A Fast and Stable Hybrid Genetic Algorithm for the Ratio-Cut Partitioning Problem on Hypergraphs

Thang Nguyen Bui
Dept. of Computer Science
Pennsylvania State University
Middletown, PA 17057

Byung Ro Moon
Dept. of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802

Abstract—A genetic algorithm (GA) for partitioning a hypergraph into two disjoint graphs of least ratio-cut is presented. Two notable features of this algorithm are: (i) a fast local optimizer, and (ii) a preprocessing step. Some supporting combinatorial arguments for the preprocessing heuristic are also provided. Experimental results on industrial benchmarks are favorable when compared with recently published algorithms [25], [26], [19].

I. Introduction

It is generally believed that genetic algorithms (GAs) are slower than other algorithms for doing the same tasks. This is true for most genetic algorithms since genetic algorithms need a large number of generations to converge to a good set of solutions. Hybrid type GAs, which perform local optimization in every generation, are often used to achieve faster convergence and hence the number of generations is decreased. Although the running time can be shortened by this, they still usually take significantly longer time than other algorithms since each local optimization can take a considerable amount of time. Consequently, researchers in GA usually pay more attention to the quality of the solutions than their running time. In this paper, we propose a fast hybrid genetic algorithm which uses a carefully designed fast local optimization algorithm without sacrificing the quality of the solutions.

Let $G = (V, E)$ be a hypergraph, where $V$ is the set of vertices with weights, and $E$ is the set of hyperedges. A hyperedge $e$ is defined as a set of vertices. When a circuit is mapped into this graph model, the modules are mapped to vertices and the nets are mapped to hyperedges. A bipartition of $G$ is a partitioning of the vertex set $V$ into 2 disjoint sets $A$ and $B$. The total weight of the nets having at least one end point in each of $A$ and $B$ is called the size or cut size of the bipartition. For a bipartition to be practically useful, it needs to have some balance requirement between $|A|$ and $|B|$. The ratio-cut of a partition $(A, B)$ is defined as

\[
\text{cut size of the partition} / |A| |B|
\]

The ratio-cut gives penalties proportional to the degree of imbalance. Ratio cut has been used frequently in recent works as a metric to judge the quality of partitions [25], [19], [13], [9]. The ratio-cut problem is the problem of finding a partition with the minimum ratio-cut. It is known to be NP-hard [25].

The most successful partitioning techniques are group migration, simulated annealing, and spectral methods. Kernighan-Lin algorithm (KL) [21], [24] and Fiduccia-Mattheyses algorithm (FM) [16] are the two most basic group migration heuristics. FM is a variation of KL and achieves the time complexity of $O(|E|)$ by using carefully designed data structures which are currently used by several researchers [25], [13]. An important property of these group migration algorithms is that they are highly dependent on the initial solution. To effectively use this property, several researchers suggested preprocessing heuristics to provide good initial solutions [3], [4], [25], [10], [13], [23]. The most extensive report on the graph partitioning problem (particularly on the bisection problem) using simulated annealing was done by Johnson et al. [20] and competitive results were reported. Another common approach to the graph partitioning problem is to use spectral methods which have shown promising results [23], [19], [1]. They use the second smallest eigenvalue of the spectral matrix of the input graph, and the corresponding eigenvector as the main clue for clusters detection.

Recently several partitioning algorithms using genetic algorithm have been reported [22], [11], [12], [2]. Most of them showed moderate success. The authors also used genetic algorithms for graph partitioning problem in [5] and [6], and provided extensive testing results on 52 theoretical benchmark graphs and showed comfortably better result than that of simulated annealing. Results of [5] and [6] were on the graph bisection problem which requires that the graph be divided exactly in half. In this paper we consider the ratio-cut problem for VLSI circuit graphs, i.e., hypergraphs. In [6], the authors suggested the technique of preprocessing heuristics to achieve a small performance improvement due to the preprocessing heuristics. In this paper, we adapt and extend the technique to hypergraphs and weighted graphs. We tested the algorithm on the MCNC benchmark circuits and some other circuits. Experiments with this algorithm showed the most stable results with visible improvement over recently published results [25], [26], [19].

