Hybrid Symbolic-Explicit Techniques for the Graph Coloring Problem

S. CHIUSANO, F. CORNO, P. PRINETTO, M. SONZA REORDA

Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

Abstract

This paper presents an algorithmic technique based on hybridizing Symbolic Manipulation Techniques based on BDDs with more traditional Explicit solving algorithms. To validate the approach, the graph coloring problem has been selected as a hard-to-solve problem, and an optimized solution based on hybrid techniques has been implemented. Experimental results on a set of benchmarks derived from the CAD for VLSI area show the applicability of the approach to graphs with millions of vertices in a limited CPU time.

1. Introduction

Many search and optimization problems in CAD for VLSI can be modeled resorting to graph theory. This motivates the efforts to improve algorithmic solutions to graph problems and to customize general-purpose algorithms to special categories of problem instances commonly found in the electronic CAD field.

Several optimization problems in the CAD area can be modeled as a Graph Coloring problem. For instance, register allocation, resource sharing, and state minimization are natural candidates to be solved resorting to it. This paper concentrates on efficiently finding exact and approximate solutions to the graph coloring problem.

Problems arising in graph theory have long been the subject of investigation of Operation Research, and many efficient algorithms have been developed. Both exact and approximate solutions can be found, and reasonable efficiency is achieved by backing common optimization procedures (e.g., branch and bound, linear programming) with heuristics.

When graph size increases, the memory required for the data structure representing the graph soon exhausts the available resources. There is therefore a limit of some thousand nodes on the graphs that can be analyzed on a standard workstation. Besides, the complexity of many elementary operations, such as finding adjacent nodes to a given node, is usually linear in the size of the graph. This *enumeration* problem significantly contributes to the time complexity of the resulting algorithms.

In Electronic CAD problems it is not unusual to have quite sparse graphs with a large number of vertices, in the orders of millions. The goal of this paper is to develop techniques able to deal with such graphs, and in particular to find good colorings on large sized graphs.

In the last years very efficient techniques for representing and manipulating Boolean functions have been developed using Binary Decision Diagrams (BDDs) [Brya92]. It is possible to handle complex functions of a large number of variables with acceptable memory and CPU time requirements [BRBr90]. In the fields of formal verification, test pattern generation, and automated synthesis, BDDs are employed to represent graphs through *characteristic functions* [BCMD90], that encode a set or a relation as a Boolean function.

Some attempts have been made to profit from the efficiency of BDDs for solving more complex graph problems than traversals. A direct application of symbolic techniques showed their ineffectiveness for the problem at hand [CPSo93], limiting the size of tractable graphs to some hundred nodes. Due to these limitations, in [CPSo95] we developed new algorithmic techniques, named *hybrid techniques*, that aim at exploiting the best features of both traditional and symbolic techniques. The rationale behind hybrid techniques is the encoding of various data structures as *characteristic functions*, while conserving a branch and bound organization of the algorithm. In this framework, most heuristics developed for solving the problem with classical techniques are still applicable, while data structures are extremely more compact and many elementary operations can be performed in shorter time. Experimental results proved that only hybrid techniques are able to solve the maximum clique problem in graphs with millions of vertices.

The goal of this paper is to extend hybrid techniques in order to solve a computationally harder problem (i.e., graph coloring), when graphs used in CAD systems are considered. The paper is organized as follows: Section 2 formalizes the context in which the coloring problem is being considered. In Section 3 an algorithm based on hybrid techniques is presented, while Section 4 reports

ED&TC '97 on CD-ROM

some experimental evidence of the attained efficiency. Section 5 concludes the paper.

2. Graph Coloring

Given an undirected graph $G = (V, E)$, the *graph coloring problem* consists in assigning a color to each vertex of the graph such that no two adjacent vertices get the same color. The minimum number of colors required for graph coloring is called *chromatic number* of the graph. Finding a minimum graph coloring is known to be an NP-complete problem [Gibb85].

This paper proposes a *symbolic branch-and-bound* algorithm adopting hybrid techniques. An undirected graph, composed of a set V of vertices and a set E of edges, can be represented by a characteristic function χ_E encoding the adjacency relation E, once vertices are given a Boolean encoding. We advocate the use of mixed symbolic and explicit algorithmic techniques in order to increase the feasibility of the solution.

