SAT-Based Optimal Hypergraph Partitioning with Replication

Michael G. Wrighton[1]
Tabula, Inc.
3250 Olcott St.
Santa Clara, CA 95054
mwrighton@tabula.com

André M. DeHon
California Institute of Technology
Computer Science, 256-80
Pasadena, CA 91125
andre@cs.caltech.edu

ABSTRACT

We propose a methodology for optimal k-way partitioning with replication of directed hypergraphs via Boolean satisfiability. We begin by leveraging the power of existing and emerging SAT solvers to attack traditional logic bipartitioning and show good scaling behavior. We continue to present the first optimal partitioning results that admit generation and assignment of replicated nodes concurrently. Our framework is general enough that we also give the first published optimal results for partitioning with respect to the *maximum subdomain degree* metric and the *sum of external degrees* metric.

We show that for the bipartitioning case we can feasibly solve problems of up to 150 nodes with simultaneous replication in hundreds of seconds. For other partitioning metrics, we are able to solve problems up to 40 nodes in hundreds of seconds.

1 INTRODUCTION

Balanced, k-way hypergraph partitioning is a fundamental problem in the design of integrated circuits. The precise details of the partitioning problems vary by application [1], but all known useful formulations of balanced partitioning result in NP-hard optimization problems.

Although effective heuristics exist to solve many partitioning problems, few provably optimal solution techniques have been explored. Where optimal techniques have been applied, significant quality per given runtime improvements have been observed over heuristics for small problem sizes [2]. Rather than focus on time-quality tradeoffs, we decided to explore the limits of solving partitioning optimally.

A key definition in computer science is that any NP-complete problem can be transformed into any other NP-complete problem given only a polynomial amount of time and space [3]. Seminal work nearly two decades ago [4] suggested that partitioning can be efficiently remapped to Boolean satisfiability. Recent advances in SAT solvers allow enormous SAT instance to be solved (or proven UNSAT) within seconds [5]. These results have been obtained across a range of benchmarks represented in a canonical "conjunctive normal form" (CNF). Annual competitions have yielded rapid SAT solver progress in recent years.

A key limitation of the early SAT-mapped partitioning work is that the published results handled a very traditional and specific partitioning formulation. "Real-world" partitioning problems may have complex formulations [6]. The author of the early work, however, recognized a broader potential of the SAT-mapped approach [4]:

"An attractive feature of this approach is that the entire space of feasible solutions can be represented in a compact way, facilitating the search for optimal solutions under complex cost functions and associated constraints."

We developed and benchmarked a framework for SAT-mapping more complex cost functions and constraints than considered in prior work on optimal partitioning. This framework allows us to construct optimal k-way partitions. Further, we consider partitioning balance constraints that allow nodes to appear in multiple partitions (i.e. replication). To illustrate cost function flexibility, this work considers three well-established metrics for partitioning quality: The traditional *total cut hyperedges* metric penalizes every edge that is not fully contained within a single partition. The *sum of external degrees* function penalizes every entry or exit of a wire from a partition. In multi-way partitioning problems, this may be appropriate as solvers targeting this cost function prefer solutions where edges interact with small numbers of partitions over solutions where the same number of edges are split over all of the partitions. Finally, the *maximum subdomain degree* metric limits the maximum IO into any given partition as opposed to the average IO over all of the partitions. Though runtime remains an issue for the more sophisticated metrics, we show techniques that are sufficiently general to consider cost functions that are tightly defined by their application domain.

With this work, we expand the literature on optimal partitioning with the following innovations:

- SAT formulations for three distinct partitioning metrics
- An enhancement for optimal k-way partitioning (as opposed to simple bipartitioning)
- Concurrent replication and partitioning of nodes as a natural part of problem formulation

We show that when considering the traditional bipartitioning problem, our technique scales to more difficult problems than a branch-and-bound implementation. Within that formulation, we can consider integrated replication set generation for not more than an order of magnitude runtime penalty. Finally, for some small netlists, we show the first published optimal results for two sophisticated cost metrics, with and without replicated nodes.

2 PRIOR WORK

We are aware of a single published work on hypergraph partitioning via Boolean SAT [4]. Devadas considers the traditional formulation of hypergraph bipartitioning. He shows that within a reasonable amount of time (14 minutes in the longest case), the hardware and SAT solvers of 1989 could optimally bipartition a benchmark netlist of 32 nodes under the *total cut hyperedges* metric.

Instead of SAT, the recent work to date on optimal netlist partitioning has focused on branch-and-bound techniques — an approach to which the cut hyperedges metric avails itself due to several clever techniques available in that formulation [2]. These optimizations, by their nature, are limited to the traditional bipartitioning formulation.

Previous implementations of k-way and broader cost metric partitioning have been limited to heuristic techniques. For example, Karypsis's group at the University of Minnesota has developed partitioners that generate k-way partitionings to minimize the *sum of external degrees*, *total cut hyperedges*, and *maximum subdomain degree* metrics [7, 8].

The best-known techniques for allowing replication in order to improve partitioning quality are based on network flows [9, 10]. These techniques offer exact solutions to several unbalanced formulations of the bipartitioning problem. At the cost of optimality, they may be used as kernels in heuristics for k-way, balanced hypergraph partitioning and replication.

