor of two slower than $PROP_{20}$, and thus is four times faster than GFM (see Table 5).

Timing Results: We showed earlier that the time complexities of PROP and SHRINK-PROP are $\Theta(nd \log n) = \Theta(pq_e \log n)$ for a binary-search tree data structure. FM on the other hand has a complexity of $\Theta(nd)$ due to the use of a bucket data structure, which can only be used if all net costs are assumed to be one (or uniform). This is the case in our implementation of FM, and it is thus very fast; PROP is about 4.6 times slower than FM per run (but obtains 30% better results). If the assumption of unit net cost cannot be made, as in the case when circuits are partitioned to minimize timing [27, 21] or graph models of netlists are partitioned [5], then FM will require an initial $\Theta(n \log n)$ sorting routine plus a $\Theta(\log n)$ insertion routine (for a binary search tree like AVL) for every updated neighbor after a move, making its time complexity also $\Theta(nd \log n)$—this slows it down by a factor of $\log n$. However, PROP will have the same complexity under non-uniform net costs. In order to demonstrate how PROP compares to both versions of FM, i.e., with bucket (b) and tree (t) data structures, we have implemented both of these and tabulated CPU times for different ACM/SIGDA benchmark circuits in Tables 3 and 5.

The last row of Table 3 shows total times for both bucket and tree data structures for FM, while Table 5 shows individual times per circuit and total times for FM_{100} for the tree data structure. Table 5 also gives the run times for these circuits for all the other algorithms compared here[5]. From Tables 3 (last row) and 5,

[5]It should be noted that all run times obtained for PROP and SHRINK-PROP (as well as for FM and LA) were on multi-user

CPU Times (in Seconds) Per Circuit											
Test Case	**Sun Sparc-5**					**DEC 3000 Model 500 AXP**		**Sun Sparc-10**			**Sun Sparc-5**
	FM_{100}	$LA\text{-}2_{40}$	$LA\text{-}3_{20}$	$PROP_{20}$	SHRINK-$PROP_{20}$	EIG1	Paraboli	MELO	WIN-DOW	GFM	GMetis
balu	42	17	19	16	18	6	16	7		24	14
p1	57	20	22	19	25	3	18	8		16	12
bm1	58	22	22	20	26			4			12
t4	102	49	791	49	67			24			21
t3	133	47	466	51	64			27			23
t2	118	58	820	64	79			29			26
t6	109	35	27	75	110			31			32
struct	115	62	43	42	50	7	35	38		80	27
t5	213	91	1397	97	149			67			46
19ks	312	106	105	87	137			79			39
p2	353	115	99	139	276	18	137	89	> 116	224	53
s9234	664	188	141	139	228	24	490	516		672	58
biomed	724	222	175	250	404	521	711	496	> 421	1440	95
s13207	1023	286	207	177	360	44	2060	710		1920	102
s15850	1060	342	262	291	510	78	2731	1197		2560	114
industry2	2418	702	741	867	1930	707	1367	1855	> 1385	4320	245
industry3				850	1500	195	761			4000	299
s35932				597	741	2067	2627			10160	266
s38584				1070	1795	348	6517			9680	397
s38417				949	1509	281	2041			11280	281
golem3				6554	17314	1893	10822				450
Total CPU Times (Secs)											
3 circuits				1255	2610				> 1922		
13 circuits				5406	9346					46376	
14 circuits				11960	26660	6192	30333				
16 circuits	7501	2361	5331	2383	4433			5177			
21 circuits				12403	27292						2612

Table 5: CPU times in seconds for various partitioners, for each circuit and total times over all circuits.

Test Case	# Cells	# Nets	# Pins	Test Case	# Cells	# Nets	# Pins
ibm01	12752	14111	50566	ibm10	69429	75196	297567
ibm02	19601	19584	81199	ibm11	70558	81454	280786
ibm03	23136	27401	93573	ibm12	71706	77240	317760
ibm04	27507	31970	105859	ibm13	84199	99666	357075
ibm05	29347	28446	126308	ibm14	147605	152772	546816
ibm06	32498	34826	128182	ibm15	161570	186608	715823
ibm07	45926	48117	175639	ibm16	183484	190048	778823
ibm08	51309	50513	204890	ibm17	185495	189581	860036
ibm09	53395	50513	204890	ibm18	210613	201920	819697

Table 6: ISPD-98 Benchmark circuit characteristics.

