Genetic Algorithms for Graph Partitioning

EECE 583: Algorithms for CAD

Ken Mansfield
University of British Columbia
Department of Electrical and Computer Engineering
kmansfield@ece.ubc.ca

I. Introduction

I have implemented various methods of performing graph partitioning using Genetic Algorithms and I will contrast and compare them in this paper. This project is inspired by the work done by Bui and Moon [1] and Nan et al. [2].

II. Literature Review of Related Work

A graph partition problem is defined by an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges such that it is possible to partition G into smaller sub graphs. The cut size of a partition is defined to be the number of edges whose adjacent vertices are located in different partitions. A graph that is split into two partitions is known as a graph bi-partition.

In the VLSI domain, circuits are often bigger than a single chip and need to be partitioned amongst chips. Additionally, circuits frequently need to be divided among different portions of a chip. Other areas where graph partitioning is used includes matrix factorization, load balancing, and community detection. Finding an exhaustive optimal solution is known to be NP-Complete, so faster heuristic algorithms are used.

The most popular solutions have traditionally been the Kernighan-Lin (KL) [5] algorithm and the simulated annealing algorithm [8]. The KL algorithm is initialized with a balanced partition of vertices and computes several passes. In each pass, the algorithm improves upon the initial solution by swapping vertices to create a new solution. The KL algorithm has proven to be very time efficient and produces excellent results, better than simulated annealing, thus many new techniques are based on KL such as Fiduccia-Mattheyses [6]. Genetic algorithms have also been used to solve the graph partitioning problem [1,2,3,4]. Some methods are compared and contrasted below.

A. *Bui and Moon: Genetic Algorithm and Graph Partitioning*

Bui and Moon [1] devised an approach that uses a hybrid Genetic Algorithm to solve graph partitioning. A genetic algorithm starts with an initial set of solutions known as a population. Each set of solutions, known as chromosomes, are combined to allow the population evolve after many iterations. When the algorithm is finished, the best solution in the population is returned as the final solution. In each iteration, also known as a generation, two members of the population are picked based upon a probability distribution. The two members are then combined through a crossover operation that produces a new solution called the offspring. Next a mutation is performed on the offspring to make a slight modification. This enhances the diversity of the population as well as allowing for stochastic exploration of the search space. The algorithm then decides which member of the population should be replaced by the offspring, if it is suitable. This process is then repeated many times until a suitable solution has been found. This is known as a steady-state genetic algorithm because the population does not increase and all but one of the previous chromosomes remain the same. In the case that most or all chromosomes are removed in a single step, then it is known as a generational genetic algorithm.

To apply a genetic algorithm to the graph partitioning problem, the chromosome can be defined by a sequence of gene values of {0,1}. For a bi-partitioned graph there would be an equal number of 0's and 1's in the chromosomes.

Bui and Moon [1] implement a hybrid genetic algorithm by augmenting the chromosome evolution with a local improvement heuristic. They start by initializing a population of 50 chromosomes with random values such that there are an equal number of 0's and 1's. For selecting the parents, they have implemented their fitness function as $F_i = (C_{\underline{W}} - C_i) + (C_W - C_b)/3$, where C_W is the worst observed chromosome and C_b is the best, and C_i is the cut size of solution i. The parents are then selected with a probability proportional to F_i. This allows solutions with a better cut size to be much more likely to be selected, but still allows other solutions to be selected with a lower probability.

Bui and Moon claim that genetic algorithms are not so good at fine tuning around local optima so they augment their method with a variation of the KL algorithm. They apply KL to the child chromosome that is generated on every iteration. The result of this is that the algorithm converges in far fewer iterations than would be achievable by GA alone.