This section aims at settling the general framework of the symbolic branch-and-bound that will be used for graph coloring: hybrid techniques are applied to a very naive text-book exhaustive algorithm.

Starting from this naive implementation, Section 3 introduces several heuristics in order to improve the performance and to broaden the applicability spectrum towards graphs typical of CAD problems.

The basic exhaustive algorithm is based on a recursive procedure, "search", sketched in Fig. 1. *Colors* contains all the possible colors that can be assigned to vertices: whenever the procedure is called, a color c is assigned to a vertex v of the graph.

```
search (set_of_vertices R, relation Coloring, relation
No_Coloring) {
  vertex v ; color c; set of vertices Adc;
  if R is empty /* no vertices left uncolored */
    save the best solution
  else
    extract v from R and delete it from R
    Adc = vertices of R not yet colored and adjacent to v
    for each color c of Colors not forbidden to v
      update Coloring with the assignment of c to v
      update No_Coloring by assigning c to Adc
      search (R, Coloring, No_Coloring)
      restore Coloring and No_Coloring  }
```

Figure 1: *Text-book Exhaustive Algorithm*

A *color* is defined as *forbidden* for a vertex if it has been assigned to at least one node adjacent to it. The procedure takes a set of vertices R to be colored, a relation $Coloring \subseteq V \times Colors$ storing the current color assigned to each vertex, and a relation *No_Coloring*. The relation $No\_Coloring \subseteq V \times Colors$ represents the colors that are *forbidden* to a vertex.

At each step, a vertex v is extracted from R and each color not forbidden to v is in turn assigned to it and the corresponding search subtree is recursively explored.

To improve the effectiveness, an approximate solution can be used as a starting point for the search. We use a greedy coloring algorithm implemented with hybrid techniques [CPSo95].

The following section develops, starting from the above skeleton, a symbolic branch-and-bound algorithm and some heuristics to reduce the computational complexity and to extend the applicability to large graphs.

3. The Algorithm

Starting from the straightforward formulation of Section 2, an efficient branch and bound algorithm has been developed by applying hybrid techniques. A set of heuristics allows to efficiently prune the search space without sacrificing the exactness of the solution. In the worst case, CPU time or memory occupation may limit the complete search space exploration: in this case the algorithm still provides an approximate solution, as well as tight upper and lower bounds.

The set of adopted heuristics is quite different from the ones adopted for traditional, explicit implementations, since the complexity of elementary operations is radically different. Manipulation of sets is relatively inexpensive when accomplished with BDDs. The developed heuristics are therefore composed of a sequence of set operations, defined on sets of vertices or colors.

Fig. 2 shows the pseudo code of the algorithm, completed with all the implemented heuristics. Due to space constraints, in the following of this Section we will shortly describe and motivate the different optimizations.

Optimization 1: Max. Clique as a lower bound

In addition to the upper bound computed by an approximate coloring procedure (Section 2), the preprocessing phase initially computes a lower bound on the chromatic number, given by the size of the maximum clique of the graph [CPSo95].

To avoid finding permutations of already found solutions, the vertices of the maximum clique are assigned arbitrary colors before the search procedure is activated.

Optimization 2: Pruning the search tree

The general principle driving our branch-and-bound implementation is to improve the current upper bound. If a solution is known employing *chromatic_number* colors, then the search procedure will only consider colorings having at most (*chromatic number*-1) colors.

To compute this, when a new solution is found, the set *Colors* is restricted (line 3) to a subset of the colors used by it. The search tree is then backtracked (line 12) until a partial solution with the updated number of available colors is found. The color to be excluded is chosen to minimize the number of backtrack steps.

Optimization 3: Choosing a good order of visit

As a general rule, the algorithm tries to do the best use of the colors already assigned to some vertex (C_{busy}) before using new ones (C_{free}). To implement a short look-ahead in this choice, it first computes C_{best}, the colors already forbidden to adjacent nodes (line 9): the choice of any of these colors poses no additional constraints and therefore is preferred. These subsets of *Colors* are easily computed and updated thanks to the underlying BDD engine (Tab. 1).

