Improving Combinatorial Optimization: Extended Abstract*

Geoffrey Chu
Department of Computing and Information Systems,
University of Melbourne, Australia
gchu@csse.unimelb.edu.au

Abstract

Combinatorial Optimization is an important area of computer science that has many theoretical and practical applications. In the thesis [Chu, 2011], we present important contributions to several different areas of combinatorial optimization, including nogood learning, symmetry breaking, dominance, relaxations and parallelization. We develop a new nogood learning technique based on constraint projection that allows us to exploit subproblem dominances that arise when two different search paths lead to subproblems which are identical on the remaining unfixed variables. We present a new symmetry breaking technique called SBDS-1UIP, which extends Symmetry Breaking During Search (SBDS) by using the more powerful 1UIP nogoods generated by Lazy Clause Generation (LCG) solvers. We present two new general methods for exploiting almost symmetries by modifying SBDS-1UIP and by using conditional symmetry breaking constraints. We solve the Minimization of Open Stacks Problem, the Talent Scheduling Problem (CSPLib prob039), and the Maximum Density Still Life Problem (CSPLib prob032) many orders of magnitude faster than the previous state of the art by applying various powerful techniques such as nogood learning, dynamic programming, dominance and relaxations. We present cache aware data structures for SAT solvers which allows sequential and parallel versions of SAT solvers to run more quickly. And we present a new load balancing scheme for parallel search called *confidence based work stealing*, which allows the parallel search to make use of the information contained in the branching heuristic.

1 Introduction

Combinatorial Optimization is an important area of computer science that has many theoretical and practical applications. Constraint Programming (CP) is a relatively new, and very effective technology for solving combinatorial optimization problems. In the thesis [Chu, 2011], we present important contributions to a number of areas in CP, including nogood learning, symmetry breaking, dominance, relaxations, and parallelization. The thesis is divided into four parts, each addressing a different area of CP.

*The thesis on which this extended abstract is based was the recipient of the 2012 ACP Doctoral Research Award [Chu, 2011]

2 Background and Definitions

Let $\equiv$ denote syntactical identity, $\Rightarrow$ denote logical implication and $\Leftrightarrow$ denote logical equivalence. A *Constraint Satisfaction Problem* (CSP) is a tuple $P \equiv (V, D, C)$, where V is a set of variables, D is a domain over V (i.e., a set of unary constraints), and C is a set of n-ary constraints. A assignment θ over V is a *solution* of P if it satisfies every constraint in D and C. The aim of a CSP is to find a solution or to prove that none exist. In a *Constraint Optimization Problem* (COP) $P \equiv (V, D, C, f)$, we also have an objective function f mapping assignments to an ordered set, e.g., $\mathbb{Z}$ or $\mathbb{R}$, and we wish to minimize or maximize f over the solutions of P.

A propagator p_c for constraint c is a function mapping domains to domains s.t. $c \wedge D \Leftrightarrow c \wedge p_c(D)$, i.e., it performs inference based on c and the current domain D, pruning away any variable/value pairs that cannot be taken given c and D. A propagation based CP solver solves a CSP by interleaving search with inference. We begin with the original problem at the root of the search tree. At each node in the search tree, we propagate all the propagators until the domain reaches a fixed point. If some variable's domain becomes empty, then the problem has no solution and the solver backtracks. If all the variables are assigned and no constraint is violated, then a solution has been found and the solver can terminate. If inference is unable to detect either of the above two cases, then the solver divides the problem into a number of more constrained subproblems and searches each of those in turn.

