Design of Parallel Portfolios for SAT-Based Solving of Hamiltonian Cycle Problems

Miroslav N. Velev[‡] **Ping Gao**

Aries Design Automation, LLC

[‡]Contact author: miroslav.velev@aries-da.com

Abstract

We study portfolios of parallel strategies for Boolean Satisfiability (SAT) based solving of Hamiltonian Cycle Problems (HCPs). The strategies are based on our techniques for relative SAT encoding of permutations with constraints, and exploit: 1) encodings that eliminate half of the ordering Boolean variables and two-thirds of the transitivity constraints; 2) 12 triangulation heuristics for minimal enumeration of transitivity; 3) 11 heuristics for selecting the first node in the Hamiltonian cycle; 4) inverse transitivity constraints; and 5) exclusivity successor constraints between neighbors. We achieve up to 3 orders of magnitude speedup on random graphs that have Hamiltonian cycles and are in the phase transition region.

Introduction

We study portfolios of parallel strategies for Boolean Satisfiability (SAT) based solving of Hamiltonian Cycle Problems (HCPs). The strategies are based on our techniques for relative SAT encoding of permutations with constraints, using the relative SAT encoding (Prestwich 2003), thus exploiting the tremendous advances in the speed and capacity of SAT solvers without reimplementing the same optimizations in specialized tools for specific problems. Particularly, we do an in-depth study of HCPs—where the goal is to find a route in a graph by visiting each node exactly once and returning to the starting node—a known class of hard combinatorial problems, classified as NP-complete (p. 199 of (Garey and Johnson 1979)). Another hard combinatorial problem, quasigroup completion (Ansótegui et al. 2004; Gomes and Shmoys 2002; Kautz et al. 2001; Velev and Gao 2009a), can be reformulated as multiple permutations with constraints; it has applications to design of experiments, and wavelength routing in switches on optical networks. Other combinatorial problems that can be viewed as permutations with constraints are discussed in (Cadoli and Schaerf 2005; Hnich et al. 2004). At NASA, problems that can be reformulated as permutations of tasks, subject to additional constraints, arise in preparing sites for human habitation (Frank 2009). The efficient encoding of real-world problems as equivalent SAT formulas is a challenge identified by Kautz and Selman (2007).

The first method proposed for HCP solving by translation to SAT was based on the absolute SAT encoding of permutations (Iwama, and S. Miyazaki 1994), and was used in (Cadoli and Schaerf 2005; Hoos 1999; Prestwich 2003). However, those papers present results for graphs with at most 24 nodes. Recently, we did an in-depth study of techniques to improve the scalability of the absolute SAT encoding for permutations with constraints, when applied to solving HCPs (Velev and Gao 2009b). We found that the above method for absolute SAT encoding of permutations does not scale for a suite with 100 graphs of 30 nodes each, generated in the phase-transition region (Cheeseman et al. 1991; Frank and Martel 1995; Vandegriend and Culberson 1998). We proposed new techniques that improved the scalability of the absolute encoding, and resulted in at least 4 orders of magnitude average speedup when solving HCPs for both satisfiable and unsatisfiable benchmarks. However, the new techniques scaled only for graphs with up to 95 nodes.

In this paper, we study portfolios of parallel strategies for SAT-based solving of HCPs. The strategies use our techniques for relative SAT encoding of permutations with constraints, and exploit: 1) encodings that eliminate half of the ordering Boolean variables and two-thirds of the transitivity constraints; 2) 12 triangulation heuristics for minimal enumeration of transitivity; 3) 11 heuristics for selecting the first node in the Hamiltonian cycle; 4) inverse transitivity constraints; and 5) exclusivity successor constraints between neighbors. We achieve up to 3 orders of magnitude speedup on random graphs that have Hamiltonian cycles and are in the phase transition region.