[1] This work was done while the first author was at Caltech.

0-7803-9451-8/06/$20.00 ©2006 IEEE.

The partitioning problem is fundamentally no different from any other finite-domain constraint satisfaction problem. This fact led us to apply unsuccessfully a Prolog-based constraint solver [11] to the task. We are not the first to note that constraint solvers which remap their problems to CNF can be significantly faster than solvers that operate on a more direct problem formulation [12].

There are numerous solvers available for SAT instances, a complete exploration of which is beyond the scope of this paper. Readers may refer to annual SAT Solver Competitions at ICSAT for the current state-of-the-art in solvers. We found that `siege_v4` [13] (which does not participate in the competition due to a 'black-box' restriction) generally provided the best performance for our SAT instances.

3 SOLUTION OUTLINE

A well-known technique for solving an NP-hard optimization problem is to transform it into a series of NP-complete decision problems. Generically, we represent this as:

```
while (upperBound != lowerBound) {
 thisTry = (upperBound + lowerBound) / 2;
 if (existsSolutionLessThan(thisTry)) {
   upperBound = thisTry;
 }
 else
   lowerBound = thisTry;
}
```

The kernel of our approach is that we build a SAT problem instance which is satisfiable if, and only if, the circuit netlist can be partitioned with a goodness metric less than or equal to some target (an NP-complete decision problem). In order to be a valid solution which meets a particular cost metric we will assert that:

$$SAT = AllNodesRepresented \wedge PartitionsBalanced \wedge MetricMet$$

Then after obtaining a SAT/UNSAT result, we have a new upper or lower bound on the optimal solution to use in a binary search for the minimum cost partitioning. Once the bounds are tight, the satisfying assignment to the SAT instance associated with the best goodness metric implies a solution to the NP-hard optimization problem. If we wish to relax the optimality constraint, we can treat problem instances where the solver times out as UNSAT. Otherwise, we must report the optimization problem as unsolved. Our experience has shown that SAT instances asserting a cost metric some distance from the minimum can be solved or proven UNSAT much more quickly than those very near the minimum. We found that this effect was more significant than the count of CNF clauses and variables.

The SAT instances we generate have kN binary inputs, representing which nodes appear in each of the partitions. For 3-way partitioning, the graph in Figure 1 implies these inputs to a satisfiability instance:

$$A_0, A_1, A_2, B_0, B_1, B_2, C_0, C_1, C_2, D_0, D_1, D_2, E_0, E_1, E_2$$

A '1' in the satisfying input assignment indicates that a node appears in a particular partition. This contrasts with Devadas' approach [4], which encodes the partition assignment with a single bit per node.

Having outlined the approach, we have several natural questions.

- How do we assert a valid partitioning?
- How are the various metrics specified in the SAT instance?
- How does the SAT approach scale against other solutions?
- What size problems are feasible to solve for the various cost metrics?

The following sections answer these questions.

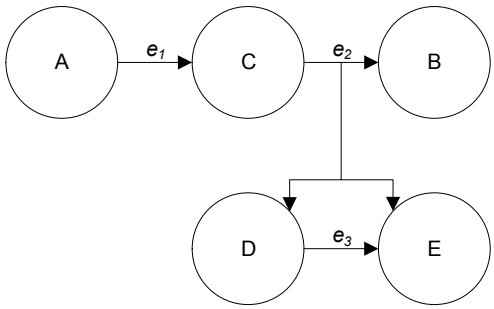

Figure 1. An Example Hypergraph

4 THE DETAILS

As stated in the previous section, we construct a SAT instance (in the form of a Boolean logic circuit). We show how each higher-level asserted variable in our circuit is constructed.

4.1 Cardinality Constraints

At several points, we will need to make an assertion about the number of '1's set over some number of variables. Our notation in this work represents '1's counters with the Σ symbol, both in equations and diagrams. In text, we refer to the '1's counter as a totalizer. We expand on our technique for asserting cardinality constraints in Section 5.1.

4.2 All Nodes Represented

In order to be a valid solution we assert that each node appears in at least one partition. Continuing with the example of a three-way partitioning of the netlist from Figure 1:

$$\begin{aligned} AllNodesRepresented = {} & (A_0 \vee A_1 \vee A_2) \wedge \\ & (B_0 \vee B_1 \vee B_2) \wedge \\ & (C_0 \vee C_1 \vee C_2) \wedge \\ & (D_0 \vee D_1 \vee D_2) \wedge \\ & (E_0 \vee E_1 \vee E_2) \end{aligned}$$

Or more generally, for a k-way partitioning:

$$AllNodesRepresented = \bigwedge_{A \in Nodes} \left(\bigvee_{0 \le i < k} (A_i) \right)$$

If the problem formulation we are considering does not admit replication, then we can assert that no node appears in two (or more) partitions:

$$NoReplicants = \bigwedge_{A \in Nodes} \bigwedge_{\substack{0 \le i < k \\ i+1 \le j < k}} \overline{A_i \wedge A_j}$$

4.3 Partitions Balanced

We make an assertion on the cardinality of each of the partitions by means of the totalizer described above. If the partitions are balanced then no partition has more than some fraction of the total nodes in the graph.

$$PartitionsBalanced = \bigwedge_{0 \le i < k} \left(\left(\sum_{A \in Nodes} A_i \right) \le MaxSize \right)$$

If replicated nodes are allowed, we may also construct a totalizer for the 'excess' nodes in the design and limit them as desired.