B. *Nan: Two Novel Encoding Strategies Based Genetic Algorithms for Circuit Partitioning*

Nan et al [2] have chosen a slightly different approach. Their fitness function reduces the fitness value of individuals that lead to unbalanced partitioning. They construct a mating pool by selecting chromosomes using the roulette wheel strategy based on their fitness, and then they randomly select parents from this mating pool. Crossover is then performed to generate offspring. They state that single point crossover is not good for the reservation and recombination of long distance schema so they choose a two-point crossover. The offspring produced, as well as a few of the best chromosomes are then used to create a new generation. This differs from [1] where they produce a single offspring that replaces a single parent each generation. Mutation is performed by simply picking cells with a random probability.

They also explore encoding the chromosomes with integers as well as binary. During the crossover using integer encoding, a certain number of illegal moves may be made. Some genes may end up on both sides of the partition which is not allowed, also a gene may be duplicated within the same block. To fix this, they propose partial mapped crossover by choosing a random cut point and swapping cells in the first parent such that the second half matches the second half of the second parent. In this manner the first half of the child is not exactly the same as either parent (but does have some similarity to the first parent), but the second half is exactly like that of the second parent. Their evaluations show that binary encoding performs better than integer encoding.

C. *Stenbeek: Finding Balanced Graph Bi-Partitions Using Hybrid Genetic Algorithm*

[3] took an approach similar to Bui and Moon in which they used a hybrid genetic algorithm using a heuristic for local optimization. Their approach differs by the way they have implemented their local optimization, as well as their genetic algorithm. They designed a heuristic which identifies clusters, which are subgraphs with a density that is higher than the density of the entire graph. In their preprocessing stage, if they detect clusters, they implement a cluster emplacement heuristic. They swap clusters between the sides of the partitions to achieve a more balanced partition. After preprocessing, their technique is similar to Bui and Moon where they apply a genetic algorithm with KL being applied to the child on each iteration. Their crossover function applies the Uniform Crossover method, which uses a fixed mixing ratio (usually 50%) between the two parents but randomly chooses the crossover points.

D. *Chardaire: A PROBE Based Heuristic for Graph Partitioning*

Chardaire et al. [4] implemented a technique, named PROBE (for Population Reinforced Optimization Based Exploration) which is population based, like GA, but is simpler because it does not include selection, mutation, and crossover. PROBE uses a population to determine subspaces that are explored to find optimized solutions, which then form a new population. The process is then repeated. PROBE takes advantage of the characteristic of the crossover operator in GAs that if two parents share a common bit, their offspring will inherit it. At each generation PROBE uses this characteristic to generate feasible offspring from feasible parents.

They then construct solutions using the Differential-Greedy algorithm proposed by Battiti and Bertossi [7] from partial bi-sections. The DG algorithm starts by running Battiti and Bertossi's Min_Max greedy construction algorithm that builds a bisection by alternatingly adding one vertex to each partition in a greedy manner. This bisection is then fed into their Fixed Tabu Search (FIXED-TS) algorithm which improves upon it by a scheme of local search and prohibition based on KL. FIXED-TS, like KL, moves a cell to the other side of the partition, however it only prohibits it from moving for a certain number of iterations.

With the solutions generated from the previous step they use the local optimization step from the Bui and Moon [1] paper. Then the process starts over again.

E. *Comparison and Thoughts*

Four different approaches to the graph partitioning problem have been described that use variations of

Genetic Algorithms. [2] is the only one that attempts to solve the problem using GA alone, the other three use hybrid approaches that perform a local improvement heuristic on the offspring during each iteration. [4] uses a modified population based algorithm that can be thought of as a simplified genetic algorithm.

The roulette method, whereby parents who have a better cut size have a higher probability of being chosen, appears to be the most popular method for parent selection. [1] and [2] use fitness functions which allow the most fit chromosome to have significantly higher probability of being chosen than the worst chromosome. [1] and [2] both implement roulette selection afterwards. [3] uses a different method which uses the value returned from their cluster enhancement algorithm as their fitness function. They also use a different selection method as well. They implement tournament selection where they select 2 chromosomes at random, and select the better one as the first parent, then they repeat to find the second method.

For crossover, [1] and [2] use 2 point crossover, where the crossover points are randomly chosen. Both [1] and [2] claim that multi-point crossover may destroy good schemas however [3] performs multi-point crossover by using uniform crossover as described above.