The rest of this paper is organized as follows. In Section II, we describe our genetic algorithm for the ratio-cut problem (GCA). In Section III, we present a preprocessing heuristic for our genetic algorithm called Weighted Depth-First-Search Reordering (WDER). Section IV provides some combinatorial arguments on the time complexity and supporting arguments for the preprocessing heuristic. Section V gives our experimental results and compares the results against others.

II. The Genetic Algorithm for Ratio-Cut

A chromosome of length $n$ on a binary alphabet is an $n$-tuple $<c_1,c_2,\ldots,c_n>$ where $c_i \in \{0,1\}$ for $i = 1,2,\ldots,n$. Each chromosome represents a solution to the problem. The terms in each of the following pairs are used interchangeably in the paper: (chromosome, solution), (vertex, module), (edge, net), (graph, circuit) and (local optimization, local improvement).

A. Overview of the Algorithm

A genetic algorithm starts with a set of randomly generated initial solutions, which is called a population. This population is then manipulated, or evolves, into different populations for several (frequently, hundreds of) generations. At the end the algorithm returns the best member of the population as the
solution to the problem. For each generation, the evolution process proceeds as follows. Two members of the population are chosen, based on some probability distribution. These two members are then combined through a crossover operator to produce an offspring. With a low probability, this offspring is then modified by a mutation operator to introduce new genes to the population, enhancing its diversity. A local optimization technique is then applied to the offspring to improve it (hybrid type GA). The offspring is then tested to see if it is suitable for the population. If it is, the algorithm uses a replacement scheme to replace a member of the population with the new offspring. We now have a new population and the evolution process is repeated until certain condition is met. This type of GA (which has an offspring per generation) is called steady-state GA as opposed to generational GA (which replaces the entire population in each generation). Detailed descriptions of the above genetic operators will be given later in the paper.

We optionally add a preprocessing step before GA starts running. The technique of preprocessing for GAs was first suggested in [5] and [6]. In the latter paper, three preprocessing heuristics for GAs solving graph optimization problems on unweighted graphs were introduced. In this paper, we present an extension of the technique for hypergraphs and weighted graphs. The overall structure of the algorithm is given below. We will refer to our algorithm for the ratio-cut problem as the Genetic Ratio-Cut Algorithm or GRCA.

preprocess; /* optional */
create an initial population; /* population size is 50 */
do {
    choose parent1 and parent2 from population;
    offspring = crossover(parent1, parent2);
    mutation(offspring);
    local-improvement(offspring); /* a modified FM alg. */
    if suited(offspring) then replace(offspring);
}while (stopping criterion not satisfied);
return the best member of the population;

B. A Location-Based Encoding

Each solution in the population is represented by a chromosome. In our problem a chromosome corresponds to a bipartition of the graph. The number of genes in a chromosome is the number of modules in the graph. Each module has a corresponding location on the chromosomes. A location has value 0 if the corresponding module is on the left side of the bipartition, and has value 1 otherwise. This type of encoding where each gene location has an explicit meaning is called location-based encoding. The fact that we used location-based encoding is very important since the preprocessing heuristic presented in Section III is applicable only to location-based encodings. The following shows a chromosome, or a bipartition, with 100 genes. Belonging to the left side are vertices 1, 2, 3, 5, ... 95, 97 and belonging to the right side are those 0, 4, 6, ..., 94, 96, 98, 99.

C. Parents Selection, Crossover and Mutation

To select two parents, we use a proportional selection scheme where the probability for the best solution to be chosen is 4 times as high as the probability for the worst solution to be chosen. This is a very common selection technique in genetic algorithm design [17].

Crossover operators are used to create a new offspring chromosome by combining parts of the two parent chromosomes.

Although single-point crossover is the most common crossover operator in GA practices, we used 5-point crossover for our algorithm based on the results of some preliminary tests that we made. More discussion on multi-point crossover is provided in Section III. Fig. 1 shows our 5-point crossover scheme. Five-point crossover chooses 5 points at random on the chromosome, dividing it into 6 parts, and copies the contents of the two parents alternately to the offspring. Let this be Offspring 1. We use one more crossover operator which is the same as the above except that it copies the complement values of Parent 2 while it copies the values of Parent 1 unchanged. Let this be Offspring 2. GRCA selects the better of the two offsprings and passes it to the local optimizer. The reason for using two crossover operators is as follows. If two parents are exactly (or almost exactly) the complement of each other, they represent the same (or almost the same) bipartition. The first crossover will create a severe inconsistency in Offspring 1, consequently that chromosome has little chance of surviving.

