Local Search and Backtracking vs Non-Systematic Backtracking

Steven Prestwich
Department of Computer Science
University College, Cork, Ireland
s.prestwich@cs.ucc.ie

Abstract

This paper addresses the following question: what is the essential difference between stochastic local search (LS) and systematic backtracking (BT) that gives LS superior scalability? One possibility is LS's lack of firm commitment to any variable assignment. Three BT algorithms are modified to have this feature by introducing randomness into the choice of backtracking variable: a forward checker for n-queens, the DSATUR graph colouring algorithm, and a Davis-Logemann-Loveland procedure for satisfiability. In each case the modified algorithm scales like LS and sometimes gives better results. It is argued that randomised backtracking is a form of local search.

Introduction

A variety of algorithms have been devised to solve constraint satisfaction and optimisation problems, two contrasting approaches being *systematic backtracking* and *stochastic local search*. Backtrackers are complete and can therefore prove unsolvability or find all solutions. They can also prove the optimality of a solution to an optimisation problem by failing to find a better solution. They often use techniques such as constraint propagation, value and variable ordering heuristics, branch-and-bound and intelligent backtracking. Backtrackers are sometimes called *constructive* because they construct solutions from partial consistent assignments of domain values to variables. In contrast, local search algorithms are typically non-constructive, searching a space of total but inconsistent assignments. They attempt to minimise constraint violations, and when all violations have been removed a solution has been found.

Neither backtracking nor local search is seen as adequate for all problems. Real-world problems may have a great deal of structure and are often solved most effectively by backtrackers, which are able to exploit structure via constraint propagation. Unfortunately, backtrackers do not always scale well to very large problems, even when augmented with powerful constraint propagation and intelligent backtracking. For large problems it is often more efficient to use local search, which tends to have superior scalability. However, non-constructive local search cannot fully exploit structure, and is quite unsuitable for certain problems. This situation has motivated research into hybrid approaches, with the aim of combining features of both classes of algorithm in order to solve currently intractable problems.

What is the essential difference between local search and backtracking, that sometimes enables local search causes to scale far better than backtracking? Answering this question may lead to interesting new hybrids. The issue was debated in (Freuder et al. 1995) and some suggestions were local search's locality, randomness, incompleteness, different search space, and ability to follow gradients. We propose that the essential difference is backtrackers' strong commitment to early variable assignments, whereas local search can in principle (given sufficient noise) reassign any variable at any point during search. Though this seems rather an obvious point, it has led to a new class of algorithms that sometimes out-perform both backtracking and local search.

Non-systematic backtracking

The new approach has already been described in (Prestwich 2000a; Prestwich 2000b; Prestwich 2001a; Prestwich 2001b) but it is summarised here. It begins like a standard backtracker by selecting a variable using a heuristic, assigning a value to it, optionally performing constraint propagation, then selecting another variable; on reaching a dead-end (a variable that cannot be assigned any value without constraint violation or, where constraint propagation is used, domain wipe-out) it backtracks then resumes variable selection. The novel feature is the choice of backtracking variable: the variables selected for unassignment are chosen randomly, or using a heuristic with random tie-breaking. As in Dynamic Backtracking (Ginsberg 1993) only the selected variables are unassigned, without undoing later assignments. Because of this resemblance we call the approach Incomplete Dynamic Backtracking (IDB). The search is no longer complete, so we must sometimes force the unassignment of more than one variable. We do this by adding an integer parameter $B \geq 1$ to the algorithm and unassigning B variables at each dead-end.