3 Part 1 - Nogood Learning

Nogood learning techniques are used to speedup constraint solving by learning implied constraints called *nogoods*. A nogood is an implied constraint of form: $\neg(l_1 \wedge \ldots \wedge l_n)$, expressing a set of conditions under which there can be no solutions to the problem. They can be propagated to prune parts of the search space. There are various ways to derive nogoods. The simplest way is to derive them through exhaustive search. Whenever a subtree reached via a set of

decisions has been exhaustively searched and has no solutions, the set of decisions we took to get to the subtree forms a nogood, which we call the decision nogood. Such decision nogoods are easy to derive but are also very weak in terms of pruning strength. It has been suggested that such nogoods can be used in CP solvers that perform restarts in order to prevent them from exploring previously explored parts of the search tree [Lecoutre *et al.*, 2007]. Another way to derive nogoods is via constraint resolution. Such nogoods are used in SAT solvers [Silva and Sakallah, 1999], and in Lazy Clause Generation solvers [Ohrimenko *et al.*, 2007; Feydy and Stuckey, 2009]. Whenever the solver finds a conflict, it is possible to find the subset of constraints responsible and to resolve them together to derive a new nogood. Such a nogood expresses the reason for the conflict, and can prevent the solver from making assignments that fail for the same reason again. They are very powerful and can provide orders of magnitude speedup on a wide range of problems.

In Part 1 of the thesis, we present a new nogood learning technique based on constraint projection. In many combinatorial optimization problems, different search paths lead to subproblems which are actually equivalent on the remaining unfixed variables. Naive CP solvers cannot recognize this and will therefore end up solving these equivalent subproblems again and again. We can prevent this by deriving nogoods which express the failure of such subproblems so that after it is examined once, equivalent subproblems will not be searched again. Such nogoods can be derived using *constraint projection*. Suppose that all subtrees of a particular search node N have been searched and has failed. Let $F \subseteq V$ be the set of variables which were fixed at the propagation fixed point of N. We calculate the projection of each constraint and domain onto the remaining unfixed variables $V \setminus F$ given the values of the fixed variables at N. The conjunction of these projections then form a valid nogood.

To be precise, given a set of variables $V \equiv \{v_1, \ldots, v_n\}$ and an arbitrary logical expression c, let $\exists_V c \equiv \exists_{v_1} ... \exists_{v_n} c$. $\exists_V$ is a projection operator which projects a logical expression onto the variables not in V. let D_F be the domains of the variables in F at N. Then $n \equiv \neg(\wedge_{c \in C \cup D} \exists_F (D_F \wedge c))$ is a nogood which expresses that the node N fails because of the effect that the current assignment on F has on the unfixed variables. This nogood can be used to prune other subtrees. If a different assignment to F gives the same projection onto the unfixed variables, then they actually lead to an equivalent subproblem as N and thus are also guaranteed to fail. We can detect this by storing these nogoods in a hash table. If another node generates a set of projections that match one of the nogoods currently in the hash table, then we know that we have examined an equivalent subproblem before and can prune this one. Note that these projections are quite easy to calculate, as we simply need to project the constraints in a piecewise manner and take their conjunction. On appropriate problems, our new nogood learning technique provides orders of magnitude speedup compared to a base solver which does not learn nogoods. Our technique is competitive with Lazy Clause Generation, which is the current state of the art in nogood learning. The nogoods derived through constraint projection are different in power to the nogoods derived by Lazy Clause Generation through constraint resolution, and can be exponentially stronger on some problems. More details on the proofs and implementation can be found in [Chu *et al.*, 2010] and [Chu *et al.*, 2012].

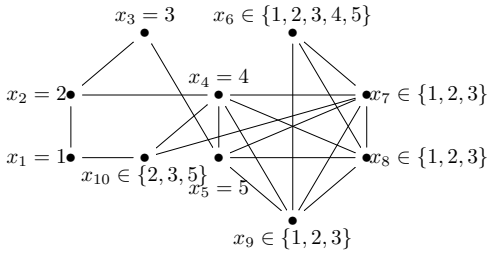

Figure 1: A graph coloring problem where we can exploit additional symmetries.

