Combining Two Local Search Approaches to Hypergraph Partitioning

Arathi Ramani and Igor Markov
Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor 48109-2122
{ramania,imarkov}@eecs.umich.edu

Abstract

We study leading-edge local search heuristics for balanced hypergraph partitioning and Boolean satisfiability, intending the generalization of such heuristics beyond their original domains. We adapt the Fiduccia Mattheyses (FM) hypergraph partitioning heuristic to Boolean Satisfiability (SAT), and the WalkSAT SAT solver to hypergraph partitioning. Focusing on balanced hypergraph partitioning, we propose a combination of the classical FM heuristic and our "cross-over" heuristic WalkPart, and empirically show that it is more powerful than each component alone. Empirically, we show a 15% improvement in net cut and a 7% improvement in runtime over a leading-edge implementation of the FM heuristic.

1 Introduction

The focus of our work is on search heuristics for balanced min-cut hypergraph partitioning, a well-known combinatorial optimization problem. In general terms, the problem seeks to assign every vertex of a hypergraph to one of two subsets to minimize the number of connections between the two subsets, subject to having approximately the same number of vertices in each subset. Hypergraph partitioning is important in circuit layout because circuits are often represented by hypergraphs, e.g. in placement by recursive bisection. Many efficient hypergraph partitioning algorithms have been developed [3, 4]. Here, we discuss the combination of two local search heuristics: the Fiduccia-Mattheyses (FM) hypergraph partitioning heuristic [3], and the biased random walk heuristic, adapted from the WalkSAT SAT solver [5].

Our work is motivated by attempts to generalize well-known local search procedures [5, 3] beyond their original domains. We developed a SAT solver based on the FM algorithm [3], and a hypergraph partitioner, WalkPart, based on the WalkSAT algorithm from [5]. The WalkPart algorithm is described in Section 3. A detailed description of the SAT solver is beyond the scope of this paper. Our results show that WalkPart outperforms a highly tuned implementation of FM on several circuit benchmarks. The algorithms also produce very different solutions and appear to have different local minima, suggesting that a hybrid strategy might be effective. Empirical results show that the hybrid approach outperforms both algorithms, in terms of runtime and solution quality. Our main contributions are summarized as follows.

- We describe the balanced min-cut hypergraph partitioning algorithm, WalkPart, based on the WalkSAT satisfiability solver [5]
- We propose the combination of the FM and WalkPart heuristics to form a "hybrid" heuristic, and perform two experiments
- Empirically, we show that a hybrid strategy achieves significantly better performance than either heuristic alone

2 Background and Previous Work

2.1 Hypergraph Partitioning Problem:

A hypergraph $H(V,E)$ is defined as a set, V, of vertices, and a set, E, of hyperedges, where each edge $e \in E$ is a set of *two or more* vertices from V. The $k-$way hypergraph partitioning problem is stated as follows: Let $H(V,E)$ be a hypergraph with weighted vertices and edges. The problem asks to find a minimum cost partition P^k of the vertices of H to k disjoint subsets. The cost function $c(P^k)$ typically used is *edge cut*, the sum of the weights of the hyperedges *cut* by the partition (this is called min-cut partitioning). A hyperedge is cut exactly when all of its vertices are in more than one subset. Constraints may be imposed on a partition e.g. limiting the total vertex weight in each subset (*balance constraints*). The balance-constrained weighted hypergraph partitioning problem is NP-hard. The algorithms in this work deal with the balanced min-cut hypergraph bi-partitioning problem, i.e., we consider only the case where $k = 2$. In circuit layout, hypergraphs represent circuits by mapping vertices to signal nodes, and hyperedges to signal nets. In this context, we use the words "node" and "net" to mean "vertex" and "hyperedge" when referring to hypergraphs in this document.

2.2 The Fiduccia-Mattheyses Heuristic

The Fiduccia-Mattheyses (FM) [3] heuristic for bi-partitioning circuit hypergraphs is an iterative improvement algorithm. FM starts with an arbitrary initial solution and changes the solution by a sequence of moves, organized as passes. At the beginning of a pass, all vertices are free to move (unlocked). Each possible move is labeled with the immediate change in total cost it would cause (called "gain"). Iteratively, a move with highest gain is selected and executed, which may result in a lower, higher or unchanged solution cost[1]. The moved vertex is locked, i.e., not allowed to move again during that pass. Since moving a vertex can change gains of adjacent vertices, all affected gains are updated. Selection and execution of a best-gain move, followed by gain update, are repeated until every vertex is locked. The best solution seen during the pass is adopted as the starting solution of the next pass. The algorithm terminates when a pass fails to improve solution quality.