A complication arises when we wish to combine IDB with forward checking. Suppose we have reached a dead-end after assigning variables $v_1 \dots v_k$, and $v_{k+1} \dots v_n$ remain unassigned. We would like to unassign some arbitrary variable v_u where $1 \leq u \leq k$, leaving the domains in the state they would have been in had we assigned only $v_1 \dots v_{u-1}, v_{u+1} \dots v_k$. How can we do this efficiently? One way to characterise forward checking is as follows: a value x is in the domain of a currently unassigned variable v if and only if adding the assignment $v = x$ would not cause a constraint violation. In backtracking algorithms this is often used to update the domains of unassigned variables as assignments are added and removed. We generalise this idea by associating with each value x in the domain of each variable v a *conflict count* $C_{v,x}$. The value of $C_{v,x}$ is *the number of constraints that would be violated* if the assignment $v = x$ were added. If $C_{v,x} = 0$ then x is currently in the domain of v. We make a further generalisation by maintaining the conflict counts of *all* variables, assigned and unassigned. The meaning of a conflict count for a currently assigned variable is: $C_{v,x}$ is the number of constraints that would be violated if the variable v were *reassigned* to x. Now on assigning or unassigning a variable, we incrementally update conflict counts in all other variable domains. This is clearly more expensive than standard forward checking, which only updates the domains of unassigned variables. But the state of each variable domain is now independent of the order in which assignments were made, and we can unassign variables arbitrarily. The scheme can also be extended to non-binary constraints (see below).

A value ordering heuristic has been found useful in several applications. It reassigns each variable to its previous assigned value where possible, otherwise to a different value. This speeds up the rediscovery of consistent partial assignments. However, to avoid stagnation it attempts to use a different (randomly-chosen) value for just one variable after each dead-end.

Experiments

In this section IDB is applied to three problems: n-queens, graph colouring and satisfiability. The experiments are performed on a 300MHz DEC Alphaserver 1000A 5/300 under Unix. For readers wishing to normalise our execution times to other platforms, the DIMACS (Johnson and Trick 1996) benchmark program *dfmax r500.5* takes 46.2 seconds on the Alphaserver.

The n-queens problem

We first evaluate IDB on the n-queens problem. Though fashionable several years ago n-queens is no longer considered a challenging problem. However, large instances still defeat most backtrackers and it is therefore of interest. Consider a generalised chess board, which is a square divided into $n \times n$ smaller squares. Place n queens on it in such a way that no queen attacks any other. A queen *attacks* another if it is on the same row, column or diagonal (in which case both attack each other). We can model this problem using n variables each with domain $D_i = \{1, \dots, n\}$. A variable v_i corresponds to a queen on row i (there is one queen per row), and the assignment $v_i = j$ denotes that the queen on row i is placed in column j, where $j \in D_i$. The constraints are $v_i \neq v_j$ and $|v_i - v_j| \neq |i - j|$ where $1 \leq i < j \leq n$. We must assign a domain value to each variable without violating any constraint.

	$n = 10$		$n = 100$		$n = 1000$	
alg	steps	succ	steps	succ	steps	succ
DFS1	81.0	100%	9929	1%	—	
DFS2	25.4	100%	7128	39%	98097	3%
DFS3	14.7	100%	1268	92%	77060	24%
IDB1	112	100%	711	100%	1213	100%
IDB2	33.0	100%	141	100%	211	100%
IDB3	23.8	100%	46.3	100%	41.2	100%
IDB4	13.0	100%	8.7	100%	13.3	100%
IDB5	12.7	100%	8.0	100%	12.3	100%
MCHC	57.0	100%	55.6	100%	48.8	100%
MCBT	46.8	100%	25.0	100%	30.7	100%

Figure 1: Results on n-queens

(Minton et al. 1992) compared the performance of backtracking and local search on n-queens problems up to $n = 10^6$. They executed each algorithm 100 times for various values of n, with an upper bound of $100n$ on the number of steps (backtracks or repairs), and reported the mean number of steps and the success rate as a percentage. We reproduce the experiment up to $n = 1000$, citing their results for the Min-Conflicts hill climbing algorithm (denoted here by MCHC) and a backtracker augmented with the Min-Conflicts heuristic (denoted by MCBT). We compute results for depth-first search (DFS1), DFS1 with forward checking (DFS2), and DFS2 with dynamic variable ordering based on minimum domain size (DFS3). In DFS1 and DFS2 the variable to be assigned is selected randomly. (Our results differ from those of Minton et al., possibly because of algorithmic details such as random tie-breaking during variable ordering.) We also obtain results for these three algorithms with DFS replaced by IDB (denoted by IDB1, IDB2, IDB3); also for IDB3 plus a backtracking heuristic that unassigns variables with *maximum* domain size [1] (IDB4), and for IDB4 plus the value ordering heuristic described above (IDB5). The IDB parameter is set to $B = 1$ for $n = 1000$ and $n = 100$, and $B = 2$ for $n = 10$ ($B = 1$ sometimes causes stagnation when $n = 10$).