4 Part 2 - Symmetry Breaking

Symmetry breaking techniques are used to speedup constraint solving in problems with symmetries. In such problems, examining one assignment is often enough to tell us everything we need to know about its symmetric counterparts (whether it is a solution, what its objective value is, etc). These redundancies in the search can exponentially increase the run time. One way to prevent this is to apply some form of symmetry breaking to prevent the CP solver from exploring symmetric parts of the search space. The most popular symmetry breaking techniques are lexicographical symmetry breaking constraints [Puget, 1993; Crawford *et al.*, 1996; Luks and Roy, 2004; Puget, 2005; Frisch *et al.*, 2003], Symmetry Breaking During Search (SBDS) [Gent and Smith, 2000], and Symmetry Breaking by Dominance Detection (SBDD) [Fahle *et al.*, 2001]. Lexicographical symmetry breaking constraints can be statically added to the problem to constrain the solver to look only for solutions which are lexicographically least within its equivalence class. SBDS and SBDD are dynamic symmetry breaking techniques which, after each partial assignment has been explored, adds constraints to disallow all of its symmetric counterparts from being solutions.

In Part 2 of the thesis, we present a new symmetry breaking technique called SBDS-1UIP, which is an extension of SBDS. Given a symmetry σ, and any nogood $n \equiv \neg(l_1 \wedge \ldots \wedge l_n)$, where l_i are literals of the problem of the form $x = v$ or $x \neq v$ for some variable x and value v, $\sigma(n) \equiv \neg(\sigma(l_1) \wedge \ldots \wedge \sigma(l_n))$ is also a valid nogood. The original SBDS technique can be thought of as using symmetric versions of decision nogoods to prune symmetric parts of the search space. In our extension SBDS-1UIP, we use symmetric versions of the 1UIP nogoods derived by Lazy Clause Generation solvers instead. We show that SBDS-1UIP can exploit at least as many symmetries as SBDS, and that it is strictly more powerful on some problems.

Example 1 Consider the graph coloring problem shown in Figure 1 where we are trying to 5-colour the nodes so that no pair of adjacent nodes having the same color. let $x_1, \ldots, x_{10}$ represent the colours of each node. Clearly, all 5 colors are interchangeable so we have a value symmetry. Suppose we have made decisions $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5$.

Propagation produces the domains shown in Figure 1. Suppose we decide to try $x_6 = 1$ next. The search eventually fails since we have a 3-clique on x_7, x_8, x_9 but only 2 colours available. The 1UIP nogood is $x_7 \neq 4 \wedge x_7 \neq 5 \wedge x_8 \neq 4 \wedge x_8 \neq 5 \wedge x_9 \neq 4 \wedge x_9 \neq 5 \rightarrow x_6 \neq 1$, which tells us we should prune $x_6 = 1$. However, we can apply the value symmetries between 1 and 2 to get a symmetric nogood: $x_7 \neq 4 \wedge x_7 \neq 5 \wedge x_8 \neq 4 \wedge x_8 \neq 5 \wedge x_9 \neq 4 \wedge x_9 \neq 5 \rightarrow x_6 \neq 2$, allowing us to prune $x_6 = 2$ as well. Similarly, we can apply the value symmetries between 1 and 3 to get a nogood that allows us to prune $x_6 = 3$. None of the traditional methods such as SBDS, SBDD or lex-leader symmetry breaking constraints are capable of pruning $x_6 = 2$ or $x_6 = 3$, as all 5 values had previously already been used in $x_1, \ldots, x_5$.

This is a very interesting result, as SBDS is already a complete method which guarantees that at most one solution in each equivalence class is found by the solver. However, SBDS-1UIP can be strictly more powerful than that, as it is capable of taking advantage of the information about which subsets of decisions and constraints actually caused a conflict in order to do even more symmetry breaking. On some problems, SBDS-1UIP can be exponentially faster than using the original SBDS. More details can be found in [Chu *et al.*, 2011].

We also present two new methods for exploiting almost symmetries. Many problems arising from real world applications do not exhibit full problem symmetry, but instead exhibit *almost symmetries*, where the symmetries are broken by a small number of constraints. Such problems often still have large amounts of symmetries that can be exploited for speedup. However, traditional symmetry breaking methods like lexicographical symmetry breaking constraints, SBDS or SBDD cannot directly handle such partially broken symmetries. We present two new general methods for exploiting almost symmetries. The first is to modify SDBS-1UIP to handle almost symmetries. The second is to treat almost symmetries as conditional symmetries and exploit them by posting conditional symmetry breaking constraints.