4.4 Metric Met

The subtleties of SAT-mapped partitioning occur when we assert that a metric is less than or equal to some given value. We begin this work by examining a single cost metric. We discuss additional metrics in Section 7.

4.4.1 Total Cut Hyperedges

The *total cut hyperedges* metric is the easiest to describe. We define an edge (e) as a source (e.Source) with a set of sinks (e.Sinks) and assert that it is cut if any of its sink nodes appear in a partition without the source node.

$$Cut(e) = \bigvee_{0 \le i < k} \bigvee_{s \in e.Sinks} \left(\overline{e.Source_i} \wedge s_i \right)$$

We employ a totalizer to sum over all the potentially cut edges, and assert that their cardinality is less than our current target.

$$MetricMet = \sum_{e \in Edges} Cut(e) \le MaxMetric$$

Figure 2 shows how we would build the MetricMet function using the *total cut hyperedges* metric for a three-way partitioning of the sample hypergraph shown in Figure 1.

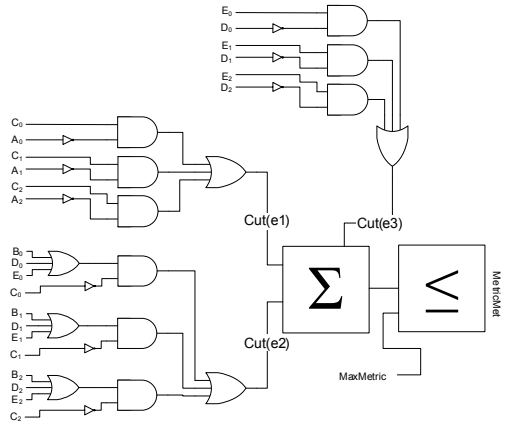

Figure 2. Cut Hyperedges Metric

5 OPTIMIZING IMPLEMENTATION

Having identified problem formulations that guarantee optimal results given unlimited SAT runtime, we move on to consider a few techniques that offer the potential of reducing the total amount of time that the SAT solver will require to solve the generated instances. Intuitively, the key to these techniques is the notion that we can improve the tractability of SAT by assuring that partial assignments of variables allow maximum implications to assign other variables in the instance. We show, on a few sample netlists, how two techniques improve our runtimes.

5.1 Cardinality Constraints

One way to assert cardinality constraints in our SAT instances is via a tree of adders. This is the technique employed by Devadas [4]. A desirable property of this approach is that it requires few clauses or variables in the generated SAT instance. However, SAT solvers operate on a partial assignment of variables. Figure 3 shows that if we limit the '1's cardinality over a set of four variables to one, even after three of the input variables are set to '1', the solver will not be able to prune the search space.

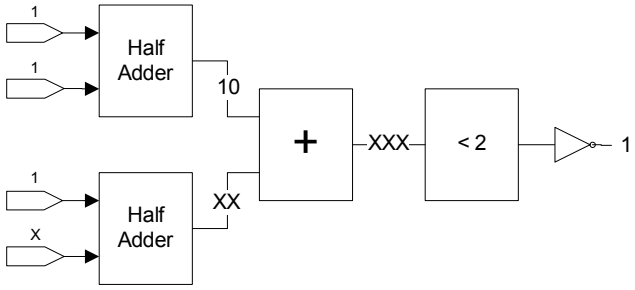

Figure 3. Cardinality constraint expressed in binary arithmetic

We improve on Devadas' cardinality constraints by sacrificing the brevity of the binary approach for the strategy of Bailleux and Boufkhad [12, 14]. They construct binary trees of *totalizers*, each operating on pre-unate[2] input. The pre-unate representation allows the solver to more quickly discover conflicts and prune the search space with fewer input variables assigned.[3] We then assert SAT on the appropriate inverted output from the top-level totalizer. Figure 4 shows how a SAT solver would interpret the cardinality constraint from before with the same partial assignment of inputs. In fact, as soon as any two input pins are assigned '1', the totalizer tree will produce a contradiction. The complexity of the CNF encoding is $O(N^2)$ clauses and $O(N \log (N))$ variables if we are constraining N variables to a sum of N-1 (the most pathological case).