The results in Figure 1 show that replacing DFS by IDB greatly boosts scalability in three cases: the simple backtracking algorithm, backtracking with forward checking, and the same algorithm with dynamic variable ordering. Even the basic IDB algorithm scales much better than all the DFS algorithms, and IDB3 performs like MCHC. The additional backtracking and value ordering heuristics further boost performance, making IDB the best reported algorithm in terms of backtracks. It also compares well with another hybrid: Weak Commitment Search (Yokoo 1994) requires approximately 35 steps for large n (Pothos and Richards 1995). In terms of execution time IDB is also efficient, each backtrack taking a time linear in the problem size.

It should be noted that IDB is not the only backtracker to perform like local search on n-queens. Similar results were obtained by Minton et al.'s MCBT algorithm (see Figure 1)

[1]Recall that domain information is also maintained for assigned variables.

and others. These algorithms rely on good value ordering heuristics. In MCBT an initial total assignment I is generated by the Min-Conflicts heuristic and used to guide DFS in two ways. Firstly, variables are selected for assignment on the basis of how many violations they cause in I. Secondly, values are tried in ascending order of number of violations with currently unassigned variables. This *informed backtracking* algorithm performs almost identically to MCHC on n-queens and is complete. However, MCBT is still prone to the same drawback as most backtrackers: a poor choice of assignment high in the search tree will still take a very long time to recover from. IDB is able to modify earlier choices, as long as the B parameter is sufficiently high, so it can recover from poor early decisions. This difference is not apparent on n-queens, but it will be significant on problems for which value ordering heuristics are of little help.

Graph colouring

Graph colouring is a combinatorial optimisation problem with real-world applications such as timetabling, scheduling, frequency assignment, computer register allocation, printed circuit board testing and pattern matching. A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges between vertices. Two vertices connected by an edge are said to be *adjacent*. The aim is to assign a colour to each vertex in such a way that no two adjacent vertices have the same colour, using as few colours as possible.

We apply an algorithm similar to IDB5 (see previous section), again with forward checking implemented via conflict counts. However, the largest-domain backtracking heuristic used for n-queens causes the search to stagnate on some colouring problems, so we use a modified heuristic: randomly select a variable with domain size greater than one, if possible; otherwise select randomly. We also use a different variable ordering: the Brélaz heuristic (Brélaz 1979). This selects an unassigned variable with minimum domain size, breaking ties by considering the degree in the uncoloured part of the graph, and breaking further ties randomly. The Brélaz heuristic is used in the well-known DSATUR colouring algorithm (as well as in other constraint applications) and our IDB algorithm is a modified DSATUR. Again like DSATUR, on finding each colouring our algorithm restarts with one less available colour, and reuses the previous assignments as far as possible.

3-colourable random graphs are often used as colouring benchmarks. (Pothos and Richards 1995) compared the performance of MCHC with that of Weak Commitment Search (WCS) on these graphs, with 60 vertices and average degree between 3 and 7. Using a maximum of 2000 steps both algorithms had a lower success rate around the phase transition near degree 5: WCS (without Brélaz heuristics or forward checking) dipped to about 80% while MCHC dipped to less than 5%. A modified MCHC allowing uphill moves performed much like WCS. We generate such graphs in the same way and apply IDB to them under the same conditions: over 1000 runs with tuned values of the parameter B (increasing from 1 to 9 near the phase transition) its success rate dips to 98.5% at degree 4.67. Hence IDB beats MCHC on at least some colouring problems.