[1][2][3] all choose to randomly mutate the children with a set probability. [2] also implements an integer encoded technique which does its crossover and mutation completely differently, as explained above.

[1][3] and [4] each perform a local improvement heuristic at each step. They each perform a KL based heuristic step that is based on the strategy first described by [1]. They describe their motivation in applying KL is to find local minima, as well as allowing the algorithm to converge quicker. [3] goes further to apply a cluster enhancement algorithm as a preprocessing step, as well as during each iteration. [4] performs an additional, differential-greedy step [5], before performing the local improvement from [1].

[1] and [2] compare their results directly against KL/FM and claim that their results are superior, however they are both computationally more expensive. [2] takes 50x as long to converge as FM. [1] only compares their results against multi-start KL so the time comparison is difficult to measure. [3] compares their results with [1] with comparable final cut sizes as well as comparable performance. [4] compares their results directly to [1] with the conclusion that their results are comparable, however, computationally much more expensive.

III. Implementation

The Bui and Moon paper describes a Genetic Algorithm which creates a new child every iteration and performs local improvements on that child. The first thing I want to test is how well the Bui algorithm will work without the local improvement.

A. *Genetic Algortihm (non-generational)\*

1. Initialize 50 balanced, but random chromosomes.
2. Calculate the costs of each chromosome.
3. Calculate the fitness of each chromosome.
4. Perform Roulette selection to pick two parents.
5. Apply the crossover operator to both parents.
6. Apply mutation to the child.
7. Perform replacement.
8. Go to step 2.

B. *Genetic Algortihm With Local Improvement*

The algorithm works exactly the same as above except that a local improvement step is performed between step 6 and 7.

C. *Generational Genetic Algorithm*

1. Initialize 50 balanced, but random chromosomes.
2. Calculate the costs of each chromosome.
3. Calculate the fitness of each chromosome.
4. Perform roulette selection to create a mating pool of 25 possible parents.
5. From the new mating pool, randomly choose 2 parents.
6. Perform crossover to generate a child.
7. Perform mutation on the child.
8. If the child has a lower cost than the parents, add to the new generation. If the child has a higher cost then the parents add to the new generation 30% of the time, otherwise add the best parent to the next generation.
9. Do steps 6 to 8 until 50 chromosomes have been generated.
10. Go to step 2.

D. *Generational with Local Improvement*

Same as above, except that KL is performed on the child generated after step 7.

E. *Testing*

I had the benefit of being able to compare the results against the results from the K&L bi-partitioning algorithm that I wrote for the assignment from class. The main approach to ensuring that the algorithm is being written correctly is to make sure that results converge and are comparable to the results for K&L. It was quickly observed that improperly written cost functions, or selection operators resulted in diverging results or extremely poor results.

Visualizing the circuit is also useful for debugging as it allows us to ensure that the circuit is still balanced and still contains the correct number of cells.

Some attempt at more formal unit testing was made, however, due to time constraints and constant modifications these tests were not completely comprehensive for the final product. For example, a unit test was initially created for the crossover operator when a single non-random crossover point was chosen. However, when the code was modified to allow multiple random crossover points, this unit test was no longer valid.

IV. Results – Non Generational

Tuning the genetic algorithms proved to be a monumental task. For example, the Bui fitness function makes the best chromosome 4x as likely to be chosen as the worst. Changing this to around 8x as likely improved the results slightly, but making this 16x made the results worse. As well as changing this to a lower amount, ie 2x.

A. *Population*

Modifying the population had some affect as well. The generational algorithm responded well to 50 chromosomes, but the non-generational algorithm sometimes responded very well to smaller populations.

B. *Crossover*

The crossover operator was designed to be tunable for different numbers of crossover points, as well as different levels of randomness for picking the crossover points. The algorithms performed fairly similarly with either single point crossover or 2 point crossover, but performed poorly with 3 or more crossover points. They also performed poorly when the amount of randomness for the range of the crossover points was large.