it is easy to see that PROP is among the fastest partitioners. It is very comparable to FM_{100}(b) and LA-2_{40} (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 65% better cutsizes than EIG1. Assuming that the Sun Sparc 5, Sparc 10 and the DEC 3000 are comparable in speed[6], $PROP_{20}$ is about 8.5 times faster than GFM ($PROP_{80}$ is thus more than twice as fast as GFM), 3.15 times faster than FM_{100}(t), 2.5 times faster then PARABOLI, about 2.2 times faster than LA-3_{20} and MELO, and at least 1.5 times faster than WINDOW. $PROP_{20}$ is, however, 4.7 times slower than GMetis (but obtains 17.6% better results), which is a multilevel technique, and derives its speed from that paradigm (all other methods compared here work on flat netlists). This paradigm is orthogonal to the basic partitioning technique, and it is possible to cast our methods also in a multilevel framework to obtain faster and even better results—this will be investigated in future work. SHRINK-$PROP_{20}$ obtains 6.2% better results than $PROP_{20}$ with a time-penalty factor of only two; it is thus also faster than most previous state-of-the-art techniques.

6.2 ISPD-98 Benchmarks

Table 6 gives the characteristics of the 18 circuits in the ISPD-98 benchmark suite [2]. These circuits range from medium size (more than 10K nodes and nets, and 50K pins) to very large (more than 200K nodes and nets, and 800K pins). FM (bucket data structure), PROP and SHRINK-PROP results were obtained for these circuits on a Sun Ultra Sparc1 workstation. Table 7 compares the mincuts, average-cuts and times per run over 20 runs of these partitioners. PROP-20 obtains 24% better mincuts and 43% better average-cuts than FM-20; it is about 4.7 times slower than FM-20. However, since FM is extremely fast, this is still a tolerable slowdown for PROP, especially given the tremendous cutsize improvements it offers. SHRINK-

workstations, and were obtained when other user programs were running, many times with higher priority.

[6] The SPECmarks of these machines indicate that the DEC 3000 is faster than the Sparc 5 (Model 85) by a factor of 1.5 to two, and the Sparc 5 is faster than the Sparc 10 by a factor of 1.5.

Test Case	FM 20-runs			PROP 20-runs			SHRINK-PROP 20-runs		
	Min	**Avg**	**Time**	**Min**	**Avg**	**Time**	**Min**	**Avg**	**Time**
ibm01	193	487.6	6.2	185	230.4	25.5	181	197.8	43.8
ibm02	277	456.4	10.4	275	458.8	78.2	263	333.9	134.8
ibm03	1357	2227.3	24.3	1027	1286.3	81.8	978	1140.2	158.5
ibm04	735	1183.4	22.1	588	654.2	88.1	548	595.9	134.5
ibm05	2300	3025.5	34.2	2024	2548.9	128.5	1715	1818.1	524.9
ibm06	1015	1452.5	31.4	982	1150.8	127.6	913	1029.6	257.0
ibm07	1133	2179.9	36.9	871	973.6	156.0	847	918.0	279.4
ibm08	1712	2844.6	50.9	1284	1781.3	215.6	1165	1498.5	580.4
ibm09	1285	2431.4	51.4	683	1048.4	240.6	628	862.8	373.1
ibm10	1903	2584.6	66.0	1423	1755.9	308.8	1367	1611	683.7
ibm11	1692	4031.6	68.0	1009	1199.9	265.5	981	1103.3	567.8
ibm12	2288	3459.8	76.7	2242*	3257.5*	381.3*	2081	2711.7	962.8
ibm13	1163	2438.7	75.1	950	1164.3	384.8	943	1124.5	695.7
ibm14	2647	6835.7	185.5	2164	2735.9	622.7	1902	2399.6	2276.2
ibm15	5428	8050.8	181.6	2977	3613.1	872.5	2622	3080.0	2238.6
ibm16	2687	5659.9	169.9	2277	3233.2	1157.3	2009	2416.7	3316.9
ibm17	3861	7251.7	229.8	2432	4151.8	1071.9	2369	3053.4	4563.7
ibm18	1555	3150.7	238.2	1879	2794	1109.1	1544	2242.8	4106.8
Total	33231	59752.5	1559.3	25272	34038.8	7316.5	23056	28129	21898.6
Impr. over FM-20	-	-	-	24%	43%	4.7×	31%	53%	14×
Impr. over PROP-20	-	-	-	-	-	-	8.8%	17.4%	3×

* The PROP results for ibm12 are reported for parameters $p_{init} = 0.98, g_{up} = 1.5, g_{lo} = -1.5, iter = 1, min = 0.1$, as the PROP parameters used for the other circuits (see caption below) yield results for the circuit that are not representative of PROP.