In the first, we modify SBDS-1UIP to keep track of exactly which constraints were used in order to derive a nogood n. Given an almost symmetry σ, if none of the constraints which break the symmetry σ were used to derive n, then $\sigma(n)$ is still a valid nogood and we can post it to achieve additional pruning. Alternatively, we could exploit the almost symmetry by considering it as a conditional symmetry. Suppose the set of constraints which break the symmetry σ is H, then we can post the conditional symmetry breaking constraint $\wedge_{c \in H} \sigma(c) \rightarrow sb(\sigma)$, where $\wedge_{c \in H} \sigma(c)$ are the conditions, and $sb(\sigma)$ is a standard symmetry breaking constraint for σ, e.g., a lex-leader symmetry breaking constraint. Both techniques are capable of producing orders of magnitude speedups on appropriate problems. More details of these methods can be found in the thesis.

5 Part 3 - Relaxation and Dominance

In Part 3 of the thesis, we examine several well studied problems. We present a new solver for the Minimization of Open Stacks Problem (MOSP) [Yuen and Richardson, 1995]. The MOSP was the subject of the 2005 Modeling Challenge. It has appeared in many different guises in the literature, e.g., graph pathwidth, gate matrix layout (see [Linhares and Yanasse, 2002] for a list of 12 equivalent problems). The MOSP is known to be NP-hard [Linhares and Yanasse, 2002]. The previous state of the art in solving the MOSP [Garcia de la Banda and Stuckey, 2007] utilizes a branch and bound algorithm and top down dynamic programming in order to exploit the massive amounts of subproblem equivalences inherent in the problem. This solver was several orders of magnitude faster than all the other entrants in the 2005 Modeling Challenge. Our new solver is many orders of magnitude faster yet again. We identify 4 dominance relations which describe several different conditions under which moving a task forward or backwards in the schedule will always lead to a solution which is at least as good. When exploited, these dominance relations provides some 5-6 orders of magnitude speedup. We also find an effective relaxation technique that allows us to speed up the proof of optimality on the hardest instances by a further 3-4 orders of magnitude. We identify the customers which are least densely connected with other customers (via shared product requests) and relax the problem by removing them. We relax the problem very aggressively initially. However, we try to detect when we have over relaxed and unrelax the problem by adding a removed customer back in. To do this, we first run a very good search heuristic on the original problem to get a probable optimal solution. When we relax the problem, we run this search heuristic again. If we find a solution better than the one we found for the original problem, then it is quite likely that we have over relaxed, in which case we stop relaxing and start unrelaxing until we can prove the correct optimality bound. On the hardest instances, we can often relax away up to one third of the customers and still be able to prove the same lower bound as the original problem. More details can be found in [Chu and Stuckey, 2009].

We present a new solver for the Talent Scheduling Problem [Cheng *et al.*, 1993]. In the Talent Scheduling Problem, we have a set of actors and a set of scenes. We wish to find an ordering of the scenes such that the cost of hiring the actors are minimized. Each actor is involved in some subset of scenes and must be on location from the first to the last of their scenes, getting paid for each day they are on location. The Talent Scheduling Problem problem has a lot of subproblem equivalences which can be exploited by caching [Smith, 2005], dynamic programming, or similar techniques. We present a new solver based on dynamic programming that is some 2-3 orders of magnitude faster than the previous state of the art. We show that using bounded dynamic programming gives 1-2 orders of magnitude speedup compared to a naive dynamic programming approach. Exploiting dominance relations based on moving a scene forwards or backwards in the schedule gives a further $\sim$2 times speedup, and utilizing a lower bound calculation based on relaxations gives a further 1 order of magnitude speedup. More details can be found in [Garcia de la Banda *et al.*, 2011].