Optimization 4: Mandatory color assignments

Finally, some heuristics, based on *mandatory color assignments*, have been implemented in order to assign, as soon as possible, all the colors implied by current constrains. This avoids to uselessly visit most of the graph before discovering that the coloring can not be completed. At each step a vertex is extracted from *R'*, if it is not empty, otherwise from *R* (lines 5,6).

4. Experimental Results

The algorithms have been implemented by using the BDD package developed at our institution. The size or the tool amounts to less than 1,000 C source code lines, BDD package excluded. Experimental results have been run on a Sun SparcStation 20/50 with 64MB RAM, with a limit of 1,200,000 BDD nodes set. A time limit of 3 hours of CPU time has also been enforced.

Our purpose was to solve the coloring problem in graphs with the typical characteristics of problems in the area of VLSI, thus we chose to work on *realistic graphs*. Since no established benchmark graphs of large size are available for the graph coloring problem, for the sake of repeatability we worked on the State Transition Graphs (STGs) of standard benchmark circuits. In particular we worked on ISCAS'89 benchmarks [BBKo89] and some additional circuits we created (whose name starts with "fsm"). The adjacency relation χ_E among states has been extracted from the topological description of each circuit. The Fault Independence graphs [AbFr90] for the ISCAC'85 benchmarks c432 and c499 were also introduced in order to have some benchmark whose size was not a power of two.

The characteristics of the benchmark graphs are presented in Tab. 2. Note that BDDs are a compact data structure, orders of magnitude smaller than the explicit representation. This allowed us to work on graphs with millions of vertices.

A preliminary step (Tab. 3) computes the lower bound as the maximum clique and the upper bound as a greedy coloring; both the algorithms were presented in [CPSo95]. While the greedy coloring requires a negligible CPU time, the exact computation of the maximum clique takes less than 3 hours of CPU, except in s1196 and s1238. In these cases a large completely connected component was used as a lower bound, and the reported CPU time relates to the best result found.

```
search (set_of_vertices R, relation Coloring, relation
No_Coloring) {
      vertex  v; color c; set_of_vertices Adc, R';
      set_of_colors C_best, C_free, C_busy;
1:    if (R = ∅) {
2:     save the best solution;
3:     Colors = one color less than the best solution;
      } else {
4:     R' = {v ∈ R not colorable with colors(Coloring)};
5:     if (R' ≠ ∅) extract v from R';
6:     else extract v from R;
7:     Adc = { vertices adjacent to v and ∈ to R };
8:     compute C_busy, C_free, C_best; /* Tab. 1 */
9:     if (C_best ≠ ∅)  {
10:      C_busy = {choose a color c ∈ C_best};
11:      C_free = ∅ ; }
12:    while [(C_busy ≠ ∅) &&
         (colors(Coloring) ⊆ Colors)] {
13:      extract color c from C_busy and delete it;
14:      search (R - {v}, Coloring ∪ (v, c),
           No_Coloring ∪ (Adc, c)); }
15:    if [(C_free ≠ ∅) && (colors(Coloring) ⊂ Colors)] {
16:      extract color c from C_free;
17:      search (R - {v}, Coloring ∪ (v, c),
           No_Coloring ∪ (Adc, c)); } }
```

Figure 2: *The Algorithm*

C_{busy}	$\exists v\ \chi_{Coloring}(v,c) \cdot \neg\exists v\ (\chi_{No\_Coloring}(v,c) \cdot (\text{vertex } v))$
C_{free}	$\chi_{Colors}(c) \cdot \neg\ [\exists v\ \chi_{Coloring}\ (v, c)]$
C_{best}	$\chi_{Cbusy}(c) \cdot \neg\exists v\ [\neg\chi_{No\_Coloring}\ (v, c) \cdot \chi_{Adc}\ (v)]$

Table 1: *BDD expressions*

For some of the benchmarks (s208, s382, s400, s444, s953) the lower and upper bounds already coincide, and therefore the chromatic number is known a priori. This justifies the effort spent in the preprocessing phase, in particular to compute an exact maximum clique.