The crossover operator also employs normalization, where offspring are generated using the second parent in both with all values complemented as well as un-complemented. Different options were analyzed to determine which offspring was returned:

1. Return the child with the best cut size.
2. Return the most balanced child.
3. Return the child that is most similar to parent1 based on hamming distance.

Ultimately, option 3 gave the best results, however, all 3 options returned respectable results.

C. *Replacement*

For replacement, various choices were explored:

1. Replace the worst in the population.
2. Replace the worst parent, else replace the worst in the population.
3. Replace the closest parent (by hamming distance) if better, else replace the worst in the population.

Option 1 caused premature convergence resulting in a terrible final cut size. In this situation, given the purpose of the fitness function to select the best parents with a higher probability, the population quickly gets replaced with the schemas of a small selection of the best parents. To prevent this from happening there has to be some way of only replacing chromosomes that are very similar. Since the offspring are naturally similar to their parents, it makes the most sense to replace one of the parents, if suitable. Another consideration is that we wish to avoid replacing the best chromosomes in a population, so we try to only replace a member in the population if the child is better. Options 2 and 3 both performed much better than option 1.

D. *Mutation*

Mutation allows the algorithm to get out local-minima, however, too much mutation prevents the algorithm from inheriting the best schema from the population. The mutation methods from both the Bui and Nan paper were tried, however, at the end I found that less mutation generally presented the best results. In particular, less mutation presented a much faster convergence for the earlier steps. It would be worthwhile to try a more dynamic approach whereby the amount of mutation stays low when the amount of convergence is high, and the amount of mutation is increased once convergence has slowed down or stopped and is stuck at a local minimum.

E. *Final Cut Size*

The final results are respectable, however, the final cut sizes are larger than the cut sizes for the optimized K&L algorithm (approximately 44% larger).

V. Experimental Setup

Developed in C++ using gcc.

Usage: ./project.exe [filename] [display mode] [generational] [local improvement]

Display mode: 0 = normal, 1 = no lines, 2 = no graphics

Generational: 0 = non-generational, 1 = generational

Local Imrpovement: 0 = no, 1 = yes.

Eg. ./project.exe apex 4 0 1

Makefile and test script (testAllFiles.sh) provided.

VI. Results – Generational

To save time, many of the operations share the same code from the non-generational algorithm, in particular fitness, crossover, and mutation. No replacement is performed in generational algorithms because the entire population is replaced in each generation.

A. *Mating Pool*

The standard size for mating pools is 50% of the size of the entire population. A larger mating pool was tried, however, this resulted in slower convergence and an inferior result.

B. *Final Cut Size*

The final cut sizes for the generational algorithm were 46% larger than K&L, and slightly larger than the non-generational algorithm.

				no KL	**With KL**	**no KL**	**With KL**
		Initial Random Placement	**K&L**	**non-generational**	**non-Generational**	**Generational**	**Generational**
	alu2	126	40	45	24	44	24
	apex1	497	166	195	111	198	114
	apex4	800	212	284	141	243	147
	c880	153	39	57	28	58	28
	cm138a	12	4	4	4	4	4
	cm150a	26	6	6	6	7	6
	cm151a	14	5	4	4	4	4
	cm162a	22	6	6	5	5	5
	cps	468	130	145	101	168	105
	e64	242	72	78	47	72	56
	paira	521	5	167	1	199	2
	pairb	521	5	167	1	183	2
Average		**283.5**	**57.5**	**96.5**	**39.4166**	**98.75**	**41.4166**
Average excluding pairb		**261.91**	**62.27**	**90.09**	**42.91**	**91.09**	**45.00**
% change vs KL (excluding pairb)				**144.67%**	**68.91%**	**146.28%**	**72.26%**

Table 1: Results

VII. Results – Local Improvement

The Bui paper performs K&L at each iteration as a local improvement step. The belief is that genetic algorithms are poor at fine tuning around local optima. Performing K&L also has the benefit of causing the hybridized genetic algorithm to converge much faster.