	DSATUR		IDB		
graph	col	sec	col	sec	B
R125.1	5	0.0	5	0.01	1
R250.1	8	0.0	8	0.1	1
DSJR500.1	12	0.2	12	0.4	1
R1000.1	20	1.6	20	1.81	1
R125.5	36	63.4	36	0.06	1
R250.5	66	2.9	65	0.33	2
DSJR500.5	130	17.6	122	1.99	7
R1000.5	246	75.8	235	8.61	6
DSJC125.5	20	0.7	18	16.4	1
DSJC250.5	35	198	32	81.3	1
DSJC500.5	63	5.8	59	4.41	1
DSJC1000.5	114	5.0	107	13.2	1

Figure 2: Results on selected DIMACS benchmarks

Minton et al. found that DSATUR out-performed MCHC on these graphs, and (Jónsson and Ginsberg 1993) found that depth-first search and Dynamic Backtracking both beat MCHC and the GSAT local search algorithm on similar graphs. To compare IDB with DSATUR we therefore need harder benchmarks. We use selected DIMACS graphs and compare IDB with M. Trick's DSATUR implementation [2] which also performs some pre-processing of graphs by finding a large clique, an enhancement IDB does not have. We use two types of graph: random and geometric. Random graphs have little structure for constraint algorithms to exploit, while geometric graphs tend to have large cliques. Even without the clique enhancement DSATUR works quite well on geometric graphs (Culberson, Beacham, and Papp 1995), so these are a challenging test for IDB.

Figure 2 shows the results. IDB results are means over 10 runs and start with the number of colours shown plus 10. DSATUR's results are obtained from a single run because its heuristics are deterministic. We shall also mention results for five other algorithms in (Joslin and Clements 1999): Squeaky Wheel Optimization, TABU search (both general-purpose combinatorial optimisation algorithms), Iterative Greedy, distributed IMPASSE and parallel IMPASSE (all pure colouring algorithms).

On the sparse geometric graphs ({R,DSJR}*.1) IDB and DSATUR are both very fast, often requiring no backtracks at all. The other five algorithms also find these problems easy. On the denser geometric graphs ({R,DSJR}*.5) IDB consistently finds the same or better colourings in shorter times than DSATUR or any of the other algorithms. On two of the graphs IDB finds a better colouring than any of the other algorithms: the previous best for DSJR500.5 was distributed IMPASSE with 123 in 94.1 seconds, and the best for R1000.5 was Squeaky Wheel Optimization with 239 in 309 seconds (times normalised to our platform).

On the random graphs (DSCJ*.5) IDB also finds consistently better colourings than DSATUR. Though DSATUR's reported times are sometimes shorter, it was given 5 minutes to find a better colouring but did not; it tended to

[2] http://mat.gsia.cmu.edu/COLOR/color.html

find a reasonable colouring quickly, then made little further progress. However, IDB is not the best algorithm on the random graphs, beating only DSATUR and TABU and performing roughly like Iterative Greedy.

Satisfiability

The satisfiability (SAT) problem is of both practical and theoretical importance. It was the first problem shown to be NP-complete, has been the subject of a great deal of recent research and has well-known benchmarks and algorithms, making it ideal for evaluating new approaches. The SAT problem is to determine whether a Boolean expression has a satisfying set of truth assignments. The problems are usually expressed in conjunctive normal form: a conjunction of clauses $C_1 \wedge \ldots \wedge C_m$ where each clause C is a disjunction of literals $l_1 \vee \ldots \vee l_n$ and each literal l is either a Boolean variable v or its negation $\neg v$.

Many systematic SAT and other constraint algorithms are based on backtracking. An early example is the Davis-Putnam procedure in Loveland's form (Davis, Logemann, and Loveland 1962), using depth-first search with inference rules such as unit propagation. Modern systematic SAT solvers are still based on this scheme and include TABLEAU (Crawford and Auton 1996), RELSAT (Bayardo and Schrag 1997), POSIT (Freeman 1994), SATO (Zhang 1997), GRASP (Silva and Sakallah 1996) and SATZ (Li and Anbulagan 1997). The main difference between these solvers is their variable ordering heuristics; SATZ also has a preprocessing phase and RELSAT uses look-back techniques. There are also several local search SAT algorithms. Early examples are GSAT (Selman, Levesque, and Mitchell 1992) and Gu's algorithms (Gu 1992). GSAT has since been enhanced in various ways (Selman, Kautz, and Cohen 1994) giving several WSAT variants. Other local search SAT algorithms include DLM (Wu and Wah 2000) and GRASP (Resende and Feo 1996). DLM is based on Lagrange multipliers, while GRASP is a greedy, randomised, adaptive algorithm. (There is no relation between the GRASP backtracking and local search algorithms.)