We also completely solve the Maximum Density Still Life Problem [Bosch and Trick, 2004] for all n. The Maximum Density Still Life Problem is based on Conway's Game of Life. The aim is to find the maximum number of live cells that can appear in an $n \times n$ region in a *still life* according to the rules of

the game. It is an extremely difficult combinatorial optimization problem with a raw search space of $O(2^{n^2})$. Previous search methods using IP [Bosch, 1999] and CP [Bosch and Trick, 2004] could only solve up to $n = 9$, while a CP/IP hybrid method with symmetry breaking [Bosch and Trick, 2004] could solve up to $n = 15$. An attempt using bucket elimination [Larrosa and Dechter, 2003] reduced the time complexity to $O(n^2 2^{3n})$ but increased the space complexity to $O(n2^{2n})$. This method could solve up to $n = 14$ before it ran out of memory. A subsequent improvement that combined bucket elimination with search [Larrosa *et al.*, 2005], used less memory and was able to solve up to $n = 20$. We completely solve the problem for all n by using a combination of remodeling, relaxations, bounded dynamic programming and customized search.

The key to solving the problem is a mathematical proof that a "wastage" number could be assigned to each 3×3 pattern such that the total number of live cells is a function of n minus the sum of the wastage of all the 3×3 patterns in the $n \times n$ region. This proof tells us exactly which 3×3 patterns are good and which ones are wasting space, and allows us to remodel the problem in terms of minimizing wastage rather than maximizing the number of live cells. The reformulation allows tighter bounds on the objective to be found, which allows branch and bound to prune failed subtrees earlier. Also, by analysing existing optimal solutions in terms of wastage, we found that wastage only occurs near the edge or corners of the $n \times n$ region in optimal solutions, leading to the conjecture that it is only the boundary conditions which are forcing wastage. This was subsequently confirmed and meant that we can derive a proof of optimality by relaxing the problem onto only a width 13 boundary no matter how large n is, significantly reducing the number of variables in the problem. Furthermore, the relaxed problem has a constraint graph with low pathwidth, and thus can be solved using dynamic programming in $O(n)$ (albeit with a very large constant factor). By taking advantage of the translational symmetry inherent in the still life constraints, we were able to prove tight upper bounds on the number of live cells for all n in finite time. Then, by finding optimal solutions which achieve these upper bounds and which satisfy additional periodicity constraints, we were able to construct provably optimal solutions for all n. See Figure 2 for an optimal solution to $n = 100$. The Maximum Density Still Life problem is now completely closed. More details on the method can be found in [Chu *et al.*, 2009c] and [Chu and Stuckey, 2012].

6 Part 4 - Parallelization

Parallelization is an obvious way to try to speedup constraint solving, especially in view of the recent proliferation of multi-core systems. In part 4 of the thesis, we identify cache contention as a serious bottleneck that severely limits the amount of speedup achievable by parallel SAT and Lazy Clause Generation solvers on multi-core systems. We alter the data structures used in the state of the art SAT solver MiniSat [Eén and Sörensson, 2003] to be more cache aware, leading to a sequential SAT solver that is some 80% faster on average and a parallel SAT solver that is some 140% faster on average.

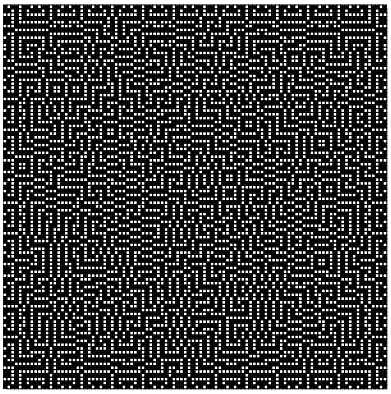

Figure 2: Optimal solution for $n = 100$.

More details on these data structures can be found in [Chu *et al.*, 2009a].

We also examine the interaction between branching heuristics and dynamic load balancing schemes. This important theoretical issue has not been properly addressed in the literature. Many analyses of parallel search and load balancing schemes implicitly or explicitly assume that the total amount of work is fixed (e.g. [Kumar and Rao, 1987]). This assumption is not true for satisfiable problems or for optimization problems, as finding a (good) solution quickly can dramatically reduce the total amount of work required. Much work in CP is concerned with developing strong branching heuristics which can shape and order the branches of the search tree so that (good) solutions are found early on in the search. However, many commonly used parallelization schemes, e.g., splitting the search tree as close to the root as possible, focus on maximizing the granularity of the work partitions and completely ignore the branching heuristic. This can result in many of the processors searching in unfruitful parts of the search tree.