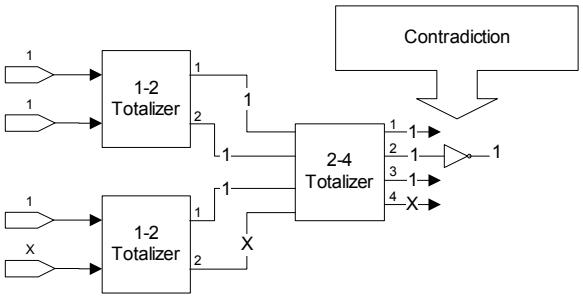

Figure 4. Cardinality constraint expressed with pre-unate totalizers

5.2 Symmetry Breaking

In a k-way partitioning, we have *k!* potential orderings (i.e. symmetries) of the partition sets. When we generate the satisfiability instance, we can break one degree of symmetry by preassigning a single node to an arbitrary partition. We considered several potential heuristics to select the breaking node: The node connected to the most other nodes; the node connected to the least other nodes; and a random node. Table 2 shows that *not* breaking the symmetry reliably causes the longest runtimes over our selected netlists. When we do break the symmetry, choosing a random node seems to provide as good a result as any of the more crafted selection heuristics.

For k > 2 partitioning, preassigning a single node for symmetry breaking leaves obvious symmetries unaddressed. Adding another hard symmetry breaking preassignment (assigning another node to another arbitrary partition) would destroy our optimality guarantee. However, we can safely preassign a second node to *either* the same partition as the first *or* another arbitrary partition. We generalize the technique by inserting a "weak backbone" as a symmetry constraint. We construct a weak backbone for the netlist as follows:

```
weakBackbone(k, netlist) {
 P = heuristicPartition(k, netlist)
 for i ← 0 to k – 1
 do
   Node = randomSelection(P[i])
   assertInOneOf(Node, 0..i)
}
```

The power of this approach is that it allows us to imply an ordering of the partitions, but without removing the optimality guarantee from the result. Table 2 shows that the weak-backbone approach provides better scalability than simply constraining a single node as the number of partitions generated increases.

[2] "Pre-unate" represents k in N bits by setting the first k bits to '1' and the trailing N - k bits to '0'. For example, if N = 5, three is expressed as '00111'.

[3] More formally, unit propagation on any subset of assigned variables restores generalized arc-consistency.

Table 1. Effect of Cardinality Constraints

Netlist	Size	k	SAT Runtime (ms)		
			Binary	Bailleux & Boufkhad	Speedup
ex4	55	2	16360	2762	5.9
		3	20130	4168	4.8
		4	42877	10960	3.9
		5	93999	15332	6.1
		6	174480	22921	7.6
misex2	97	2	16098	1246	12.9
		3	59514	14344	4.1
		4	105858	14678	7.2
		5	160268	16358	9.8
		6	524047	62035	8.4
5xp1	100	2	73936	8631	8.6
		3	207867	40410	5.1
		4	716010	102163	7.0
		5	Timeout	243489	-
		6	Timeout	652259	-
f51m	114	2	7416	2010.25	3.7
		3	25970	3049	8.5
		4	27988	2694	10.4
		5	135951	9881	13.8
		6	533034	34044	15.7
kirkman	151	2	400696	22714	17.6
		3	742069	75493	9.8
		4	1442291	160343	9.0
		5	Timeout	Timeout	-
		6	Timeout	Timeout	-

6 BENCHMARKING RESULTS

In order to show the potential for practical usefulness of our technique, we quantify the performance of our SAT mappings. In this section, we examine our benchmark results for traditional bipartitioning with cut hyperedges as the metric – we perform a side-by-side comparison against an optimized and widely-used branch-and-bound partitioner. We continue to show that adding replication to this formulation does not cause runtime to increase too much.

6.1 Methodology

Our implementation of the approach generates ISCAS89 format files, which represent the satisfiability instances. We convert these to CNF via a Perl script [15]. As benchmark hypergraphs, we employed 4-LUT FPGA mappings (generated via Flowmap [16]) of the small IWLS93 benchmark circuits. These netlists are appropriate to explore this methodology as they consist of nodes of equal size – standard-cell mapped circuits would complicate our construction of balance constraints. We ran our flow on dual-processor 2.8 GHz Intel Pentium 4 machines with 512 KB L2 cache and 4 gigabytes of RAM. Whenever we report a CPU time, it is the total time spent in the SAT solver.

6.2 SAT Solver Choice

The experience of the SAT community is that it is unusual for a given solver to be ideal across a range of problem types. Therefore, we conducted an evaluation of several SAT solvers [13, 17, 18]. We determined that a solver would receive a 'pass' for a partitioning-derived satisfiability instance if it could solve (SAT/UNSAT) the problem within a reasonably long amount of time (which we arbitrarily set at two hours of CPU time). At this stage, we deemed the higher priority to find a solver that would solve many problems than to reduce the average runtime of solved instances. Over many SAT instances, we found that `siege_v4` consistently solved more instances than the other solvers.

If a solver superior to `siege_v4` appears, it is a simple matter to adjust our flow to leverage the new tool because the SAT community has widely standardized upon the CNF representation of problem instances.

Table 2. Effect of various symmetry-breaking choices.