Background

In the *relative encoding* (Prestwich 2003), represented are the relative positions of objects with respect to each other in a permutation. The motivation for this encoding is the observation that in the absolute encoding of permutations, if node v is to be moved forward (backward) in a permutation, then all nodes between the old and the new positions of v will also have to be moved, which will result in value changes for the Boolean variables encoding the placement of all those nodes. In contrast, in the relative encoding, we only need to change the relative position of node v with respect to the rest of the nodes by changing the values of some or all of the Boolean variables used for node v, while

keeping the rest of the nodes in their original relative positions, i.e., keeping the values assigned to the Boolean variables encoding the relative placement of those nodes, thus significantly reducing the number of Boolean variables whose values have to be changed, and so significantly reducing the corresponding work of the SAT solver. Since Prestwich presented constraints for Hamiltonian paths, while our focus is on Hamiltonian cycles that require additional constraints to ensure that the first and last nodes in the permutation are neighbors in the graph, we present next an extension of his formulation for Hamiltonian cycles.

Two types of Boolean variables are introduced—successor and ordering variables. The *successor variables* encode the constraints that exactly one out of a node's neighbors is selected to be its successor in the permutation, and exactly one is selected to be its predecessor, based on the direct encoding (de Kleer 1989). A successor Boolean variable s_{ij} is introduced for every ordered pair of nodes (v_i, v_j) that are neighbors in the graph, i.e., 2 such variables are required for each edge—one variable for each of the two orderings of the two nodes of the edge—such that s_{ij} is true iff node v_j appears immediately after v_i in the permutation that represents the ordering of the nodes in the Hamiltonian cycle. The constraints for the successor variables are:

- each node v_i has at least one successor from its neighbors: $s_{ij} \vee s_{ik} \vee \ldots$, where the neighbors of v_i are nodes v_j, v_k, ... ;
- each node v_i has at most one successor from its neighbors: $\neg s_{ij} \vee \neg s_{ik}$, for every pair of neighbors v_j and v_k of node v_i;
- each node v_i is the successor of at least one of its neighbors: $s_{ji} \vee s_{ki} \vee \ldots$, where the neighbors of v_i are nodes v_j, v_k, ... ; and
- each node v_i is the successor of at most one of its neighbors: $\neg s_{ji} \vee \neg s_{ki}$, for every pair of neighbors v_j and v_k of node v_i.

The *ordering variables* encode the relative ordering of the nodes in the permutation that represents the Hamiltonian cycle, starting from the node selected to be first, v_f, and ending with the first node's neighbor selected to be the last node in the cycle, v_l. An ordering Boolean variable o_{ij} is introduced only for ordered pairs of two different nodes (v_i, v_j), where $i \neq j$. The ordering variable o_{ij} is defined to be true iff node v_i appears before node v_j in the permutation. The ordering variables satisfy several properties:

- *transitivity*: $o_{ij} \wedge o_{jk} \Rightarrow o_{ik}$, i.e., $\neg o_{ij} \vee \neg o_{jk} \vee o_{ik}$, for all permutations of 3 nodes v_i, v_j, and v_k, such that $i \neq j, j \neq k$, and $i \neq k$;
- *antisymmetry*: $\neg(o_{ij} \wedge o_{ji})$, or equivalently $\neg o_{ij} \vee \neg o_{ji}$, i.e., it is impossible for node v_i to be before node v_j and for node v_j to be before node v_i simultaneously;
- the first node v_f precedes all others: o_{fi}, for all $i \neq f$; and
- the first node's neighbor v_l selected to be the last node in the cycle succeeds all others: $s_{lf} \Rightarrow o_{il}$, or equivalently $\neg s_{lf} \vee o_{il}$, for all $i \neq f$, $i \neq l$, and all neighbors v_l of the first node v_f.

Finally, the relationship between successor and ordering variables is: $s_{ij} \Rightarrow o_{ij}$, or equivalently $\neg s_{ij} \vee o_{ij}$, i.e., if node v_i has its neighbor v_j selected to be v_i's successor along the Hamiltonian cycle, then the ordering variable o_{ij} is true.

Prestwich enforced transitivity for all possible permutations of three nodes. However, this results in $n \cdot (n-1)$ ordering variables and $n \cdot (n-1) \cdot (n-2)$ transitivity constraints for a graphs with n nodes, i.e., in a prohibitive number of transitivity constraints for large graphs. We refer to this method as *exhaustive enumeration of transitivity*.