IDB has already been applied to SAT by modifying a simple Davis-Logemann-Loveland procedure. The use of conflict counts is easily adapted to unit resolution in SAT: the value of $C_{v,x}$ denotes in how many clauses (i) all other literals are false, and (ii) the literal containing v has sign $\neg x$. We now describe new heuristics which improve performance on some problems. The new variable ordering heuristic randomly selects a variable with domain size 1; if there are none then it selects the variable that was unassignable at the last dead-end; when that has been used, it selects the most recently unassigned variable. For backtracking the variable with lowest conflict counts is selected, breaking ties randomly. A probabilistic value ordering rule is used as follows. Each variable records its last assigned value (initially set to a random truth value). For each assignment, with probability $1/u$ the negated value is preferred, and with probability $1 - 1/u$ the value itself, where u is the number of currently unassigned variables. Finally, the integer parameter B is replaced by a real parameter p ($0 < p < 1$). At each dead-end one variable is unassigned as usual, and further variables are iteratively unassigned with probability p.

problem	time	p
f600	4.35	0.01
f1000	84.1	0.01
f2000	686	0.01
ssa-7552-*	0.31	0.998
ii16*1	0.305	0.1
ii32*3	1.17	0.1
2bitadd_11	0.026	0.998
2bitadd_12	0.028	0.998
2bitcomp_5	0.002	0.998
3bitadd_31	11.0	0.998
3bitadd_32	8.6	0.998
bw_large.a	0.425	0.9
bw_large.b	7.75	0.95
bw_large.c	48.9	0.98
bw_large.d	295	0.997
hanoi4	4156	0.98

Figure 3: IDB on selected SAT benchmarks

In (Prestwich 2000b) IDB was shown to scale almost identically to WSAT on hard random 3-SAT problems. In (Prestwich 2000a) it was shown to efficiently solve three classes of problem considered hard for local search (though DLM has recently performed efficiently on these problems (Wu and Wah 2000)): all-interval series, parity learning and some AIM problems; it also beat backtracking, local search and some hybrids on SAT-encoded geometric graph colouring problems. Figure 3 shows the performance of the new SAT variant on benchmarks from three sources: the 2nd DIMACS Implementation Challenge, [3] the SATLIB problem repository, [4] and the International Competition and Symposium on Satisfiability Testing in Beijing. [5] It has similar performance to the previous implementation on most of the problems already tested, but is several times faster on these problems.

Random 3-SAT is a standard benchmark for SAT algorithms, the hard problems being found near the phase transition. Backtracking can efficiently solve such problems only up to a few hundred variables (Crawford and Auton 1996) and we were unable to solve f600 with SATZ, but local search algorithms can solve much larger problems (Selman, Kautz, and Cohen 1994; Wu and Wah 2000). The results, which are means over 20 runs, verify that IDB has the scalability of local search, though WSAT is faster and the latest DLM implementation can solve f2000 in 19.2 seconds on a 400MHz Pentium-II processor.

Circuit fault analysis SAT problems are harder for backtracking than for local search. The IDB results are means over 20 runs for all four DIMACS problems denoted by ssa-7552-*. It has similar performance to WSAT, is slower than DLM, and faster than the GRASP local search algorithm and

[3]ftp://dimacs.rutgers.edu/pub/challenge/

[4]http://aida.intellektik.informatik.th-darmstadt.de/~hoos/SATLIB

[5]http://www.cirl.uoregon.edu/crawford/beijing

four backtrackers (SATZ, GRASP, POSIT and RELSAT), ranking joint second of eight algorithms.

Circuit synthesis SAT problems are also hard for backtrackers. The IDB results for these problems are means over 100 runs. On the benchmarks ii16*1 and ii32*1 it is slower than DLM and GSAT with random walk but faster than the GRASP local search algorithm, performing like a reasonably efficient local search algorithm. On the benchmarks 2bitadd_11 etc, IDB times are means over 20 runs. It has similar performance to that of GSAT with random walk and is many times faster than the backtrackers SATZ, GRASP, POSIT and RELSAT on the 3bitadd problems. However, WSAT is several times faster again.