After crossover, GRCA tries a mutation with a mutation probability of 0.015. That is, each gene value is complemented with probability 0.015.

D. Local Improvement

Due to space limitation, the discussion in this subsection assumes that the reader is familiar with the Fiduccia-Mattheyses algorithm (FM) of [16]. After crossover and mutation, GRCA applies a local improvement process on the offspring. We use a weak variation of FM. We intentionally use the term local improvement instead of local optimization since we don’t try to get local optimum points but try to get only a visible improvement. As described in [16] the time complexity of FM is $O(|E|)$. The standard FM has several passes, usually around 10. Each pass determines a set of modules to be moved. The algorithm stops when a pass, or two consecutive passes do not produce a better solution. The number of vertices to be moved in each pass can be as large as $|V| - 1$. In our variation we allow only one pass, furthermore, the number of vertices to be moved is restricted to be no more than Max-Moving-Size, a parameter that we have tried with several different values. If Max-Moving-Size is chosen to be large, e.g., $|V| - 1$, for a number of graphs the average solution sizes are rather high. A possible reason is that a strong local optimization technique causes GRCA to converge prematurely. On the other hand setting Max-Moving-Size to be very small, say 2 or 6, then the resulting solutions are very poor and very slow to converge. We set Max-Moving-Size equal to $|V|/6$ after an extensive test considering the tradeoff between performance and time. An empirical measure showed that this weakened version of FM takes around 1/20 of the time taken by a standard FM. Since this local improvement is the bottleneck of GRCA (all other operations are quite cheaper, $O(|V|)$ time) this fast local improver makes GRCA efficient. We will see in
Section V that GRCA generally takes less time than EIG1-IG [9] and about 9 times longer than Rcut2.0 [25, 26].

We also add one important change to the tie-breaking rule in FM. In maintaining the gain lists, we used cut-size gains instead of ratio-cut gain for efficiency. If we were to maintain a ratio-cut gain list we would have to recalculate every gain in every k-hop of FM, which would result in a running time of $O[V^2|J|]$. Wei and Cheng [25] provided a good explanation for this: "unless the size of a module is unusually large, it won't contribute much to the denominator of the ratio-cut value $\frac{\text{gain}(r)}{\text{gain}(r) + \text{cut}(r)}$ of module $i". So we consider the two modules with maximum cut-size gains in each side. If maximum gains of the two sides are equal, it is intuitively natural to give priority to the side which causes larger ratio-cut gain. When the sizes of the max-gain modules are comparable, the vertex in the larger partition will have the advantage. We observed that this tie-breaking rule tended to give very well-balanced partitions but it was not so good at finding considerably unbalanced partitions with good ratio cuts. We think this is the main motivation that Wei and Cheng added the left/right shifting step in their Rcut1.0 [25]. In our algorithm we introduce a token and the side having this token wins the tie break. At the start, the token is given to a side at random. In a situation where there are consecutive ties, unless the loser in the previous tie break wins this round, the winner keeps taking the token. Using this scheme we could have GRCA try more unbalanced partitions. We also found that this strategy was capable of getting good unbalanced cuts but had a hard time maintaining balanced solutions. We could often observe that when an extremely unbalanced partition with very small cut is produced in the early stage of GRCA, the population got rapidly dominated by that partition and consequently tended to stuck at the search space around that solution even when there were other balanced solutions with better ratio cuts. Finally, we remedied this by allowing only $|V|/10$ consecutive wins by tie breaks. But unlimited wins are still allowed by non-tie. With this adjustment GRCA showed that it could maintain both balanced and unbalanced solutions.

It may be interesting to see how much the genetic process contributes to the space search in addition to the FM's searching power. In [25], Rcut1.0 was reported to perform 36% better than FM. In our experiments, we found that Rcut2.0, an enhanced version of Rcut1.0. It should be also noted that GRCA was not comparable to existing algorithms when we removed this local optimization part.