Eliminating Half of the Ordering Variables and Two-Thirds of the Transitivity Constraints

In our recent work (Velev and Gao 2009c), we observed that the ordering variables o_{ij} and o_{ji} are complements of each other, since when node v_i precedes node v_j then o_{ij} is true and o_{ji} is false by definition, and vice versa—when node v_j precedes node v_i then o_{ij} is false and o_{ji} is true by definition. Thus, we can introduce an ordering variable o_{ij} only if the node index i is less than the node index j, and replace the ordering variable o_{ji} with $\neg o_{ij}$. Hence:

LEMMA 1. *For a triple of nodes* $\{v_i, v_j, v_k\}$*, whose indices are ordered* $1 \leq i < j < k \leq n$*, where n is the number of nodes in the graph, after half of the ordering variables are eliminated by introducing an ordering variable* o_{pq} *only for an ordered pair of nodes* (v_p, v_q) *where the node indices satisfy* $1 \leq p < q \leq n$*, and replacing the ordering variable* o_{qp} *with* $\neg o_{pq}$*, then the six transitivity constraints that are introduced for the triple of nodes* $\{v_i, v_j, v_k\}$ *in exhaustive enumeration of transitivity will reduce to two transitivity constraints:* (1) $\neg o_{ij} \vee \neg o_{jk} \vee o_{ik}$*; and* (2) $o_{ij} \vee o_{jk} \vee \neg o_{ik}$.

Thus, half of the ordering variables, two-thirds of the transitivity constraints, and all the antisymmetry constraints (that evaluate to true now) are eliminated. This results in fewer decisions to be made by the SAT solver when solving the resulting CNF formula, as well as reduces significantly the *Boolean Constraint Propagation* (BCP)—the iterative process of assigning values to literals as implied by clauses that have become unit, i.e., where all literals but one have assigned values, and those values are false, so that the only literal without a value has to be set to true in order for the clause to be satisfied. BCP takes up to 90% of the execution time of SAT solvers (Moskewicz et al. 2001).

Minimal Enumeration of Transitivity

Exhaustive transitivity requires $n \cdot (n-1)$ ordering variables and $n \cdot (n-1) \cdot (n-2)$ transitivity constraints, where n is the number of nodes in the graph. Eliminating half of the ordering variables and applying Lemma 1 results in $n \cdot (n-1) / 2$ ordering variables and $n \cdot (n-1) \cdot (n-2) / 3$ transitivity constraints. In this section we aim to further reduce the number of ordering variables and transitivity constraints by introducing only a minimal set of such variables and constraints in a way that the property of transitivity will be enforced for the *required ordering variables*—those that are used in the constraints for the relative encoding, except for the transitivity constraints.

We introduced the concept of a *relational graph* that has the same set of nodes as the original graph that we want to find a Hamiltonian cycle in, but an extended set of edges, such that a pair of nodes is connected with an edge in the relational graph iff an ordering variable is introduced for that pair of nodes. We will use the relational graph to efficiently enforce the property of transitivity for the required ordering variables. These variables determine three types of edges that must be present in the relational graph:

1. each edge from the original graph, since the relationship between successor and ordering variables requires that each edge in the original graph should have an associated ordering variable;
2. given a choice for a first node, v_f, an edge between v_f and each of the other $n - 1$ nodes in the relational graph, because of the constraints that v_f must precede all other nodes in the permutation; and
3. given a choice for a first node, v_f, an edge between each of v_f's neighbors and the other $n - 2$ nodes in the relational graph besides v_f, because of the conditions that when that neighbor is selected to be the last node in the permutation, then the other nodes must precede that neighbor in the permutation (that v_f precedes the last node is already ensured with constraints that resulted in edges of type 2 above).

We refer to the resulting graph as *base relational graph*, and designate it R_B. Because of the second and third types of edges that depend on the choice of the first node, v_f, the base relational graph must be constructed after the first node in the Hamiltonian cycle is selected. Note that the base relational graph R_B is different from the original graph for which we are searching for a Hamiltonian cycle, and is used for an efficient relative SAT encoding of the node permutation.