Blocks World yields some of the hardest SAT benchmarks (bw_large.*), on which there is no clear dominance between backtracking and local search. Comparing IDB times over 20 runs with published results for the systematic TABLEAU, SATZ, GRASP, POSIT and RELSAT, and the local search algorithm WSAT, we find that on the smallest problem IDB ranks seventh, on the next it ranks sixth, on the next fourth, and on the largest second. This reflects its local search-like scalability.

TABLEAU is able to solve hanoi4 in 3.2 hours on an SGI Challenge (time given in SATLIB), and DLM in a mean of 6515 seconds over 10 runs on a Pentium-III 500MHz with Solaris 7. Hence IDB appears to be the fastest current algorithm on hanoi4.

Like most local search algorithms IDB is incomplete, but (Hoos 1998) defines an analogous property to completeness for randomised algorithms called *approximate completeness*, also called *convergence* in the literature: if the probability of a finding an existing solution tends to 1 as the run time tends to ∞ then the algorithm is convergent. The new SAT heuristics for IDB allow the convergence property to be proved as follows. Any search algorithm that periodically restarts with random variable assignments is clearly convergent, because each restart has a non-zero probability of finding a solution directly. Now IDB has a non-zero probability (p^{n-u-1} where n is the number of SAT variables, u the number of unassigned variables, and $p > 0$) of restarting, and on reassigning each variable it has a non-zero probability of selecting either the same value as previously used ($1 - 1/u$) or the negated value ($1/u$). Therefore IDB may discover a solution immediately following any dead-end, and is convergent. As pointed out by (Morris 1993), convergence does not prevent the time taken to find a solution from being unreasonably long. However, Hoos found that modifying local search algorithms to make them convergent often boosted performance, and we have found the same with IDB.

Discussion

IDB combines backtracker-like constraint handling with local search-like scalability, giving competitive results across a wide variety of problems. Nevertheless, it is not always the fastest approach for any given problem (for example WSAT is faster on many SAT benchmarks) and should be seen as complementary to existing approaches. Its main strength lies in its ability to solve problems that are too large for backtracking yet too structured for local search, as shown on geometric graph colouring. Previous results provide further evidence of this strength: an IDB-modified constraint solver for Golomb rulers out-performed the systematic original, and also a genetic algorithm (Prestwich 2001b); an IDB-modified branch-and-bound algorithm for a hard optimisation problem out-performed the systematic original, and all known local search algorithms (Prestwich 2000b); and very competitive results were obtained on DIMACS maximum clique benchmarks using forward checking with IDB (Prestwich 2001b).

Should IDB be classified as a backtracker, as local search, as both or as a hybrid? It may appear to be simply an inferior version of Dynamic Backtracking (DB), sacrificing the important property of completeness to no good purpose. A counter-example to this view is the n-queens problem, on which DB performs like chronological backtracking (Jónsson and Ginsberg 1993), whereas IDB performs like local search. Another is the random 3-SAT problem, on which DB is slower than chronological backtracking (though a modified DB is similar) (Baker 1994) while IDB again performs like local search. We claim that IDB is in fact a local search algorithm, and in previous papers it has been referred to as Constrained Local Search. Our view is that it stochastically explores a space of consistent partial assignments while trying to minimise an objective function: the number of unassigned variables. The forward local moves are variable assignments, while the backward local moves are unassignments (backtracks). However, to some extent the question is academic: even if IDB is not local search, results show that it captures its essence.

Other researchers have designed algorithms using backtracking techniques but with improved scalability. Iterative Sampling (Langley 1992) restarts a constructive search every time a dead-end is reached. Bounded Backtrack Search (Harvey 1995) is a hybrid of Iterative Sampling and chronological backtracking, alternating a limited amount of chronological backtracking with random restarts. (Gomes, Selman, and Kautz 1998) periodically restart chronological or intelligent backtracking with slightly randomised heuristics. (Ginsberg and McAllester 1994) describe a hybrid of the GSAT local search algorithm and Dynamic Backtracking that increases the flexibility in choice of backtracking variable. They note that to achieve total flexibility while preserving completeness would require exponential memory, and recommend a less flexible version using only polynomial memory. Weak Commitment Search (Yokoo 1994) builds consistent partial assignments, using greedy search (the min-conflict heuristic) to guide value selection. On reaching a dead-end it restarts and uses learning to avoid redundant search. Local Changes (Verfaillie and Schiex 1994) is a complete backtracking algorithm that uses conflict analysis to unassign variables leading to constraint violation. Limited Discrepancy Search (Harvey 1995) searches the neighbourhood of a consistent partial assignment, trying neighbours in increasing order of distance from the partial assignment. It is shown theoretically, and experimentally on job shop scheduling problems, to be superior to Iterative Sampling and chronological backtracking.