We analyze this problem and develop a new work stealing scheme called *confidence based work stealing*, which can make use of the information provided by the branching heuristic. The new scheme tries to partition the processors in proportion to the estimated solution density of each subtree. The solution densities are initially estimated using the strength of the branching heuristic at the node, and is then dynamically updated using negative reinforcement learning as nodes are searched and failed. The new parallel algorithm produced near linear or super linear speedup on all the problems we tested. More details on the method can be found in [Chu *et al.*, 2009b].

7 Conclusion

This thesis presents important contributions to a number of areas in Constraint Programming, including nogood learning, symmetry breaking, relaxations, dominance and parallelization. These include new generic techniques such as nogood learning using constraint projection, SBDS-1UIP for symmetry breaking, SBDS-1UIP and conditional symmetry breaking constraints for almost symmetry breaking, and confidence-

based work stealing for parallel search, as well as problem specific techniques such as dominance rules and relaxations which gave orders of magnitude speedup on the Minimization of Open Stacks Problem, the Talent Scheduling Problem, and the Maximum Density Still Life Problem.

References

[Bosch and Trick, 2004] Robert Bosch and Michael Trick. Constraint Programming and Hybrid Formulations for Three Life Designs. *Annals of OR*, 130(1-4):41–56, 2004.

[Bosch, 1999] R. Bosch. Integer Programming and Conway's Game of Life. *SIAM Review*, 41(3):596–604, 1999.

[Cheng *et al.*, 1993] T. C. E. Cheng, J. E. Diamond, and B. M. T. Lin. Optimal Scheduling in Film Production to Minimize Talent Hold Cost. *Journal of Optimization Theory and Applications*, 79(3):479–482, 1993.

[Chu and Stuckey, 2009] Geoffrey Chu and Peter J. Stuckey. Minimizing the Maximum Number of Open Stacks by Customer Search. In *Proc. CP 2009*, volume 5732 of *LNCS*, pages 242–257. Springer, 2009.

[Chu and Stuckey, 2012] Geoffrey Chu and Peter J. Stuckey. A complete solution to the maximum density still life problem. *Artificial Intelligence*, 2012. Accepted on 8 February 2012.

[Chu *et al.*, 2009a] Geoffrey Chu, Aaron Harwood, and Peter J. Stuckey. Cache Conscious Data Structures for Boolean Satisfiability Solvers. *JSAT*, 6(1-3):99–120, 2009.

[Chu *et al.*, 2009b] Geoffrey Chu, Christian Schulte, and Peter J. Stuckey. Confidence-Based Work Stealing in Parallel Constraint Programming. In *Proc. CP 2009*, volume 5732 of *LNCS*, pages 226–241. Springer, 2009.

[Chu *et al.*, 2009c] Geoffrey Chu, Peter J. Stuckey, and Maria Garcia de la Banda. Using Relaxations in Maximum Density Still Life. In *Proc. CP 2009*, volume 5732 of *LNCS*, pages 258–273. Springer, 2009.

[Chu *et al.*, 2010] Geoffrey Chu, Maria Garcia de la Banda, and Peter J. Stuckey. Automatically Exploiting Subproblem Equivalence in Constraint Programming. In *CPAIOR*, volume 6140 of *LNCS*, pages 71–86. Springer, 2010.

[Chu *et al.*, 2011] Geoffrey Chu, Maria Garcia de la Banda, C. Mears, and P. J. Stuckey. Symmetries and Lazy Clause Generation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.

[Chu *et al.*, 2012] Geoffrey Chu, Maria Garcia de la Banda, and Peter J. Stuckey. Exploiting subproblem dominance in constraint programming. *Constraints*, 17(1):1–38, 2012.

[Chu, 2011] Geoffrey Chu. *Improving Combinatorial Optimization*. PhD thesis, The University of Melbourne, 2011. Available at http://repository.unimelb.edu.au/10187/11036.