Besides the three types of required edges in R_B that were described, we want to introduce additional edges such that the resulting graph becomes *chordal* (Rose 1970). That is, the graph has the property that every cycle of length greater than three has a chord—an edge joining two nodes that are not adjacent in the cycle. Chordal graphs are also called *triangulated graphs*. We can then generate a sufficient set of transitivity constraints by enforcing transitivity for each triangle in the resulting graph. Computing a minimal triangulation consists of embedding a given graph into a triangulated graph by adding a set of edges called *a fill*. Finding a fill that is minimum is NP-complete (Yanakakis 1981). However, there are good heuristic solutions. The ones that we explored are described next.

Our triangulation method proceeds as a series of elimination steps, starting with graph $G_0 = R_B$. On elimination step i, we create a new graph G_i that is identical to G_{i-1}, except that some vertex v_m and its incident edges are removed, and new edges are possibly added in the following way: for every pair of distinct vertices v_j and v_k such that G_{i-1} contains the edges (v_m, v_j) and (v_m, v_k), we add the edge (v_j, v_k) to G_i if this edge does not already exist. This process continues until we reach an empty graph G_n. Let R_T be the relational graph obtained from R_B after extending it with all edges added during the elimination process. It can be shown that R_T is a chordal graph (Rose 1970). To choose which node v_m to eliminate on step i, we implemented the following 12 triangulation heuristics that select the node with the current minimum degree, and if there are several nodes with such degree, the tie is broken as follows: *t1*—break the tie based on a lesser sum of neighbors' degrees; *t2*—break the tie based on a greater sum of neighbors' degrees; *t3*—break the tie based on a lesser fill (i.e., additional edges) that will be added at this step of the elimination; *t4*—break the tie based on a greater fill that will be added at this step of the elimination; *t5*—break the tie based on a lesser original degree of the node in R_B; *t6*—break the tie based on a greater original degree of the node in R_B; *t7*—break the tie based on a lesser number of additional triangles created in R_T that include node v_m, and edges from G_{i-1} and those added at step i; *t8*—break the tie based on a greater number of additional triangles created in R_T that include node v_m, and edges from G_{i-1} and those added at step i; *t9*—select the node that will result in a minimum fill at this step of the elimination, and break ties by selecting the first such node in the graph description; *t10*—select the node that will result in a minimum fill at this step of the elimination, and break ties randomly; *t11*—select the node that will result in a minimal number of additional triangles created in R_T that include node v_m, and edges from G_{i-1} and those added at step i, and break ties by selecting the first such node in the graph description; and *t12*—select the node that will result in a minimal number of additional triangles created in R_T that include node v_m, and edges from G_{i-1} and those added at step i, and break ties randomly.

We use the triangles in R_T to enforce transitivity of the ordering variables. When also eliminating half of the ordering variables, we only introduce one ordering variable per edge in R_T and the two transitivity constraints per triangle—see Lemma 1.

Triangulation heuristic *t10* was also explored by Kjaerulff (1990), who found it to have the best performance when triangulating belief networks in knowledge representation. He called that heuristic minimum fill. Rose (1973) refers to it as minimum deficiency heuristic, and references other papers (Sato and Tinney 1963; Tinney and Walker 1967) in which it is assumed that this heuristic produces a near optimal node-elimination ordering.

Selecting the First Node for the Hamiltonian Cycle

We exploit the observation that when the search is for a Hamiltonian cycle, as opposed to a Hamiltonian path, the first node in the cycle (i.e., permutation) can be selected in any way, since all nodes have to be included in the cycle, and if a cycle exists then all nodes are symmetrical in it. Thus, if a cycle does not exist when a particular node is selected as a first node, then a cycle would not exist when any other node is selected as a first node. That is, the first node can be chosen in any way because of the symmetry of a solution, and the general formulation of the relative encoding.