The FM heuristic is now used as a component in multi-level partitioning algorithms [4] which create a series of clustered hypergraphs, partition the most clustered with "flat" FM and then refine unclustered versions of the produced partition by applying FM in incremental mode. A highly optimized, leading edge implementation of FM in this context is available in [2]. This work only deals with "flat" partitioning heuristics in the hope that improved flat heuristics will lead to improved multi-level heuristics. This assumption will be verified in our future work.

[1] In a problem with balance constraints, moves should also not violate the legality constraint

Table 1: **Results for 10 passes of a Hybrid Algorithm: FM is run once to completion, followed by 100K moves of WalkPart alternated with single passes of FM.**

Benchmark Name	Nodes	Nets	WalkPart			FMPart		Hybrid		Cut Ratio	Time Ratio
			Moves	Time	Cut	Time	Cut	Time	Cut		
ibm01+10	12752	14111	2M	9.16	610.67	1.54	477.00	2.0	**284.00**	0.59	1.29
ibm01+4	12752	14111	500K	2.56	506.00	1.39	464.33	2.667	**413.33**	0.89	1.91
ibm02+10	19601	19584	500K	3.61	329.67	2.15	402.00	3.8	**290.67**	0.72	1.76
ibm02+4	19601	19584	500K	3.61	654.33	2.14	**386.00**	3.7	523.33	1.35	1.72
ibm03+10	23136	27401	2M	14.15	1603.33	3.84	1487.67	5.0	**1157.33**	0.77	1.30
ibm03+4	23136	27401	500K	3.84	1175.00	3.14	2488.00	5.86	**1093.33**	0.43	1.86
ibm04+10	27507	31970	500K	4.50	1589.00	5.30	718.67	9.0	**606.66**	0.84	1.69
ibm04+4	27507	31970	2M	14.94	2121.00	3.52	2505.00	5.0	**1089.33**	0.43	1.41
ibm05+10	29347	28446	2M	17.34	4804.67	9.54	2793.33	14.75	**1869.66**	0.66	1.54
ibm06+10	32498	34826	2M	12.73	**750.00**	5.63	1683.00	10.5	1117.00	0.66	1.86
ibm07+10	45926	48117	2M	18.07	2483.33	9.63	1347.33	13.98	**1190.67**	0.88	1.45
ibm08+10	51309	50513	2M	24.92	1902.67	11.76	3076.00	25.5	**1574.66**	0.51	2.16
ibm09+10	53395	60902	2M	19.60	2069.00	14.38	1936.00	29.0	**1365.33**	0.70	2.01
ibm10+10	69429	75196	2M	19.79	3414.67	15.36	2200.33	30.22	**1440.33**	0.65	1.96
ibm11+10	70558	81454	2M	21.61	**2433.33**	19.78	4203.33	34.13	2992.00	0.71	1.72
ibm12+10	71076	77240	2M	22.99	**3803.67**	23.03	4200.67	36	4525.33	1.07	1.56
Average										0.7	1.7

2.3 The WalkSAT Satisfiability Solver

The WalkSAT algorithm [5] is a stochastic local search heuristic for Boolean Satisfiability. WalkSAT is an improvement on GSAT (Selman et al., 1992), a greedy algorithm that starts with a random assignment and flips variables one at a time. The variable chosen for flipping is *always* the one that would cause the greatest immediate reduction in the number of violated clauses. WalkSAT [5] improves on GSAT by using a biased random walk to avoid getting stuck in local minima.

To select the next variable to flip, WalkSAT picks a clause at random from the unsatisfied clauses. With probability p, it flips a variable picked randomly from this clause. With probability $1-p$ it flips the variable that would cause the *least number of clauses* to be violated when flipped.

3 The WalkPart Algorithm

The WalkPart algorithm is a stochastic local search algorithm for balance-constrained hypergraph bisection. It performs a sequence of moves, until a preset maximum number of moves is reached. A "move" moves a vertex from one partition to another, if balance constraints are not violated. Moves are chosen using criteria similar to WalkSAT: with probability p, a vertex is picked at random, and with probability $1-p$, a vertex is chosen to minimize some cost function. The selected vertex is then moved to the other partition (if balance constraints allow). After a move, the costs of vertices adjacent to the moved vertex will change, and these are updated.

The move selection procedure is explained in more detail as follows. A net from among the cut nets is chosen at random. With probability p, a vertex is chosen from that net at random, and with probability $1-p$, the vertex that minimizes the cost function under the decision heuristic in use is chosen from the net. If a chosen move is illegal, the selection process is repeated until a legal move is found.