Netlist	Size	k	SAT Runtime (ms)				
			None	Least Conn.	Most Conn.	Rand.	Weak Back.
ex4	55	2	2782	1448	1202	1622	1678
		3	4226	2687	2238	3311	1851
		4	11480	7361	3440	7226	3079
		5	15652	9491	6238	11006	6422
		6	23563	15420	9169	16311	8840
misex2	97	2	1237	1343	998	928	1092
		3	15082	9293	4291	5481	4443
		4	14740	10695	7269	7095	4814
		5	16671	14101	9831	8530	5463
		6	64989	56707	45003	38717	23974
5xp1	100	2	8692	8477	5663	5347	7317
		3	42566	40228	22779	18809	27546
		4	103773	91713	70658	72145	41012
		5	248916	240253	172330	192982	113177
		6	665235	846848	651781	441434	490670
f51m	114	2	2022	1729	1021	2540	1503
		3	3091	2358	1939	2259	2060
		4	2713	2941	2623	2004	2059
		5	10000	13515	12639	12848	10118
		6	34828	32823	31933	25315	21705
kirkman	151	2	23141	18457	18788	16561	17206
		3	77397	66562	71601	52921	47441
		4	164108	136261	172171	97556	60365
		5	Timeout	574638	687024	876982	501577
		6	Timeout	Timeout	Timeout	Timeout	Timeout

6.3 Scalability Against Branch and Bound

We used our approach and the branch-and-bound solver from Capo [2] to optimally bipartition our benchmark circuits (which have from 10 to 255 nodes). We allowed up to a 10% unbalanced partitioning. We validated that the partitioning metrics generated between the two solvers were identical. Figure 5 shows that while the branch-and-bound approach is superior for many netlists, as the complexity increases, our approach dominates. We present plots sorted by both SAT runtime and branch-and-bound runtime because merely considering node count is not a strong enough predictor of problem complexity to give a clear visualization.

We observe that as branch-and-bound's runtime increases, our SAT-mapped formulation offers much better scaling properties. Branch-and bound times out on many of the benchmarks that SAT completes (even though we allowed the branch-and-bound implementation 10x longer runtime). There are no examples in the benchmark set however where SAT times out and branch-and-bound does not.

6.4 Bipartitioning with Integrated Replication

Typically, adding replication to the logic bipartitioning SAT formulation does not increase the runtime by more than an order of magnitude. We considered the case of allowing two partitions, each 60% of the total size of the netlist. Table 3 shows that, for large benchmarks of fewer than 152 nodes (the largest netlists we could reliably partition without timing out), we can obtain improvements in cutsize with a modest overhead in compute time over the non-replicated case. In many cases, the freedom to replicate nodes allows us to find an optimal solution even more quickly.

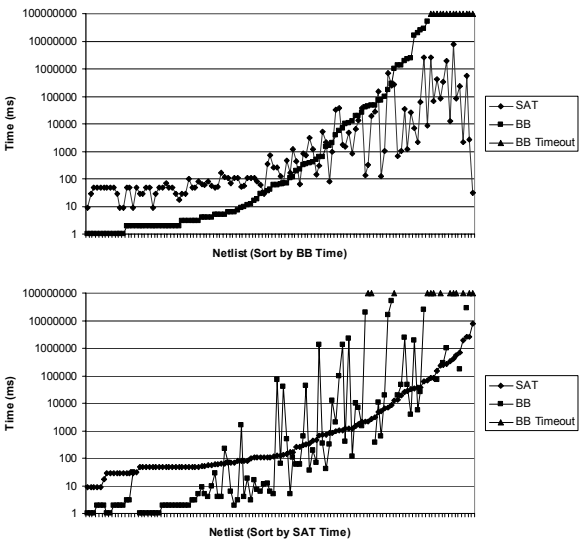

Figure 5. SAT vs. Branch-and-Bound Scaling

7 BEYOND CUT HYPEREDGES

In principle, a key feature of the SAT formulation is that we need not be limited to simple formulations of the partitioning problem. It is not difficult to construct satisfiability instances that represent more sophisticated cost metrics than total cut hyperedges. We conducted experiments on two such metrics (which are formally described in the appendix) and show results for three benchmarks of forty nodes. We report the first results from an optimal algorithm to solve partitioning for these metrics (and allow replicated nodes in the formulation) – unfortunately, our results to date indicate that the SAT formulation appears to be a very slow method of attacking these problems. Our results consider dividing the nodes into partitions of maximum size = (1.2 / k × total node area). In the replicated case we allow nodes to appear in multiple partitions while keeping the maximum size fixed. We report a timeout when the SAT solver times out at 1200 s.

Table 3. Optimal Bipartitioning with Simultaneous Replication

Netlist	Size	No Replication		Replication		Slowdown	% Cutsize Impr.
		Cut	ms	Cut	ms		
c8	131	8	1413	8	2228	1.58	0
sao2	133	15	188887	10	7401	0.04	33
s641	135	13	55061	10	16559	0.30	23
s713	137	13	56494	10	12840	0.23	23
mm9b	141	17	344367	15	3348853	9.72	12
C1355	147	16	32097	16	117767	3.67	0
C499	147	16	28155	16	292111	10.38	0
cse	148	18	1522416	11	221276	0.15	39
cht	151	5	170	5	145	0.85	0
kirkman	151	12	11317	9	15006	1.33	25
Avg.						**2.82**	**15.5**

7.1 Sum of External Degrees

When we consider the *sum of external degrees cost* ("SOED") function, we must consider every hyperedge as a potential external degree of one or more partitions. If a net is cut, it will appear as an output degree on exactly one partition and an input degree on at least one partition. A totalizer tree sums over all the potential partition pins (all the hyperedges in the hypergraph for every partition for inputs, and all the hyperedges again for outputs).

$$MetricMet = \begin{array}{l} \sum_{\substack{e \in Edges \\ 0 \le i < k}} \left(Cut(e) \wedge \overline{e.Source_i} \wedge \left(\bigvee_{s \in e.Sinks} (s_i) \right) \right) \\ + \sum_{e \in Edges} Cut(e) \end{array} \le MaxMetric$$

Table 4 shows that our SAT formulation is not yet efficient enough to reliably optimize this cost function, even for small, forty node netlists. We present the results to show our current progress on optimizing this metric.