We implemented the following 11 heuristics for selecting the first node: *f1*—choose the first node in the graph description; *f2*—choose the first node of max. degree in the graph description; *f3*—choose the first node of min. degree in the graph description; *f4*—choose the first node of average degree in the graph description; *f5*—random choice; *f6*—choose a node of max. degree, and break ties based on a lesser sum of neighbors' degrees; *f7*—choose a node of max. degree, and break ties based on a greater sum of neighbors' degrees; *f8*—choose a node of average degree, and break ties based on a lesser sum of neighbors' degrees; *f9*—choose a node of average degree, and break ties based on a greater sum of neighbors' degrees; *f10*—choose a node of min. degree, and break ties based on a lesser sum of neighbors' degrees; and *f11*—choose a node of min. degree, and break ties based on a greater sum of neighbors' degrees. In contrast, Prestwich (2003) selected the first node for a Hamiltonian path (he was searching for Hamiltonian paths, as opposed to cycles) to be the first node in the graph description.

Inverse Transitivity Constraints

We consider triples of nodes v_i, v_j, and v_k, such that v_i and v_j are neighbors in the original graph. Then, if node v_j is selected to be the successor of node v_i in the permutation, as indicated by the successor Boolean variable s_{ij} being true, and node v_i precedes node v_k in the permutation, as indicated by the ordering Boolean variable o_{ik} being true, it follows that node v_j must also precede node v_k in the permutation, i.e., o_{jk} must be true, since v_j is the closest node that is after v_i and thus v_k must be after v_j, i.e., $s_{ij} \wedge o_{ik} \Rightarrow o_{jk}$, or equivalently $\neg s_{ij} \vee \neg o_{ik} \vee o_{jk}$ (see Fig. 1.a). We call these constraints *forward inverse transitivity constraints*. Note that they are different from the regular transitivity constraints that state that if v_i is before v_j, and v_j is before v_k, then v_i is before v_k.

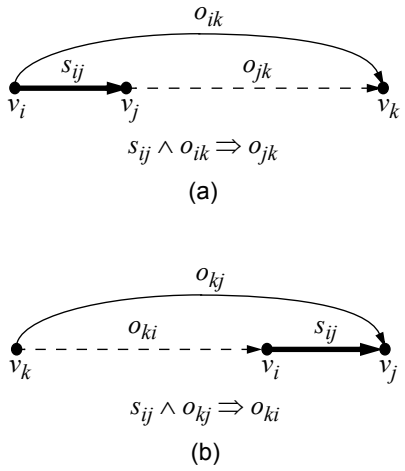

Figure 1. Illustration of: (a) forward inverse transitivity; (b) backward inverse transitivity.

We can also define a variant of these constraints for the case when node v_k is before node v_j, as indicated by the ordering Boolean variable o_{kj} being true, such that node v_j is selected to be the successor of node v_i in the permutation, as indicated by the successor Boolean variable s_{ij} being true. Then, it follows that node v_k must be before v_i, i.e., o_{ki} must be true, since v_i is the closest node that is before v_j by being the predecessor of v_j and thus v_k must be before v_i, i.e., $s_{ij} \wedge o_{kj} \Rightarrow o_{ki}$, or equivalently $\neg s_{ij} \vee \neg o_{kj} \vee o_{ki}$ (see Fig. 1.b). We call these constraints *backward inverse transitivity constraints.*

When we eliminate half of the ordering variables, as discussed earlier, only two of the above four ordering variables will actually be introduced, while the other two will be each replaced with the negation of the introduced ordering variable for the same pair of nodes. In our tool, we imposed the inverse transitivity constraints for every ordered pair of neighboring nodes (v_i, v_j) from the original graph, and for every node v_k that is different from v_i and v_j, such that the ordering variables for v_k with respect to both v_i and v_j have been introduced by corresponding edges in the triangulated relational graph R_T. Note that these constraints are optional, and that we can use only a subset of them.

Exclusivity Successor Constraints Between Neighbors

We exploit the observation that given a pair of nodes v_i and v_j that are neighbors in the original graph, if node v_j is selected to be the successor of node v_i in the permutation, as indicated by the successor Boolean variable s_{ij} being true, then node v_i cannot be a successor of node v_j, since v_i is already a predecessor of v_j as determined by s_{ij} being true, i.e., the successor variable s_{ji} should be false, and vice versa. Thus, for a pair of neighboring nodes v_i and v_j, the corresponding successor Boolean variables should not be true simultaneously: $\neg s_{ij} \vee \neg s_{ji}$. Without this constraint, the SAT solver could waste work in exploring an infeasible portion of the solution space. Note that these constraints are optional, and that we can use only a subset of them.