E. Replacement Scheme and Stopping Criteria

After having generated a new offspring and locally improved it, GRCA decides whether to replace a member of the population with the new offspring. Inevitably, we saw that the quality of the solution depends greatly on the replacement scheme. It was observed that with a loose replacement scheme, e.g., always replace the worst member of the population with the new offspring, GRCA can converge very quickly at the expense of loosing population diversity, which causes performance degradation.

GRCA first compares the ratio-cut of the offspring and that of the more similar (in bit-wise difference) parent. If the offspring is better than the parent then it replaces the parent. If not, GRCA discards the offspring (we call this scheme the NEAR replacement scheme). At some point, GRCA adaptively changes its replacement scheme. It is still the same in that it compares the offspring with the similar parent. But when it fails to replace it, it now compares the offspring with the other parent. Only when the offspring is not better than the second parent, does GRCA discard it (we call this scheme the COMBI replacement scheme). The reason we use this adaptive replacement scheme is that the ratios of failed replacements are too high during the latter generations of GRCA using solely the replacement scheme NEAR (a few hundreds of consecutive fails were often observed). Consequently the NEAR scheme takes a very long time to converge. We say a swing occurs when an offspring is discarded at the end of a generation. The event that 7 consecutive swings occur is called an out. GRCA starts with the NEAR replacement scheme. It switches its replacement scheme to COMBI after two outs have occurred. It stops when two more outs occur. The number of swings, namely 7, for one out was chosen after considering the performance/time tradeoff. The more swings for an out, the better the results and the longer the running time. The reason we use two outs instead of one is to avoid accidental consecutive swings. A detailed account of various replacement schemes is given in [5].

III. Potential Performance Improvement By Preprocessing

A. Motivation

A schema represents a pattern of chromosomes. A schema on binary alphabets is defined as an $n$-tuple $\langle s_1, s_2, \ldots, s_n \rangle$ where $s_i \in \{0, 1, +\}$ for $i = 1, 2, \ldots, n$. In a schema, the symbols 0 and 1 are called specific symbols; ‘+’ specifies the don’t-care positions in the pattern. The defining length of a schema is defined as the length from the leftmost specific symbol to the rightmost specific symbol in that schema. By a $k^\text{th}$ order schema we mean a schema with $k$ specific symbols. A schema can also be viewed as a set of chromosomes containing the pattern specified by its specific symbols. With chromosomes of length 8 as examples, a $3^{rd}$ order schema $**110****$ represents the set of chromosomes with symbols 1, 1, and 0 at the third, fourth and sixth positions, respectively. Therefore, it contains chromosomes $10110000$, $00110000$, $11110111$, and many more.

When a single-point crossover is applied to two parents, some schemas in the parents survive and some do not. If the crossover point divides the specific symbols of a schema into two parts, only one part can be copied from one parent. Therefore, the survival probability of a schema is anti-proportional to the defining length when a single-point crossover is applied. This is one of the most important implications of the Fundamental Theorem of Genetic Algorithm [18]. But when we use multi-point crossovers, the survival probability of a schema does not necessarily depend on the defining length. This was first observed by De Jong [14] and later studied in more details by De Jong and Spears [15]. Their key point is that a schema is not disrupted when an even number (including 0) of crossover points fall between the two specific symbols of every pair of adjacent specific symbols. This is a motivation of our suggestion of a new schema; we treat specific symbols in group rather than by individual.

We observe that the survival probability of a schema is highly dependent on its inner structure. For instance, consider the two $8^{th}$ order schemas H1 and H2 with the same defining length of 28. Specific symbols are evenly distributed in Schema H1; they are highly clustered in Schema H2.

\begin{align*}
\text{H1:} & \quad \text{110****}\text{110****} \\
\text{H2:} & \quad \text{110****110****}
\end{align*}

If we use a single-point crossover, the survival probability through the crossover is $\frac{2}{28}$ for both H1 and H2. For simplicity, we don’t consider the case that the schema is recovered by chance with copied values from the other parent. If we use 2-point crossover, the survival probability of H1 is $\frac{2^0 + 2^0}{2^8} = \frac{57}{54}$; that of H2 is $\frac{2^0 + 2^0 + 2^2}{2^8} = \frac{207}{54}$. Clustered schema H2 has
nearly 4 times higher probability of survival. Defining lengths have little to do with survival probabilities in this context. The idea of our preprocessing is to transform apparently important schemas into clustered forms like H2.