Figure 6 shows how we assert the sum of external degrees function for a three-way partitioning of the example circuit in Figure 1.

Table 4. Sum of External Degrees Optimization

Netlist	k	No Replication		Replication	
		SOED	ms	SOED	ms
misex1	2	25	195	21	79
	3	33	6538	27	1434
	4	35	5824	34	220448
	5	42	895024		Timeout
	6		Timeout		Timeout
bbara	2	22	428	18	423
	3	31	45958	28	640239
	4	38	2515958		Timeout
	5		Timeout		Timeout
	6		Timeout		Timeout
ex7	2	22	897	18	1647
	3	32	119169		Timeout
	4		Timeout		Timeout
	5		Timeout		Timeout
	6		Timeout		Timeout

7.2 Maximum Subdomain Degree

Minimizing the *maximum subdomain degree* ("MSD") requires the most intricate SAT formulation. At first, it appears sufficient to modify slightly the SOED formulation. However, the replicated nodes create an additional complexity. We must only charge one partition for the output from a replicated node.

We employ totalizers at several points in the assertion of the MSD value. First, we employ totalizers for each partition to sum the number of input pins into each partition – this is similar to the SOED metric. Then we totalize the outputs.

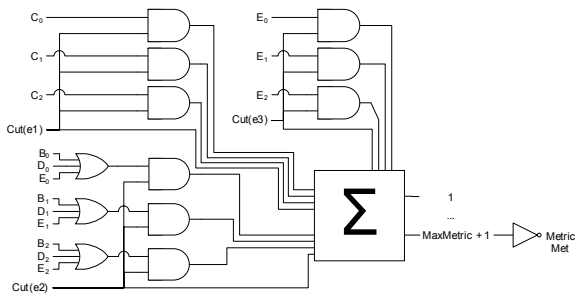

Figure 6. Sum of External Degrees Metric

We assert that if an edge is cut, its output node must count as an output in exactly one of the partitions where it appears. We employ separate totalizer trees to sum the inputs and outputs over each of the partitions (subdomains). We assert that for every partition, $Sum_{MaxMetric+1}$ is false. If we wish, this technique is easily extended to to solve the FPGA clustering problem (where each cluster would typically have a particular number fixed inputs and fixed outputs as opposed to general IO pins [6]). Figure 7 shows the satisfiability instance we construct for this metric on the example hypergraph in Figure 1.

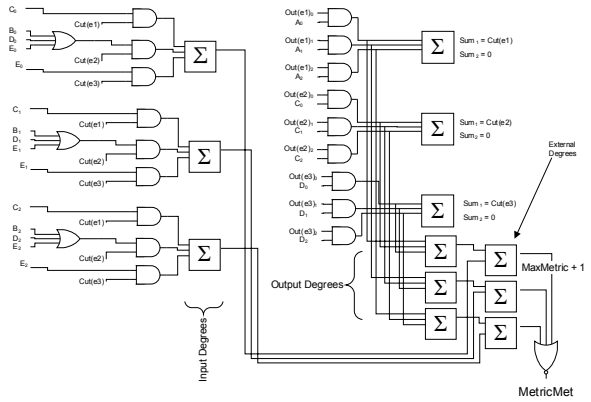

Figure 7. Maximum Subdomain Degree Metric

Table 5 shows that our formulation of the MSD problem scales better than the SOED formulation. Further, we see at least anecdotally, that allowing replication improves results for this metric.

8 FUTURE DIRECTIONS

We believe that the flexibility introduced by representing partitioning as a SAT problem creates new opportunities for optimal solutions with diverse notions of partitioning quality. Future work will broadly consider how we can improve the scaling properties.

8.1 Higher K Partitioning

So far, we have found that as K is increased, the time to compute an optimal partition set goes up quickly. We believe that this could be somewhat alleviated if more of symmetries in partition ordering could be broken. If we were to optimize a partitioning metric where the partitions naturally form a spatial order, we could break this symmetry.

8.2 Hybrid Cost Functions

We have seen that two cost functions other than cut-hyperedges are very difficult to solve optimally with SAT. The cut-hyperedges solutions may be very helpful in identifying the gross locality in the hypergraphs. We believe that by creating a hybrid cost function, we can significantly speedup the search for solutions to more complex partitioning problems. For example, at the cost of our optimality guarantee, we might seed an MSD or SOED problem instance with some hard preassignments from a cut hyperedges solution.