Summary of Results

Given the many heuristics for the various choices in translating HCPs to SAT, we can use different combinations of parameters to form different strategies that will be used in parallel portfolios, where the rest of the strategies will be stopped as soon as one of them returns a solution. Our experiments for graphs with up to 550 nodes, generated in the phase transition region, resulted in speedup of up to 3 orders of magnitude on individual benchmarks, for which the best individual strategy (as measured by the total time that it takes for an entire suite of 100 benchmarks, each with the given number of nodes) had taken several thousand seconds, while a strategy that was ranked among the top four found a solution in a couple of seconds, thus achieving a speedup of 3 orders of magnitude on individual benchmarks. Increasing the number of parallel strategies reduces the variation of the run times for individual benchmarks, although the major speedup was achieved with a portfolio of only 4 parallel strategies.

Conclusion

We studied portfolios of parallel strategies for SAT-based solving of Hamiltonian Cycle Problems (HCPs). The strategies were based on our techniques for relative SAT encoding of permutations with constraints, and exploited: 1) encodings that eliminate half of the ordering Boolean variables and two-thirds of the transitivity constraints; 2) 12 triangulation heuristics for minimal enumeration of transitivity; 3) 11 heuristics for selecting the first node in the Hamiltonian cycle; 4) inverse transitivity constraints; and 5) exclusivity successor constraints between neighbors. We achieved up to 3 orders of magnitude speedup on random graphs that have Hamiltonian cycles and are in the phase transition region.

References

Ansótegui, C.; del Val, A.; Dotú, I.; Fernández, C.; and Manyà, F. 2004. Modeling Choices in Quasigroup Completion: SAT vs. CSP. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, 137–142.

Cadoli, M.; and Schaerf, A. 2005. Compiling Problem Specifications into SAT. *Artificial Intelligence* 162(1–2): 89–120.

Cheeseman, P.; Kanefsky, B.; and Taylor, W. M. 1991. Where the Really Hard Problems Are. In *Proceedings of the 12th International Joint Conference on AI (IJCAI'91)*, 331–337.

Crawford, J.M.; and Baker, A.B. 1994. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, 1092–1097.

de Kleer, J. 1989. A Comparison of ATMS and CSP Techniques. In *Proceedings of the 11th Int'l. Joint Conference on Artificial Intelligence (IJCAI'89)*, 290–296.

Eén, N.; and Sörensson, N. 2005. MiniSat—A SAT Solver with Conflict-Clause Minimization. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing*.

Frank, J.; and Martel, C. U. 1995. Phase Transitions in the Properties of Random Graphs. In *Proceedings of Principles and Practice of Constraint Programming (CP'95)*.

Frank, J. 2009. Personal Communication.

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

Goldberg, E.; and Novikov, Y. 2002. BerkMin: A Fast and Robust Sat-Solver. In *Proceedings of Design, Automation, and Test in Europe (DATE'02)*, 142–149.

Gomes, C. P.; and Shmoys, D. B. 2002. The Promise of LP to Boost CSP Techniques for Combinatioral Problems. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'02)*, 291–305.

Hnich, B.; Walsh, T.; and Smith, B. M. 2004. Dual Modelling of Permutation and Injection Problems. *Journal of Artificial Intelligence Research (JAIR)* 21: 357–391.

Hoos, H. H. 1999. SAT-Encodings, Search Space Structure, and Local Search Performance. In *Proceedings of the 16th Int'l. Joint Conference on Artificial Intelligence (IJCAI'99)*, 296–303.

Huang, J. 2007a. The Effect of Restarts on the Efficiency of Clause Learning. In *Proceedings of the 20th Int'l. Joint Conference on Artificial Intelligence (IJCAI'07)*, 2318–2323.

Huang, J. 2007b. A Case for Simple SAT Solvers. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, 839–846.

Iwama, K.; and Miyazaki, S. 1994. SAT-Varible Complexity of Hard Combinatorial Problems. In *Proceedings of the IFIP 13th World Computer Congress* (1): 253–258.