B. New Schemas: C-Schemas

We introduce a new type of schema, clustered schema or c-schema. Define a qth order c-schema as \(D_0 C_1 D_1 C_2 D_2 \ldots C_q D_q\), where \(C_i \in \{0, 1\} [0, 1]\), \(i = 1, 2, \ldots, q\), and \(D_i \in \{0, 1\}\), \(i = 0, 1, \ldots, q\). Note that the definition of order here is different from the general definition of order as in previous subsections. Any ordinary \(r^{th}\) order schema can be represented by a \(q^{th}\) order c-schema such that \(q \leq r\). Define \(P_k(D_0 C_1 D_1 \ldots C_q D_q)\) as the probability that a c-schema \(D_0 C_1 D_1 \ldots C_q D_q\) survives through a \(k\)-point crossover. For the c-schema to survive, even number of crossover points should fall on each \(D_i\) for \(i = 1, 2, \ldots, q-1\) and furthermore no crossover point fall within any \(C_i\) for \(i = 1, 2, \ldots, q\). We have (see [8] for detail):

\[
P_k(D_0 C_1 D_1 \ldots C_q D_q) = \sum_{i_1, \ldots, i_{q+1} \in [k/2]} \left(\begin{array}{c}
\binom{2^{i_1+1}}{2i_1} \\ 2i_1+1
\end{array}\right) \ldots \left(\begin{array}{c}
\binom{2^{i_{q+1}+1}}{2i_{q+1}+1} \\ 2i_{q+1}+1
\end{array}\right) \left(\begin{array}{c}
\binom{k}{k}
\end{array}\right)
\]

C. Weighted-DFS Reordering (WDFR)

Every location on a chromosome has its own meaning in location-based encodings of genetic algorithms. In our case, each vertex has a corresponding location on a chromosome. Therefore the \(i^{th}\) value on a chromosome specifies which side the \(i^{th}\) module belongs to. The most natural indexing scheme is to use the indices given by the input file. We found that we could improve the performance of genetic algorithms by re-indexing the vertices [6] and here we extend the technique to hypergraphs.

Given hypergraph \(G\), we first convert \(G\) into a graph by the standard clique transformation [19], where each \(k\)-pin net is transformed into a complete subgraph on \(k\)-vertices, with each edge having weight equal to \(\frac{1}{n^k}\). To save computation time, we ignore nets with more than 20 pins. As a rationale, 20 pins in a single net are hard to be considered as closely related ones since most net sizes are less than 10. The heuristic does depth first search (DFS) on this weighted graph starting at a random initial vertex. The only difference from the traditional DFS is in the tie breaking. Here, when DFS encounters unvisited vertices adjacent to a vertex, it visits a vertex connected to the current vertex by the largest weight edge. We reindex the vertices by the visiting order of DFS. By this ordering clustered modules tend to form a better c-schema than a set of arbitrary modules with the same cardinality. It should be noted that this transformed graph is not used by the main genetic algorithm. Instead, the information is used only for location assignment of modules on chromosomes. This is a big difference from preprocessings for other standard partitioning algorithms where pre-processings are to provide good initial solutions for them [6] [4]. In GAs, the solutions in the initial population are randomly generated.

IV. COMBINATORIAL ARGUMENTS

A. Time Complexity

It is clear that the processes choose, crossover, mutation, suited and replace all take \(O(\sqrt{\mathcal{V}})\) time. FM is practically an \(O(\mathcal{E})\) time algorithm. Since we used a weak variation of FM for the local improvement, it takes \(O(\mathcal{E})\) time with a much smaller constant factor than that of the original FM. Therefore, the time complexity of GRCRA is \(O(K : \mathcal{E})\) where \(K\) is the number of generations. In our experiments, \(K\) ranges from 700 to 1600. Our local improvement takes on the average 1/20 of the time of a full FM. This is the main reason why GRCRA’s running time remains quite reasonable. The data for the running time of GRCRA will be shown in Section V.