Table 7: Comparisons of cutsizes of ISPD-98 circuits for the $45 - 55\%$ balance criterion produced by FM, PROP, and SHRINK-PROP, all for 20 runs. Except for ibm12 (see * above), PROP parameters used are $p_{init} = 0.98, g_{up} = 2, g_{lo} = -2, iter = 1, min = 0.1$. SHRINK-PROP parameters used are $p_{init} = 0.3, g_{up} = 1.5, g_{lo} = -1.5, iter = 1, min = 0.1, g_s = 0.1$. The SHRINK-PROP results are further refined by PROP with parameters $p_{init} = 0.3, g_{up} = 1.75, g_{lo} = -1.75, iter = 1, min = 0.1$. The **Min** column is the best result from all the runs, the **Avg** column shows the average cutsize over those runs, and the **Time** column is the CPU time (in seconds) per run.

PROP-20 is 31% and 53% better in mincuts and average-cuts, respectively, over FM-20; it is about 14 times slower. Further, SHRINK-PROP-20 obtains 8.8% better mincuts and 17.4% better average-cuts than PROP-20, while being about three times slower than it. These results also show that the average-cut improvements of PROP and SHRINK-PROP over FM, and that of SHRINK-PROP over PROP is significantly more than the respective mincut improvements (which is itself substantial). This leads to the conclusion that as we proceed from FM to PROP to SHRINK-PROP, the partitioners become more stable and less reliant on the initial random cut for obtaining the final mincut. Another way of looking at this is the percentage difference between the average-cut and mincut for each partitioner: 44% for FM, 26% for PROP, and only 18% for SHRINK-PROP.

We also note that both PROP and SHRINK-PROP are "flat" partitioners, i.e., they do not use the multi-level paradigm use in many recent partitioners like hMetis [22], ML_c [3], and LSR/MFFS [8]. The multilevel paradigm is orthogonal to our techniques, and can be used with both PROP and SHRINK-PROP to yield faster and probably better results. Further, flat partitioners play an important role in complex problems like timing-driven placement [12, 27] in the presence of multiple constraints [13, 25]), where it may be detrimental to "hide" useful information about the circuit by clustering subcircuits into large meta-nodes, as is done in multilevel partitioners. It is, however, interesting to note that the SHRINK-PROP mincut results are within 2.5% of those of hMetis (total mincut of 22473 over all circuits of Table 6), one of the best multilevel partitioners, for the ISPD-98 circuits as reported in [2].

7 Conclusions

We have presented two probabilistic-gain based approaches PROP and SHRINK-PROP for iterative-improvement min-cut partitioning. The methodologies are based on futuristic lookahead gains based on probability and conditional probability formulations, and on clustering effects caused by correct promotion and demotion of neighbors of moved nodes. Results obtained for the ACM/SIGDA and ISPD-98 benchmark circuit suites show that our partitioners outperform other iterative-improvement methods like FM and LA by wide margins, and that we also outperform recent state-of-the-art partitioners like WINDOW, MELO, EIG1, Paraboli, GFM and GMetis by significant margins. The run times of PROP also compare favorably with those of the iterative-improvement and other recent techniques; it is comparable to those of FM and LA, and much faster than those of most recent state-of-the-art methods. SHRINK-PROP is slower than PROP by a factor of about two (with the benefit of yielding 6.2% better results) for the ACM/SIGDA suite, and by a factor three for the ISPD-98 suite (while yielding 8.8% better mincuts). SHRINK-PROP is thus also faster than most previous and current flat (i.e., non-multilevel) state-of-the-art techniques. It is also promising that SHRINK-PROP's mincut results are within 2.5% of those of hMetis, which is one the best multilevel partitioners. Multilevel partitioning is a paradigm that can be combined with PROP and SHRINK-PROP to obtain even better results. However, flat partitioners such as PROP and SHRINK-PROP are probably more

suitable for complex placement problems, such as those for sub-micron VLSI with multiple objectives and constraints.

Besides the significant improvements in mincut results, our new techniques have also demonstrated that the iterative-improvement approach can be effectively used with more informed gain functions to partition very large circuits. The significance of this new-found efficacy of the iterative-improvement paradigm is that it has many advantages that can now be readily exploited. Chief among them is that the node-move process inherent in this approach lends itself to a high degree of flexibility in tuning the objective of the partitioning process to numerous goals like min-cut, delay minimization and power minimization. This flexibility also facilitates the partitioning process to work with almost any given design constraints like required size ratio between partition subsets, crosstalk bounding and uniform thermal distribution. This means that iterative-improvement based techniques can be readily adapted to changing design flows and process technologies, and are thus scalable with time. Other partitioning approaches are not so flexible or scalable. Hence iterative-improvement, with new gain functions such as those in PROP and SHRINK-PROP, emerges as a powerful, yet efficient, paradigm for partitioning and placing large and complex circuits.