Kautz, H. A.; Ruan, Y.; Achlioptas, D.; Gomes, C. P.; Selman, B.; and Stickel, M. E. 2001. Balance and Filtering in Structured Satisfiable Problems. In *Proceedings of the 17th Int'l. Joint Conference on Artificial Intelligence (IJCAI'01)*, 351–358.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as Satisfiability. *5th Int'l Planning Competition, Int'l Conf. on Automated Planning and Scheduling (ICAPS'06)*.

Kautz, H. A.; and Selman, B. 2007. The State of SAT. *Discrete Applied Mathematics* 155(12): 1514–1524.

Kjaerulff, U. 1990. Triangulation of Graphs—Algorithms Giving Small Total State Space. Technical Report R 90-09, Institute for Electronic Systems, Denmark.

Moskewicz, M.W.; Madigan, C.F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an Efficient SAT Solver. *38th Design Automation Conference (DAC'01)*, 530–535.

Pipatsrisawat, K.; and Darwiche, A. 2007. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, 294–299.

Pipatsrisawat, K.; and Darwiche, A. 2008. A New Clause Learning Scheme for Efficient Unsatisfiability Proofs. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 1481–1484.

Prestwich, S. D. 2003. SAT Problems with Chains of Dependent Variables. *Discrete Applied Mathematics* 130(2): 329–350.

Rose, D. 1970. Triangulated Graphs and the Elimination Process. *Journal of Mathematical Analysis and Applications* 32, 597–609.

Rose, D. 1973. A Graph-Theoretic Study of the Numerical Solution of Sparse Positive Definite Systems of Linear Equations. In *Graph Theory and Computing*, R.C. Read, *ed.*, Academic Press, N.Y., 183–217.

Ryan, L. Siege SAT Solver v.4, http://www.cs.sfu.ca/~loryan/personal/.

Sato, N.; and Tinney, W.F. 1963. Techniques for Exploiting the Sparsity or the Network Admittance Matrix. *IEEE Transactions on Power Apparatus and Systems* 82 (69), 944–950.

Smith, S.F.; and Cheng, C.-C. 1993. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings of 11th National Conference on Artificial Intelligence (AAAI'93)*, 139–144.

Tinney, W.F.; and Walker, J.W. 1967. Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization. *Proceedings of the IEEE* 55 (11), 1801–1809.

Vandegriend, B.; and Culberson, J. 1998. The $G_{n,m}$ Phase Transition Is Not Hard for the Hamiltonian Cycle Problem.

Journal of Artificial Intelligence Research 9, 219–245.

Velev, M. N. 2007. Exploiting Hierarchy and Structure to Efficiently Solve Graph Coloring as SAT. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'07)*, 135–142.

Velev, M. N.; and Gao, P. 2008. Comparison of Boolean Satisfiability Encodings on FPGA Detailed Routing Problems. In *Proceedings of Design, Automation and Test in Europe (DATE '08)*, 1268–1273.

Velev, M. N.; and Gao, P. 2009a. Efficient SAT-Based Techniques for Design of Experiments by Using Static Variable Ordering. In *Proceedings of 10th International Symposium on Quality Electronic Design (ISQED '09)*, 371–376.

Velev, M.N.; and Gao, P. 2009b. Efficient SAT Techniques for Absolute Encoding of Permutation Problems: Application to Hamiltonian Cycles. In *Proceedings of the 8th Symposium on Abstraction, Reformulation and Approximation (SARA'09)*.

Velev, M.N.; and Gao, P. 2009c. Efficient SAT Techniques for Relative Encoding of Permutations with Constraints. In *Proceedings of the 22nd Australasian Joint Conference on Artificial Intelligence (AI'09)*, A. Nicholson, and X. Li, *eds.*, LNAI 5866, Springer-Verlag, 517–527.

Xing, Z.; Chen, Y.; and Zhang, W. 2006. MaxPlan: Optimal Planning by Decomposed Satisfiability and Backward Reduction. *5th Int'l Planning Competition, Int'l Conf. on Automated Planning and Scheduling (ICAPS'06)*, 53–56.

Yanakakis, M. 1981. Computing the Minimum Fill-in Is NP-Complete. *SIAM Journal of Algebraic and Discrete Mathematics* 2, 77–79.