One theoretical drawback of WDFR is that its worst case time complexity is \(O(\mathcal{V}^{1/3})\) for this to be \(\Theta(\mathcal{E})\) we need two restrictions: (i) the maximum degree of modules should be bounded by a constant, which is the case in most VLSI circuits, and (ii) the number of pins in a net should also be bounded by a constant, this is not the case in many circuits. We resolved this by ignoring nets with more than 20 points as far as WDFR is concerned. This causes little harm since this omission occurs only in WDFR. In practice, we observed that WDFR took no more than 2% of GRCRA’s total running time.

B. How WDFR Helps to Construct Good C-Schemas

Define an island to be a connected subgraph forming a cluster. The edges that have at least one end point in the island and at least one end point outside of the island are called bridges. Note that our usage of the term “bridge” is different from the standard usage in graph theory. Fig. 2 shows an island with 2 bridges. In the case of the graph bipartition problem, vertices in an island that have few bridges are more likely to belong to the same side than an arbitrary set of vertices. This implies that vertices in an island with few bridges are more likely to form a high-quality schema than an arbitrary set of vertices. Based on this observation, we now think about how a schema consisting of the vertices of an island is transformed into a probably better schema. We concentrate on the vertices in the island. Consider the schema consisting of vertices in an island, that is, only positions occupied by vertices in the island have specific symbols in that schema. The following proposition (see [9] for more details) is applied also to WDFR, which says that an island with a small number of bridges is likely to form a good c-schema.

Proposition. For an island with \(r\) bridges, vertices in the island form c-schemas of the form \(D_0 C_1 D_1 C_2 D_2 \ldots C_q D_q\) where \(q \leq r + 1\) if the vertices are located on the chromosomes by the visiting order of DFS.

Although the number of \(C_i\)’s is bounded by \(r + 1\), it may be much less than the possible maximum. For each bridge that is classified as a non-tree edge in the DFS, \(q\) decreases by 1. The case \(q \ll r\) implies a highly clustered schema. If \(D_i\) is very short, we can approximately treat \(C_i D_i C_{i+1}\) as just a \(C_i\). From the arguments in Section III and above proposition, it is apparent that the schemas corresponding to the clustered subgraphs may survive with higher probability through multipoint crossovers when we carefully rearrange the locations of vertices on chromosomes.

V. EXPERIMENTAL RESULTS

We tested GRCRA on 11 benchmark circuit graphs including 9 MCNC benchmarks. Table I shows the numbers of modules, nets, and I/O pads for each circuit. We followed the tradition
of [25] and [19] where the module areas are normalized so that the smallest non I/O pad module has unit size and all I/O pads have unit size to reduce the influence of I/O pads. We compare the results of GRCA against those of RCut2 0.25, 26 and EIGI-IG [19]. Both programs were obtained from their authors. Note that RCut2 0.25 is a recently enhanced version over RCut1.0 of [25]. EIGI-IG is the better version between the two versions in [19]. As GRCA takes approximately 9 times longer than RCut2 0.2 on the average, we compare the average result of GRCA against the best from 10 runs of RCut2 0.2 for each circuit. To reduce statistical error, we tried 100 runs of RCut2 0.2 and divided them into 10 groups each consisting of 10 runs. The bests of the groups are compared with the average of GRCA in Table II and the best of 100 runs of RCut2 0.2 is compared with the best of 10 runs of GRCA in Table III. For convenience we call 10 runs of RCut2 0.2 RCut2 0.2-10. EIGI-IG always generates the same results. Table II shows the average results of the three algorithms and Table III shows the best results. All timing data are in CPU seconds on a Sun SPARC Station 4/400. GRCA didn't always perform best among the three algorithms. In average results, GRCA performed best for 5 graphs among 11 graphs. Overall, GRCA's average results are 12.3% better than the averages of the best results of RCUT2 0.2-10 and 29.5% better than those of EIGC-IG. In best results, GRCA performed best for 10 graphs among 11 graphs. Overall, the best results of GRCA are 15.5% and 36.4% better than those of RCUT2 0.2-10 and EIGC-IG, respectively. Comparison of best results gives some disadvantage to EIGC-IG due to its deterministic output.