For the other benchmarks, the two bounds differ, and the search procedure (Fig. 2) is activated. The results of the minimum graph coloring algorithm, when it is able to complete the search and therefore give an exact solution, are shown in Tab. 4. The algorithm allowed us to

compute the chromatic number in large graphs with a very low CPU time. In fsm4b, fsm2b, and c499, although the exact coloring is not feasible within the given CPU and memory limits, the initial upper bounds are optimized and CPU times refer to the last improvement.

The algorithm is not applicable to larger benchmarks, as they require a number of BDD nodes in excess of the 1,200,000 limit. Differently from some other "pure BDD" implementation, hybrid techniques are able to give some result even in these cases, when the problem cannot be exactly solved with the resources at hand, by improving the tightness of the existing bounds. Tab. 5, in fact, reports data for all the benchmark circuits, showing the chromatic number, when known, or the range of its possible values. Also, an initial graph coloring is generated, with as many colors as the upper bound. In general, we are able to quickly find some quite accurate bounds, and in some cases to improve them and find the optimum solution.

In CAD applications it is extremely important to have this kind of graceful degradation, where al least tight bounds and an approximate solution are provided when the exact solution is too hard to find. Our results prove that the two bounds are actually very close.

5. Conclusions

This paper presents an algorithmic technique, called *hybrid technique*, based on integrating traditional Operational Research-style algorithms with BDD-based computations. To prove the effectiveness of the approach, a sample application has been chosen, namely exact graph coloring. Experimental results are shown that prove the applicability of the approach to problem instances commonly found in the CAD for VLSI area.

Hybrid techniques feature the advantages of both traditional algorithms (possibility of giving approximate solutions, easy introduction of heuristic optimizations) and BDDs (efficiency, ease of implementation of set computations).

The benchmark graphs we used, rather than being randomly generated, are representative of a class of problems encountered in the CAD for VLSI area.

The experimental results we gathered show that exact graph colorings can be obtained for some graphs with up to 2^{29} vertices (s953), by combining computation of tight lower and upper bounds and by optimizing the branch and bound algorithms with set computations. Moreover, when the exact solution can't be found, very often accurate estimates are provided: this kind of graceful degradation is seldom found in algorithms based on symbolic techniques.

We believe that this algorithmic approach can be fruitfully exploited in CAD tools to broaden their applicability, and at the same time the Operations Research community could profit from the introduction of these techniques. Our current work includes applying hybrid techniques to other graph problems in order to better validate their effectiveness.

		χ_E		
Circuit	**type**	**Vertices**	**Edges**	**BDDs**
s820, s832	STG	32	87	109
s386	STG	64	80	85
s510	STG	64	73	210
s1488, s1494	STG	64	125	230
s208	STG	256	509	109
s298	STG	16,384	81,403	943
s344, s349	STG	32,768	2,116,968	838
s420	STG	65,536	138,269	366
s1196	STG	262,144	355,119,753	240,214
s1238	STG	262,144	355,119,753	445,061
s641	STG	524,288	31,384,180	56,383
s713	STG	524,288	31,384,180	56,303
s382	STG	2,097,152	10,473,211	2,376
s400	STG	2,097,152	10,473,211	2,124
s444	STG	2,097,152	10,473,211	2,190
s526, s526n	STG	2,097,152	10,484,475	9,186
s953	STG	536,870,912	$> 2^{32}$	53,927
s838	STG	4,294,967,296	$> 2^{32}$	1,283
fsm6b	STG	16	38	36
fsm3b	STG	256	3,896	776
fsm4b	STG	256	3,269	420
fsm2b	STG	512	2,719	454
c432	FIG	519	4,044	6,490
c499	FIG	758	2,915	5,485

Table 2: *Graph characteristics (STG=State Transition Graphs; FIG=Fault Independence Graphs)*

6. References

[ABFr90] M. Abramovici, M. A. Breuer, A. D. Friedman: *Digital systems testing and testable design*, Computer Science Press, New York, NY (USA), 1990

[BBKo89] F. Brglez, D. Bryan, K. Kozminski: *Combinational profiles of sequential benchmark circuits*: IEEE Int. Symposium on Circuits And Systems, Portland (OR), USA, May 1989, pp. 1929-1934