The probabilistic-gain based iterative-improvement approach also opens up a number of exciting possibilities, for example, k-way partitioning, multiple-FPGA partitioning, and timing-driven partitioning, that can also be tackled using probability-based lookahead concepts. These issues will be explored in future research.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison Wesley, Reading, Massachussetts, 1974.

[2] C.J. Alpert, "The ISPD98 circuit benchmark suite", *Proc. ACM Int'l Symp. on Physical Design*, April 1998, pp. 80-85.

[3] C.J. Alpert, J-H. Huang and A.B. Kahng, "Multilevel circuit partitioning", *Proc. Design Automation Conf.*, June 1997, pp. 530-533.

[4] C.J. Alpert and A.B. Kahng, "A hybrid multilevel/genetic approach for circuit partitioning", *Physical Design Workshop, 1996*, pp. 100-105.

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

[6] C.J. Alpert and A.B. Kahng, "Recent directions in netlist partitioning: A survey", *Integration, The VLSI Journal*, 19(1-2), 1995, pp. 1-81.

[7] 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.

[8] J. Cong, et al., "Large scale circuit partitioning with loose/stable net removal and signal flow based clustering", *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design*, Nov. 1997, pp. 441-446.

[9] J. Cong and S. K. Lim, "Multiway Partitioning with Pairwise Movement", *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 512-516, 1998.

[10] S. Devadas, A. Ghosh and K. Keutzer, *Logic Synthesis*, McGraw Hill, 1994.

[11] S. Dutt and W. Deng, "Probability-Based Approaches to VLSI Circuit Partitioning", Technical Report, University of Illinois at Chicago, 1999. – Available on web site www.eecs.uic.edu/ dutt/publ.html.

[12] S. Dutt, "A Stochastic Approach to Timing-Driven Partitioning and Placement with Accurate Net and Gain Modeling", *TAU97: IEEE/ACM Int. Workshop on Timing Issues in Digital Systems*, Dec. 1997, pp. 246-256.

[13] S. Dutt and H. Theny, "Partitioning Around Roadblocks: Tackling Constraints with Intermediate Relaxations", *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design*, IEEE/ACM ICCAD, Nov., 1997, pp. 349-355.

[14] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", *Proc. IEEE/ACM Design Automation Conference*, June 1996, pp. 100-105—**Best-Paper Award**.

[15] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques, *Proc. IEEE/ACM International Conference on CAD,* Nov. 1996, pp. 194-200.

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

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

[18] J. Garbers, H.J. Promel and A. Steger, "Finding clusters in VLSI circuits", *Proc. Int'l. Conf. Computer-Aided Design*, 1990, pp. 520-523.

[19] M.R. Garey and D.S. Johnson, *Computers and Intractability*, W.H. Freeman and Company, New York, pp. 209-210.

[20] 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.

[21] 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.

[22] G. Karypis, et al., "Multilevel hypergraph partitioning: Application in VLSI domain", *Proc. Design Automation Conf.*, June 1997, pp. 526-529.

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

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

[25] R. Kuznar, F. Brglez and K. Kozminski, "Cost minimization of partitions into multiple devices", *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 315-320.

[26] J. Li, J. Lillis and C-K. Cheng, "Linear decomposition algorithm for VLSI design applications", *Proc. IEEE/ACM International Conference on CAD*, 1995, pp. 223-228.

[27] M. Marek-Sadowska, "Issues in timing driven layout", in *Algorithmic Aspects of VLSI Layout*, pp. 1-24, M. Sarrafzadeh and D.T. Lee, eds., World Scientific Publ. Co., 1993.

[28] 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.

[29] Y. G. Saab, "A Fast and Robust Network Bisection Algorithm", *IEEE Trans. Computers*, 1995, pp. 903-913.

[30] 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.

[31] C. Sechen and D. Chen, “An improved objective function for mincut circuit partitioning”, *Proc. Int’l. Conf. Computer-Aided Design*, Vol. 38, No. 1, Jan. 1989, pp. 62-81.

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

[33] H. Stark and J.W. Wood, *Probability, Random Processes, and Estimation Theory for Engineers*, Prentice Hall, 1986.

[34] Y. C. Wei and C. K. Cheng, “An Improved Two-way Partitioning Algorithm with Stable Performance”, *IEEE Trans. on Computer-Aided Design*, 1990, pp. 1502-1511.

[35] 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.

[36] 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.