9 CONCLUSION

We have shown several new problem formulations and speedup techniques for optimally solving hypergraph partitioning by remapping the problem to Boolean satisfiability. Combining these techniques with a leading edge SAT solver allows us to perform optimal multiway partitioning with integrated optimal replication for small benchmark netlists. For the traditional logic bipartitioning cost function, our method scales better than the optimal branch-and-bound algorithm in an academic placer. Under this model, we can add integrated generation of replicated nodes to the problem formulation for relatively small additional runtime.

We further show how our framework allows us to produce the first published optimal results of k-way logic partitioning under two more sophisticated cost metrics. We generate these with and without replicated nodes. Our runtime results to date in this effort suggest that Devadas' claim of SAT's applicability to broader cost functions may have been premature. Future work will consider how we can improve the scaling properties of SAT-based solutions to these problems.

Appendix

In this work, we consider several well-known partitioning cost metrics. For the reader's convenience, we formalize our definitions in this appendix. We adapt the descriptions given in [8] to describe our hypergraph model and partitioning metrics along with their evaluation. We make appropriate modifications to expand the model to include a notion of node replication appropriate for VLSI layout applications.

Table 5. Maximum Subdomain Degree Optimization

Netlist	k	No Replication		Replication	
		MSD	ms	MSD	ms
misex1	2	14	470	11	120
	3	12	1490	10	2667
	4	11	6734	9	13478
	5	10	14052		Timeout
	6	9	8616	8	1780920
bbara	2	12	525	9	927
	3	11	3979	10	18182
	4	11	13421	9	138265
	5	10	23967	9	107820
	6	10	45501		Timeout
ex7	2	12	3496	9	4586
	3	12	8490	9	99578
	4	11	25455		Timeout
	5	11	122051		Timeout
	6	10	218037		Timeout

A.1 Netlist and Partitioning Definitions

A *netlist hypergraph* $G = (V, E)$ consists of a set of vertices V ("logic elements") and directed hyperedges E ("wires"). Every hyperedge $e = (d, R)$ has exactly one source vertex (d) and a set of at least one sink vertex R. This model is not appropriate for all applications – for example, circuit models which include tristate drivers would not include the limitation of one source node per hyperedge.

A decomposition of V into k subsets $V_1, V_2, \ldots, V_k$, such that $\cap_i V_i = V$ is called a k-way partitioning of V. We refer to each of these subsets as a partition or subdomain. The partitions need not be disjoint if replication is permitted. A k-way partitioning of V satisfies a balance constraint specified by $[l, u]$ if, for each partition $l \leq |V_i| \leq u$.

A.2 Total Cut Hyperedges

The total cut hyperedges metric counts the number of hyperedges that are cut between the partitions. A hyperedge is cut if it has at least one sink vertex $r \in R$ in a partition where the driving vertex (d) is not assigned. Formally, a hyperedge $e = (d, R)$ is cut if there exists V_i s.t. $R \cap V_i \neq \emptyset$ and $d \notin V_i$.

For the example partitioning in Figure 8, the cut hyperedges metric is 4, because edges sourced by vertices A, B, E, and F are shared between multiple partitions.

A.3 Sum of External Degrees

The *sum of external degrees* cost function (SOED) counts the total 'pins' required over the partitions. Partitions have two types of external degrees: input and output. The input set I_i of a partition V_i is the set of hyperedges which form its input. V_i includes at least one sink vertex from i and does not include the source vertex from i. We compute the set of output degrees overall by assigning an output degree to every input degree $\cap_i I_i = O$. Note that we do not assign the output edges to particular partitions – this is significant as the source node might appear in multiple partitions. The SOED cost function is $\text{SOED} = |O| + \sum_i |I_i|$.

For the example in Figure 8: V_1 has zero inputs, V_2 has inputs from B, E, and F. V_3 has inputs from A and B. We then have that A, B, E, and F are outputs. So the SOED metric is 0 + 3 + 2 + 4 = 9.

A.4 Maximum Subdomain Degree

The most intricate metric is the *maximum subdomain degree* (MSD) criteria. We expand from the computation of the SOED metric. We separate the output hyperedge set O into disjoint subsets $O_1,\ldots,O_k$ s. t. $\cap_i O_i = O$ and $\text{MSD} = \text{MAX}_i (|O_i| + |I_i|)$ is minimized. In the non-

replicated case, the computation of the MSD metric for a given partitioning is trivial (the output hyperedges are constrained to assignment in the partitions where their source nodes are assigned). If we allow node replication, then assigning the output hyperedges to minimize the MSD for a given partitioning is more involved. Each output hyperedge o with source vertex d may be feasibly assigned to a partition $Q \in \{V_1, \ldots, V_k\}$ s.t. $d \in Q$.