There are also many hybrid approaches. (Schaerf 1997) describes an algorithm that explores the space of all partial assignments (IDB explores only the *consistent* partial assignments). The Path-Repair Algorithm (Jussien and Lhomme 2000) is a generalisation of Schaerf's approach that includes learning, allowing complete versions to be devised. The two-phase algorithm of (Zhang and Zhang 1996) searches a space of partial assignments, alternating backtracking search with local search. (de Backer et al. 1997) generate partial assignments to key variables by local search, then pass them to a constraint solver that checks consistency. (Pesant and Gendreau 1996) use branch-and-bound to efficiently explore local search neighbourhoods. Large Neighbourhood Search (Shaw 1998) performs local search and uses backtracking to test the legality of moves. (Crawford 1993) uses local search within a complete SAT solver to select the best branching variable.

Future work will include investigating whether IDB can be made complete, thus combining all the advantages of systematic and non-systematic search. This could be achieved by learning techniques, but memory requirements may make this approach impractical for very large problems.

References

Baker, A. B. 1994. The Hazards of Fancy Backtracking. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1:288–293. Seattle, WA: AAAI Press.

Bayardo, R. J. Jr, and Schrag, R. 1997. Using CSP Look-Back Techniques to Solve Real World SAT Instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 203–208. Providence, RI: AAAI Press.

Brélaz, D. 1979. New Methods to Color the Vertices of a Graph. *Communications of the ACM* 22(4):251–256.

Crawford. J. M. 1993. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. In *Second DIMACS Challenge: Cliques, Coloring, and Satisfiability*, Rutgers University, NJ.

Crawford, J. M., and Auton, L. D. 1996. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence* 81(1–2):31–57.

Culberson, J. C.; Beacham A.; and Papp, D. 1995. Hiding Our Colors. In *Proceedings of the CP'95 Workshop on Studying and Solving Really Hard Problems*, Cassis, France.

Davis, M.; Logemann, G.; and Loveland, D. 1962. A Machine Program for Theorem Proving. *Communications of the ACM* 5:394–397.

de Backer, B.; Furnon, V.; Kilby, P.; Prosser, P.; and Shaw, P. 1997. Local Search in Constraint Programming: Application to the Vehicle Routing Problem. In *Constraint Programming 97, Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*.

Freeman, J. W. 1994. Improvements to Propositional Satisfiability Search Algorithms. Ph.D. diss., University of Pennsylvania, Philadelphia.

Freuder, E. C.; Dechter, R.; Ginsberg, M. L.; Selman, B.; and Tsang, E. 1995. Systematic Versus Stochastic Constraint Satisfaction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 2027–2032. Montreal, Quebec: Morgan Kaufmann.

Ginsberg, M. L. 1993. Dynamic Backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Ginsberg, M. L., and McAllester, D. A. 1994. GSAT and Dynamic Backtracking. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, 226–237. Bonn, Germany: Morgan Kaufmann.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 431–437. Madison, WI: AAAI Press.

Gu, J. 1992. Efficient Local Search for Very Large-Scale Satisfiability Problems. *SIGART Bulletin* 3(1):8–12.

Harvey, W. D. 1995. Nonsystematic Backtracking Search. Ph.D. diss., Stanford University.

Hoos, H. H. 1998. Stochastic Local Search — Methods, Models, Applications. PhD diss., Dept. of Computer Science, TU Darmstadt.

Johnson, D. S., and Trick, M. A. eds. 1996. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 26. American Mathematical Society.