GRCA's running time showed the smallest growth rate among the three algorithms. For relatively small circuits, the running times of EIGC-IG and/or RCUT2 0.2-10 are comparable to that of GRCA. As the sizes of the circuits increase, GRCA's running time became better than the other algorithms. For the smallest circuit PrimG1, all three algorithms showed comparable time. But for the largest circuit, Industry2, GRCA's time was 5% of the RCUT2 0.2-10's time and only 14% of EIGC-IG's time.

Preprocessing didn't always show improvement. Since the rationale of preprocessing is based on the clustered subgraphs, it is thought to be dependent on the clustering pattern of circuits. We denote preprocessing GRCA by PGRCA. In Table IV we see that 5 circuits among 11 showed improvement. For some graphs, WDFR in fact caused harm. From the results, we suggest that preprocessing be used optionally. It is worth noting that in almost every case when preprocessing improves the quality of the solution or does not degrade it very much, then it usually reduces the running time of the algorithm.

VI. CONCLUSIONS

We presented a hybrid genetic algorithm for the ratio-cut partitioning problem. The algorithm uses a carefully designed variation of the Fiduccia-Mattheyses (FM) algorithm and combines it with genetic space exploration. As mentioned in Section II D, the performance of a single FM is poor and the performance of GRCA without any local optimization was also poor. Used as a preprocessing stage, however, the complete algorithm is competitive.

The preprocessing heuristic is computationally cheap as mentioned in Section IV A. The preprocessing for GA works quite differently from the preprocessing for other heuristics like FM and KL. In those cases, preprocessing heuristics are used to obtain good initial solutions. In our case, the preprocessing has nothing to do with the initial solutions (of the initial population), instead it indirectly helps GAS effectively explore the search space. As mentioned in the experimental report, WDFR has not always proved beneficial. The reason is that the WDFR heuristic is as good as that of other reordering heuristics, and the search space is almost the same.

Our variation of FM local optimization enables GRCA's running time to be much closer to that of the other algorithms. An examination of the data show that it still occupies on the average 71% of GRCA's running time. It would be nice to see how various types of local optimizations affect GRCA in performance and time.

ACKNOWLEDGMENT

We thank Dr. Chung-Kuan Cheng and Lung-Tien Lin for providing us with the RCUT2 0.2 source and the MCNC benchmark data, and Dr. Andrew Kahng and Lars Hagen for the EIGC-IG source. We also wish to thank Dr. Jason Cong and Dr. M'Lissa Smith for providing us with some industry circuits.