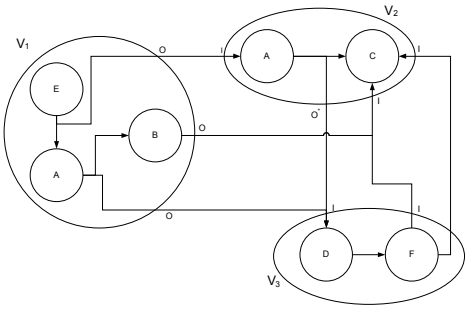

Figure 8. A Partitioned Hypergraph

For the example in Figure 8 we begin with the input assignments from SOED: V_1 has zero inputs, V_2 has inputs from B, E, and F. V_3 has inputs from A and B. We must assign the output E to V_1 and the output F to V_3. Before considering output A, V_1 has degree 2, V_2 degree 3 and V_3 degree 3. By choosing to assign the output A to V_1 instead of V_2, the MSD metric is 3.

In debugging our work and examining non-SAT generated partitions, we found that evaluating MSD with of replicated nodes is non-trivial.

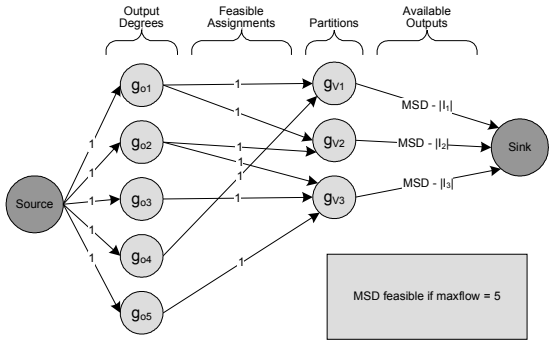

Figure 9. Max-flow Output Assignment Formulation

Our technique for evaluated the MSD metric without SAT is as follows: We perform a binary search on potential MSD values to determine the minimum MSD. We formulate the decision problem as max-flow problem as follows:

For a particular value of MSD, we allow each partition V_i to have output degree of o_i = MSD - $|I_i|$. We construct a directed flow graph that consists of a source with unit capacity flows leading to nodes representing each output hyperedge to assign. We create nodes g_{vi} for each of the potential partitions to assign outputs. We create unit flow edges from the nodes representing outputs to each of their feasible partitions. We connect every node g_{Vi} representing partitions to a sink via edges of capacity o_i. We compute the maxflow. If the max flow is equal to the total output degree, then the links with positive flow imply an assignment of output hyperedges to partitions and a demonstration that the potential MSD is feasible. If the maxflow is less than the total output degree, the specified MSD is infeasible. Figure 9 shows an example of flow graph constructed for assigning five output edges (two of which are driven by replicated nodes) to three partitions.

Acknowledgements

This research was funded in part by the NSF CAREER program under Grant CCR-0133102. We are grateful for the Capo source code, provided by Igor Markov. Discussions with Michael deLorimier were key in understanding how to evaluate the MSD metric.

REFERENCES

[1] F. M. Johannes, "Partitioning of VLSI Circuits and Systems," presented at ACM/IEEE Design Automation Conference, 1996.

[2] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-cell Layout," *IEEE Transactions on Computer-Aided Design*, vol. 19, pp. 1304-1313, 2000.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*: W. H. Freeman, 1979.

[4] S. Devadas, "Optimal Layout via Boolean Satisfiability," presented at International Conference on Computer-Aided Design, 1989.

[5] L. Zhang and S. Malik, "The Quest for Efficient Boolean Satisfiability Solvers," presented at International Conference on Computer Aided Verification, Copenhagen, 2002.

[6] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," presented at IEEE Custom Integrated Circuits Conference, 1997.

[7] G. Karypis and V. Kumar, "Multilevel K-way Hypergraph Partitioning," presented at Design Automation Conference, 1999.

[8] N. Selvakkumaran and G. Karypis, "Multi-Objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization," presented at International Conference on Computer-Aided Design, 2003.

[9] W.-K. Mak and D. F. Wong, "Minimum Replication Min-Cut Partitioning," *IEEE Transactions on Computed-Aided Design for Integrated Circuits and Systems*, vol. 16, pp. 1221-1227, 1997.

[10] L. J. Hwang and A. E. Gamal, "Min-Cut Replication in Partitioned Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 96-106, 1995.

[11] M. Wallace and A. Veron, "Two Problems - Two Solutions: One System - Eclipse," presented at IEE Colloquium on Advanced Software Technologies for Scheduling, 1993.

[12] O. Bailleux and Y. Boufkhad, "Full CNF-Encoding: The Counting Constraints Case," presented at International Conference on Theory and Applications of Satisfiability Testing, 2004.

[13] L. Ryan, *Efficient Algorithms for Clause-Learning SAT Solvers*. Masters thesis: Simon Fraser University, 2004.

[14] O. Bailleux and Y. Boufkhad, "Efficient CNF Encodings of Boolean Cardinality Constraints," presented at International Conference on the Principles and Practice of Constraint Programming, 2003.

[15] J. M. Silva, Personal Website. http://sat.inesc-id.pt/~jpms/scripts/.

[16] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Transactions on Computer-Aided Design*, vol. 13, pp. 1-12, 1994.

[17] A. Nadel, *Backtrack Search Algorithms for Propositional Logic Satisfiability*. Masters thesis: Tel-Aviv University, 2002.

[18] N. Eén and N. Sörensson, "An Extensible SAT-solver," presented at International Conference on Theory and Applications of Satisfiability Testing, 2003.