Jónsson, A. K., and Ginsberg, M. L. 1993. Experimenting with New Systematic and Nonsystematic Search Techniques. In *Proceedings of the AAAI Spring Symposium on AI and NP-Hard Problems*. Stanford, California: AAAI Press.

Joslin D. E., and Clements, D. P. 1999. Squeaky Wheel Optimization. *Journal of Artificial Intelligence Research* 10:353–373.

Jussien, N., and Lhomme, O. 2000. Local Search With Constraint Propagation and Conflict-Based Heuristics. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 169–174. Austin, TX: AAAI Press.

Langley, P. 1992. Systematic and Nonsystematic Search Strategies. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 145–152. University of Maryland: Morgan Kaufman.

Li, C. M., and Anbulagan. 1997. Look-Ahead Versus Look-Back for Satisfiability Problems. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, 341–355. Schloss Hagenburg, Austria: Lecture Notes in Computer Science 1330, Springer-Verlag.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58(1–3):160–205.

Morris, P. 1993. The Breakout Method for Escaping Local Minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 40–45. Washington DC: AAAI Press.

Pesant, G., and Gendreau, M. 1996. A View of Local Search in Constraint Programming. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, 353–366. Cambridge, MA: Lecture Notes in Computer Science 1118, Springer-Verlag.

Pothos, D. G., and Richards, E. B. 1995. An Empirical Study of Min-Conflict Hill Climbing and Weak Commitment Search. In *Proceedings of the CP-95 Workshop on Studying and Solving Really Hard Problems*, 140–146. Cassis, France.

Prestwich, S. D. 2000a. Stochastic Local Search In Constrained Spaces. In *Proceedings of Practical Applications of Constraint Technology and Logic Programming*, 27–39. Manchester, England: The Practical Applications Company Ltd.

Prestwich, S. D. 2000b. A Hybrid Search Architecture Applied to Hard Random 3-SAT and Low-Autocorrelation Binary Sequences. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, 337–352. Singapore: Lecture Notes in Computer Science 1894, Springer-Verlag.

Prestwich, S. D. 2001a. Coloration Neighbourhood Search With Forward Checking. *Annals of Mathematics and Artificial Intelligence, special issue on Large Scale Combinatorial Optimization and Constraints*. Forthcoming.

Prestwich, S. D. 2001b. Trading Completeness for Scalability: Hybrid Search for Cliques and Rulers. In *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 159–174. Ashford, Kent, England. Forthcoming.

Resende, M. G. C., and Feo, T. 1996. A GRASP for Satisfiability. In (Johnson and Trick 1996) 499–520.

Schaerf, A. 1997. Combining Local Search and Look-Ahead for Scheduling and Constraint Satisfaction Problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1254–1259. Nagoya, Japan: Morgan Kaufmann.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440–446. San Jose, CA: AAAI Press.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise Strategies for Improving Local Search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 337–343. Seattle, WA: AAAI Press.

Shaw, P. 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, 417–431. Pisa, Italy: Lecture Notes in Computer Science 1520, Springer-Verlag.

Silva, J. P. M., and Sakallah, K. A. 1996. Conflict Analysis in Search Algorithms for Propositional Satisfiability. In *Proceedings of the Eighth International Conference on Tools with Artificial Intelligence*. Toulouse, France: IEEE Computer Society Press.

Verfaillie, G., and Schiex, T. 1994. Solution Reuse in Dynamic Constraint Satisfaction Problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 307–312. Seattle, WA: AAAI Press.

Wu, Z., and Wah, B. W. 2000. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 310–315. Austin, TX: AAAI Press.

Yokoo, M. 1994. Weak-Commitment Search for Solving Constraint Satisfaction Problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 313–318. Seattle, WA: AAAI Press.

Zhang, H. 1997. SATO: An Efficient Propositional Prover. In *Proceedings of the Fourteenth International Conference on Automated Deduction*, 272–275. Townsville, Australia: Lecture Notes in Computer Science vol. 1249, Springer-Verlag.

J. Zhang, H. Zhang. Combining Local Search and Backtracking Techniques for Constraint Satisfaction. *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Conference on Innovative Applications of Artificial Intelligence*, 369–374. Portland, OR: AAAI Press / MIT Press.