Due to space constraints, we describe only the decision heuristic that performed best on the benchmarks in this paper, the *normalized node score heuristic.*

Normalized Score Heuristic: The FM heuristic is concerned with the "gain" of a node, determined by the number of nets that can be "cut" or "uncut" by moving the node. Therefore, it only considers "critical" nets that can be cut or uncut in a single move. It is mentioned in [4] that a more thorough measure would also estimate the effort required to uncut the net. We propose a new measure, the "score" of a node, defined as the number of moves required to uncut the net if the node in question was moved. In practice, scores are normalized to avoid bias due to net sizes. We can quickly approximate the score for a given node for a particular net as follows. Consider a node, n_i, presently in partition j ($j = 0/1$). Let e_k be a hyperedge containing n_i, d_k be the degree of e_k, and $m_{k,j}$ be the number of nodes from edge e_k in partition j, including n_i. Then the normalized score of n_i due to e_k is $\frac{m_{k,j}-1}{e_k}$. The total node score is the sum of the scores due to all nets in which the node occurs. The idea of using a score comes from the work in [6], which discusses an adaptation of WalkSAT to formulas with pseudo-boolean constraints. In WalkPart, with probability $1-p$ the algorithm picks the node from the chosen net with the minimum score. For the benchmarks in this paper, our empirical results are strongest when this heuristic is used, and when $p = 0.1$.

4 Hybrid Partitioner

Through empirical evaluation, we observe that WalkPart often outperforms a leading-edge implementation of FM [2], on standard industry circuit benchmarks [1]. Also, WalkPart frequently does well on benchmarks where FM does not, implying that both algorithms do not find similar solutions or search in the same areas of the search space. These observations suggest that a hybrid algorithm might be able to exploit the strengths of both.

The hybrid approach alternates runs of FMPart (the FM implementation available in [2]) and WalkPart. It is known that FMPart reduces the net cut greatly in the first few passes, and then spends time performing more passes for relatively small improvements. The hybrid algorithm starts by performing one or more passes of FMPart, and then performs moves of WalkPart to improve FMPart's solution. Control then switches back to FMPart, and so on. A combined FMPart-WalkPart pass constitutes one pass of the hybrid algorithm. A preset number of hybrid passes are run before the algorithm terminates. The use of a WalkSAT-like randomized component makes it difficult to find obvious convergence criteria. We hope to address this issue in future works. In practice, we observe that the tendency of FM to converge rapidly causes considerable reductions in net cut in a fairly small number of passes. This can be seen from our results in Section 5. Many stochastic local search approaches use such preset limits because of the unpredictable nature of the search.

5 Results

In this section, we discuss the empirical performance of our algorithms. We cannot include the results of all our experiments with different tuning and parameter settings due to limited space. We present results of two relatively simple experiments that illustrate the utility of the hybrid technique. The benchmarks used are standard IBM circuit partitioning benchmarks [1]. Although

Table 2: **Results for 10 passes of our Hybrid Algorithm. At each pass, two passes of FM are followed by 50K moves of WalkPart.**

Benchmark Name	Nodes	Nets	WalkPart			FMPart		Hybrid		Cut Ratio	Time Ratio
			Moves	Time	Cut	Time	Cut	Time	Cut		
ibm01+10	12752	14111	2M	9.16	610.67	1.54	477.00	1.42	**391.66**	0.82	0.92
ibm01+4	12752	14111	500K	2.56	506.00	1.39	464.33	1.53	**439.67**	0.94	1.1
ibm02+10	19601	19584	500K	3.61	329.67	2.15	402.00	1.9	**306.33**	0.76	0.88
ibm02+4	19601	19584	500K	3.61	654.33	2.14	386.00	2.0	**344.67**	0.89	0.93
ibm03+10	23136	27401	2M	14.15	1603.33	3.84	1487.67	2.62	**1244.33**	0.83	0.68
ibm03+4	23136	27401	500K	3.84	**1175.00**	3.14	2488.00	3.25	1351.33	0.54	1.03
ibm04+10	27507	31970	500K	4.50	1589.00	5.30	**718.67**	2.9	1131.33	1.57	0.54
ibm04+4	27507	31970	2M	14.94	2121.00	3.52	2505.00	2.76	**1071**	0.42	0.78
ibm05+10	29347	28446	2M	17.34	4804.67	9.54	2793.33	7.6	**2612.33**	0.93	0.79
ibm06+10	32498	34826	2M	12.73	**750.00**	5.63	1683.00	4.1	1160.33	0.68	0.72
ibm07+10	45926	48117	2M	18.07	2483.33	9.63	**1347.33**	9.3	1687	1.25	0.96
ibm08+10	51309	50513	2M	24.92	1902.67	11.76	3076.00	10.1	**1501.33**	0.48	0.85
ibm09+10	53395	60902	2M	19.60	2069.00	14.38	1936.00	11.98	**1676.33**	0.86	0.83
ibm10+10	69429	75196	2M	19.79	3414.67	15.36	2200.33	21.33	**1520.66**	0.69	1.38
ibm11+10	70558	81454	2M	21.61	**2433.33**	19.78	4203.33	26.67	4072.33	0.96	1.34
ibm12+10	71076	77240	2M	22.99	3803.67	23.03	4200.67	30.01	**3781**	0.90	1.30
Average										0.84	0.93

the benchmarks are chosen from the VLSI domain, all the algorithms discussed here are general and can be used on instances from any application. In the experiments, WalkPart uses the normalized score heuristic, and randomness quotient $p = 0.1$. [2] We use the FM partitioning package by Caldwell et. al., available at [2], for the FM component. WalkPart is written in C++.