REFERENCES


### Table III: The Best Results of the Three Algorithms

<table>
<thead>
<tr>
<th>Circuits</th>
<th>EIG-3G Areas</th>
<th>N-Cuta</th>
<th>R-Cutb</th>
<th>RCut2-0.10 Areas</th>
<th>N-Cuta</th>
<th>R-Cutb</th>
<th>GRCA Areas</th>
<th>N-Cut</th>
<th>R-Cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimGA1</td>
<td>757.2675</td>
<td>14</td>
<td>691</td>
<td>502.2929</td>
<td>11</td>
<td>748</td>
<td>757.2674</td>
<td>14</td>
<td>691</td>
</tr>
<tr>
<td>PrimSCH1</td>
<td>594.2160</td>
<td>14</td>
<td>1022</td>
<td>530.2243</td>
<td>200</td>
<td>1058</td>
<td>594.2160</td>
<td>14</td>
<td>1022</td>
</tr>
<tr>
<td>PrimGA2</td>
<td>2518.5835</td>
<td>87</td>
<td>590</td>
<td>2534.8938</td>
<td>77</td>
<td>520</td>
<td>2518.5835</td>
<td>87</td>
<td>590</td>
</tr>
<tr>
<td>Test02</td>
<td>3357.5349</td>
<td>87</td>
<td>690</td>
<td>3372.3739</td>
<td>77</td>
<td>615</td>
<td>3357.5349</td>
<td>87</td>
<td>690</td>
</tr>
<tr>
<td>Test03</td>
<td>4610.53640</td>
<td>102</td>
<td>131</td>
<td>10917.37133</td>
<td>42</td>
<td>5.68</td>
<td>4610.53640</td>
<td>102</td>
<td>131</td>
</tr>
<tr>
<td>Test04</td>
<td>8402.13827</td>
<td>65</td>
<td>55.9</td>
<td>6397.35832</td>
<td>42</td>
<td>41.5</td>
<td>8402.13827</td>
<td>65</td>
<td>55.9</td>
</tr>
<tr>
<td>Test05</td>
<td>5431.41408</td>
<td>6</td>
<td>26.7</td>
<td>20972.21062</td>
<td>44</td>
<td>9.96</td>
<td>5431.41408</td>
<td>6</td>
<td>26.7</td>
</tr>
<tr>
<td>Test06</td>
<td>12616.16842</td>
<td>9</td>
<td>424</td>
<td>8384.8884</td>
<td>39</td>
<td>83.4</td>
<td>12616.16842</td>
<td>9</td>
<td>424</td>
</tr>
<tr>
<td>Industry2</td>
<td>102512.21015</td>
<td>115</td>
<td>53.4</td>
<td>662924.2636</td>
<td>206</td>
<td>126</td>
<td>102512.21015</td>
<td>115</td>
<td>53.4</td>
</tr>
</tbody>
</table>

* N-Cut is the number of nets cut by the bipartition.
* Ratio-cut value is R-Cut/(N-Cut+R-Cut).

### Table IV: The Effect of Preprocessing by WDFR

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Best</th>
<th>Average</th>
<th>CPU</th>
<th>Best</th>
<th>Average</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimGA1</td>
<td>757.2674</td>
<td>14</td>
<td>24.31</td>
<td>757.2674</td>
<td>14</td>
<td>24.31</td>
</tr>
<tr>
<td>PrimSCH1</td>
<td>593.2159</td>
<td>14</td>
<td>25.89</td>
<td>593.2159</td>
<td>14</td>
<td>25.89</td>
</tr>
<tr>
<td>PrimGA2</td>
<td>2534.8938</td>
<td>77</td>
<td>163.4</td>
<td>2534.8938</td>
<td>77</td>
<td>163.4</td>
</tr>
<tr>
<td>Test02</td>
<td>19018.37132</td>
<td>42</td>
<td>65.10</td>
<td>19018.37132</td>
<td>42</td>
<td>65.10</td>
</tr>
<tr>
<td>Test03</td>
<td>658815641</td>
<td>39</td>
<td>73.80</td>
<td>658815641</td>
<td>39</td>
<td>73.80</td>
</tr>
<tr>
<td>Test04</td>
<td>1984022159</td>
<td>42</td>
<td>63.63</td>
<td>1984022159</td>
<td>42</td>
<td>63.63</td>
</tr>
<tr>
<td>Test05</td>
<td>3470837186</td>
<td>42</td>
<td>102.1</td>
<td>3470837186</td>
<td>42</td>
<td>102.1</td>
</tr>
<tr>
<td>Test06</td>
<td>83978571</td>
<td>63</td>
<td>56.88</td>
<td>83978571</td>
<td>63</td>
<td>56.88</td>
</tr>
<tr>
<td>19ks</td>
<td>10159987</td>
<td>103</td>
<td>48.90</td>
<td>352174800</td>
<td>103</td>
<td>48.90</td>
</tr>
<tr>
<td>Industry2</td>
<td>102512.21015</td>
<td>103</td>
<td>78.53</td>
<td>102512.21015</td>
<td>103</td>
<td>78.53</td>
</tr>
</tbody>
</table>

* N-Cut is the number of nets cut by the bipartition.
* Ratio-cut value is R-Cut/(N-Cut+R-Cut).
* Percentage below those of GRCA. A positive number means PGRCA has a better performance.

Assignment and Related problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, in press.


---

* N-Cut is the number of nets cut by the bipartition.
* Ratio-cut value is R-Cut/(N-Cut+R-Cut).
* Percentage below those of GRCA. A positive number means PGRCA has a better performance.