A. *Final Cut Size*

The local improvement steps resulted in superior cut sizes for both genetic algorithms. The non-generational algorithm and generational algorithms with local improvement were 41% and 38% smaller than K&L alone. They were half the size of their respective algorithms without local improvement.

VIII. Results – Speed

Genetic algorithms are known to be slow. The Fiduccia-Mattheyses variant of K&L is known to run in O(n). Genetic algorithms on the other hand, perform better when they are given ample time to converge. GA's also use random mutations, which may occasionally take suboptimal moves, to allow for greater exploration to pull itself out of local minima. This will also increase the time to converge. In particular, I observed that the algorithm could be stuck at a local minimum for hundreds of iterations and then suddenly pull itself out and continue to converge. This made it hard to choose a terminal condition for ending the algorithm as there could always be hope of further convergence due to the mutation. K&L on the other hand, will not converge any further if it is observed that the same result is produced twice in a row.

Some effort was made to ensure that the speed of each iteration was as efficient as possible by avoiding redundant operations. There is some room for improvement by using better data structures, however, these were not explored due to time constraints. For example, the chromosomes were stored in STL vectors. A bitfield could be used instead, which would have the benefit of being able to encode 64 genes in a single uint64. The crossover operator, for example, would be able to cross 64 bits in a single operation, whereas using a vector requires 64 separate operations.

IX. Conclusion

My experimental results have shown that local improvement does indeed help reduce the final cut size. The best results were obtained by using the algorithm that most closely resembles Bui and Moon's paper [1].

It is likely that the cut sizes produced by the algorithms without a Kernighan-Lin intermediate step could be improved upon, but the gains would likely not be enough to compete. There are a wide number of parameters that could be tuned for the GA, each of which affect the final results. Deciding which set of parameters produce the best results is a challenging task and could be thought of as an optimization task in its own right. The hybridized GA's, on the other hand, were not greatly affected by tuning the GA parameters. It appears that KL has the benefit of repairing damaged schema, it also has the benefit of ensuring that cut sizes only get better and not worse, unlike the crossover/mutation operation.

One of the problems with GA for graph partitioning is that good schema in a chromosome may be located on distant parts of the chromosome, and thus will be separated by the crossover points. Bui and Moon and Stenbeek et al. attempt to address these concerns with breadth-first search, and cluster pre-processing. It would be beneficial to see if these pre-processing techniques would have a positive effect on non-hybridized GAs as well.

References

[1] T.N. Bui and B.R. Moon, "Genetic Algorithm and Graph Partitioning," IEEE Transactions on Computers, vol 45, pp. 841-855, 1996.

[2] G.F. Nan, M.Q. Li, J.S. Kou, "Two Novel Encoding Strategies Based Genetic Algorithms for Circuit Partitioning," Proceedings of the Third International Conference on Machine Learning and Cybernetics, pp. 2182-2188, 2004

[3] A. G. Steenbeek, E. Marchiori and A. E. Eiben, "Finding balanced graph bi-partitions using a hybrid genetic algorithm", Proc. IEEE ICEC, pp. 90-95, 1998

[4] P. Chardaire, M. Barake and G.P. McKeown, "A PROBE-Based Heuristic for Graph Partitioning", IEEE Transactions on Computers, pp. 1707 – 1720, vol. 56, Issue: 12, 2007

[5] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", Bell Systems Technical Journal, vol. 49, pp. 291-307, 1970

[6] C. Fiduccia and R. Mattheyses, "A linear-time heuristics for improving network partitions", Proc. 19th Des. Autom. Conf., pp. 171-185, 1982

[7] R. Battiti and A. Bertossi, "Greedy and prohibition-based heuristics for graph partitioning", IEEE Trans. Comput., vol. 48, pp. 361-385, 1999

[8] D.S. Johnson, C. Aragon, l. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation", Par 1, Graph. Partitioning. Operations Research, vol. 37, pp. 865-892, 1989.