In the first experiment, we run 10 hybrid passes after first running FM once to completion. Each hybrid pass performs 100,000 WalkPart moves followed by a single FMPart pass. The solution after a pass is used as the starting solution for the next pass. Results for this experiment are shown in Table 1.

The hybrid algorithm is compared against FMPart and WalkPart, run individually. WalkPart uses the normalized score heuristic and $p = 0.1$, with two different move limits: 500,000 and 2,000,000. Because some moves are picked randomly, increasing the move limit does not always produce better solutions. We show results for the move limit where net cut was lower. For each benchmark we report average solution quality and runtime over three independent starts on a 1.2 GHz AMD Athlon processor with 1-2GB, running Linux.

In the table, columns 1-3 describe benchmark statistics. The benchmarks are named `ibm`*no*+*t*%, where `ibm` indicates an IBM benchmark, *no* is the serial number, *t* indicates the balance constraint. Tolerances specify the maximum legal disparity between subset weights as a percentage of total vertex weight. The next three columns give WalkPart's number of moves, runtime and resulting net cut. The next four columns show runtimes and net cuts for FM and the hybrid algorithm. The lowest net cut out of the 3 algorithms is boldfaced. The last two columns show the ratio of net cut and runtime of the hybrid algorithm to net cut and runtime of FMPart. Averages are shown at the bottom.

We observe that the hybrid algorithm achieves smaller cuts than both FMPart and WalkPart on most benchmarks. This is to be expected, since hybrid passes are run on a solution produced by FMPart. Nevertheless, we can make some important observations. First, FMPart frequently converges when a much better solution is achievable. Second, it is possible to achieve a better solution by introducing a randomized component with a different decision heuristic (normalized score). WalkPart can be used to lead FMPart out of local minima by performing random moves, after which FMPart can be run again to find a better solution. However, this experiment has a runtime cost. On average, this strategy improves run time by 30% over FMPart, but is 1.7 times slower.

Our second experiment improves runtime by avoiding the initial run of FMPart. This experiment also uses 10 hybrid passes. Each pass consists of 2 passes of FMPart followed by 50,000 moves of WalkPart. Results for this experiment are in Table 3.

It is clear that the second hybrid configuration is competitive with FMPart and WalkPart in terms of runtime, and frequently produces better solutions. Also, while either FMPart or WalkPart may achieve lower net cut than the hybrid algorithm, it is very rare that both of them do so. The hybrid approach is usually close to the best solution achieved by either. On average, the second hybrid configuration improves net cut by 15% and is about 7% faster than FMPart alone.

6 Conclusions and Future Work

In this work, we introduce our "crossover" hypergraph partitioning algorithm, WalkPart. We combine WalkPart with the well-known Fiduccia-Mattheyses (FM) heuristic for hypergraph partitioning to produce a hybrid heuristic that draws on the strengths of both techniques. We achieve substantial improvement over FMPart with a very simple approach that requires almost no tuning.

Our first experiment shows that net cuts achieved by FM, a widely used algorithm for this application, can be significantly improved (by 30%), albeit with a runtime cost. However, our second experiment reduces net cuts by an average of 15%, and runtime by 7% over FM. Our results show that combining these two heuristics results in a search procedure that is more powerful than either heuristic used alone. Strong empirical results for the proposed hybrid algorithm motivate further research. We are incorporating our flat partitioning algorithms into the multilevel framework [4, 2] and plan to make it available for use in CAD tools. We are also continuing our investigation of effective search procedures for partitioning and looking for new techniques that can be hybridized with existing ones.

References

[1] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite", *Proc. ISPD 1997*, pp. 80-85

[2] A. E. Caldwell, A. B. Kahng and I. L. Markov, UCLA Physical Design Tools Release: http://nexus6.cs.ucla.edu/software/PDtools/tar.gz/

[3] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions", *Proc. ACM/IEEE Design Automation Conference,* 1982, pp. 175–181.

[4] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Applications in VLSI Design", *in Proc. ACM/IEEE Design Automation Conference,* 1997, pp. 526-529.

[5] B. Selman, H. Kautz, and B. Cohen, "Noise strategies for improving local search", *AAAI '94*, pp. 337–343.

[6] J. P. Walser, "Solving Linear Pseudo-Boolean Constraint Problems with Local Search", *AAAI '97*.

[2] Parameter settings were not chosen arbitrarily. After testing, we chose the values that gave best overall performance.