[BCMD90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang: *Symbolic Model Checking: 10^{20} States and Beyond*, LICS'90: 5th Annual IEEE Symposium on Logic in Computer Science, June 1990, pp. 428–439

Circ.	Approx. No of Colors	Greedy Coloring time [s]	Max. Clique	Max. Clique time [s]
s820	6	0.1	5	0.1
s832	6	0.1	5	0.1
s386	4	0.1	3	0.1
s510	4	0.1	3	0.1
s1488	4	0.1	3	0.1
s1494	4	0.1	3	0.1
s208	3	0.1	3	0.1
s298	4	0.1	3	0.1
s344	258	0.7	257	7.9
s349	258	0.8	257	8.3
s420	5	0.1	3	2.3
s1196	1,069	158.0	≥ 1,034	56.9
s1238	1,061	5,981.2	≥ 1,046	3,727.2
s641	61	236.4	48	7,184.1
s713	61	226.6	48	7,086.4
s382	3	0.1	3	0.1
s400	3	0.1	3	0.1
s444	3	0.2	3	0.1
s526	6	3.3	3	19.8
s526n	6	3.3	3	19.7
s953	62	22.9	62	6.0
s838	7	0.5	3	605.2
fsm6b	5	0.1	4	0.1
fsm3b	15	0.1	10	0.1
fsm4b	11	0.1	7	0.1
fsm2b	9	0.1	7	0.1
c432	16	1.0	15	0.2
c499	12	0.7	8	0.5

Table 3: *Computing upper and lower bounds*

Circuit	Chromatic Number	Coloring time [s]
s820	5	0.5
s832	5	0.5
s386	4	0.5
s510	3	0.5
s1488	4	0.5
s1494	4	0.5
s298	3	59.3
fsm6b	4	0.5
fsm3b	10	163.7
fsm4b	≤9	2.6
fsm2b	≤8	1.6
c432	15	4.4
c499	≤10	515.2

Table 4: *Exact solutions*

[BRBr90] K. S. Brace, R. L. Rudell, R. E. Bryant: *Efficient Implementation of a BDD Package*, DAC'90: 27th ACM/IEEE Design Automation Conference, Orlando, FL (USA), June 1990, pp. 40–45

[Brya92] R. E. Bryant: *Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams*, ACM Computing Surveys, Vol. 24, Nr. 3, 1992, pp. 293-318

[CPSo93] F. Corno, P. Prinetto, M. Sonza Reorda: *Finding the Maximum Clique in a Graph Using BDDs*, ICVC'93: IEEE 3rd International Conference on VLSI and CAD, Taejon, Korea, November 1993, pp. 269-272

[CPSo95] F. Corno, P. Prinetto, M. Sonza Reorda: *Using Symbolic Techniques to find the Maximum Clique in Very Large Sparse Graphs*, ED&TC'95: IEEE European Design and Test Conference, Paris, France, March 1995, pp. 320-324

[Gibb85] A. Gibbons: *Algorithmic Graph Theory*, Cambridge University Press, Cambridge (MA), USA, 1985

Circuit	Chromatic Number	Total CPU time [s]
s820	5	0.6
s832	5	0.5
s386	4	0.5
s510	3	0.6
s1488	4	0.5
s1494	4	0.5
s208	3	0.1
s298	3	59.5
s344	≥257, ≤258	8.6
s349	≥257, ≤258	9.1
s420	≥3, ≤5	2.3
s1196	≥1,034, ≤1,069	215.0
s1238	≥1,046, ≤1,061	9,708.5
s641	≥48, ≤61	7,420.5
s713	≥48, ≤61	7,313.0
s382	3	0.1
s400	3	0.1
s444	3	0.3
s526	≥3, ≤6	23.2
s526n	≥3, ≤6	23.0
s953	62	29.0
s838	≥3, ≤7	605.7
fsm6b	4	0.5
fsm3b	10	164.1
fsm4b	≥7, ≤9	2.9
fsm2b	≥7, ≤8	1.8
c432	15	5.6
c499	≥8, ≤10	516.4

Table 5: *Combined results*