[Crawford *et al.*, 1996] James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-Breaking Predicates for Search Problems. In *Proc. KR 1996*, pages 148–159. Morgan Kaufmann, 1996.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Proc. SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[Fahle *et al.*, 2001] Torsten Fahle, Stefan Schamberger, and Meinolf Sellmann. Symmetry Breaking. In *Proc. CP 2001*, volume 2239 of *LNCS*, pages 93–107. Springer, 2001.

[Feydy and Stuckey, 2009] Thibaut Feydy and Peter J. Stuckey. Lazy Clause Generation Reengineered. In *Proc. CP 2009*, volume 5732 of *LNCS*, pages 352–366. Springer, 2009.

[Frisch *et al.*, 2003] Alan M. Frisch, Christopher Jefferson, and Ian Miguel. Constraints for Breaking More Row and Column Symmetries. In *Proc. CP 2003*, volume 2833 of *LNCS*, pages 318–332. Springer, 2003.

[Garcia de la Banda and Stuckey, 2007] Maria Garcia de la Banda and Peter J. Stuckey. Dynamic Programming to Minimize the Maximum Number of Open Stacks. *INFORMS Journal on Computing*, 19(4):607–617, 2007.

[Garcia de la Banda *et al.*, 2011] Maria Garcia de la Banda, Peter J. Stuckey, and Geoffrey Chu. Solving Talent Scheduling with Dynamic Programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011.

[Gent and Smith, 2000] Ian P. Gent and Barbara M. Smith. Symmetry Breaking in Constraint Programming. In *Proc. ECAI 2000*, pages 599–603. IOS Press, 2000.

[Kumar and Rao, 1987] V. Kumar and V.N. Rao. Parallel depth first search. Part II. analysis. *International Journal of Parallel Programming*, 16(6):501–519, 1987.

[Larrosa and Dechter, 2003] Javier Larrosa and Rina Dechter. Boosting Search with Variable Elimination in Constraint Optimization and Constraint Satisfaction Problems. *Constraints*, 8(3):303–326, 2003.

[Larrosa *et al.*, 2005] Javier Larrosa, Enric Morancho, and David Niso. On the Practical use of Variable Elimination in Constraint Optimization Problems: 'Still-life' as a Case Study. *J. Artif. Intell. Res. (JAIR)*, 23:421–440, 2005.

[Lecoutre *et al.*, 2007] Christophe Lecoutre, Lakhdar Sais, Sébastien Tabary, and Vincent Vidal. Recording and Minimizing Nogoods from Restarts. *JSAT*, 1(3-4):147–167, 2007.

[Linhares and Yanasse, 2002] Alexandre Linhares and Horacio Hideki Yanasse. Connections between cutting-pattern sequencing, VLSI design, and flexible machines. *Computers & OR*, 29(12):1759–1772, 2002.

[Luks and Roy, 2004] Eugene M. Luks and Amitabha Roy. The Complexity of Symmetry-Breaking Formulas. *Ann. Math. Artif. Intell.*, 41(1):19–45, 2004.

[Ohrimenko *et al.*, 2007] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation = Lazy Clause Generation. In *Proc. CP 2007*, volume 4741 of *LNCS*, pages 544–558. Springer, 2007.

[Puget, 1993] Jean Francois Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. In *Proc. ISMIS 1993*, volume 689 of *LNCS*, pages 350–361. Springer, 1993.

[Puget, 2005] Jean Francois Puget. Breaking symmetries in all different problems. In *Proc. IJCAI 2005*, pages 272–277. Professional Book Center, 2005.

[Silva and Sakallah, 1999] João P. Marques Silva and Karem A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[Smith, 2005] Barbara M. Smith. Caching Search States in Permutation Problems. In *Proc. CP 2005*, volume 3709 of *LNCS*, pages 637–651. Springer, 2005.

[Yuen and Richardson, 1995] B.J. Yuen and K.V. Richardson. Establishing the optimality of sequencing heuristics for cutting stock problems. *European Journal of Operational Research*, 84(3):590–598, 1995.