Interleaved and Discrepancy Based Search

Pedro Meseguer[1] and **Toby Walsh**[2]

Abstract. We present a detailed experimental comparison of interleaved depth-first search and depth-bounded discrepancy search, two tree search procedures recently developed with the same goal: to reduce the cost of heuristic mistakes at the top of the tree. Our comparison uses an abstract heuristic model, and three different concrete problem classes: binary constraint satisfaction, quasigroup completion and number partitioning problems. Results indicate that both search strategies often reduce search. In addition, they show that their efficiency depends on a trade-off between the number of discrepancies (branch points against the heuristic) considered at the top of the tree, and the overhead of expanding branches from these discrepancies. If the number of discrepancies is large, the overhead can outweigh the benefits.

1 INTRODUCTION

By definition, heuristics sometimes make mistakes. When searching a tree with depth-first search (DFS), mistakes made at the top of the tree can be very costly to undo. For instance, Crawford and Baker identified early mistakes as the cause of the poor performance for a Davis-Putnam procedure on Sadeh's scheduling benchmarks [2]. On some problems, a solution was found almost immediately. On other problems, an initial bad guess by the heuristic would lead to searching a subtree with the order of 2^{70} nodes. To tackle situations like this, Meseguer has proposed interleaved depth-first search [9]. At the same time, and for exactly the same reasons, Walsh proposed depth-bounded discrepancy search [14]. Both are systematic search strategies that reduce the cost of early mistakes. How do these search strategies compare, and when can we prefer one over the other?

2 BACKGROUND

A discrepancy is any branch point in a search tree where we go against the heuristic. For convenience, we assume that left branches follow the heuristic. Any other branch breaks the heuristic and is a discrepancy. For convenience, we call this a right branch. Limited discrepancy search (LDS) explores the leaf nodes in increasing order of the number of discrepancies taken to reach them [6]. On the kth iteration, LDS visits all leaf nodes with up to k discrepancies. Within each iteration, LDS explores branches with discrepancies high in the tree before those branches with discrepancies lower down.

Korf has proposed a small improvement to LDS, called ILDS that uses (an upper limit on) the maximum depth of the tree [7]. On the kth iteration, ILDS visits only those leaf nodes at the depth limit with exactly k discrepancies. LDS is less efficient since, on the kth iteration, it re-visits all those leaf nodes with between 0 and $k-1$ discrepancies. Within each iteration, ILDS explores branches with discrepancies low in the tree before those paths with discrepancies higher up. Apart from this small bias within each iteration, LDS and ILDS treat all discrepancies alike. However, we often expect heuristics to be less informed and to make more mistakes at the top of the search tree. Given a limited amount of search, it may therefore pay to explore discrepancies at the top of the search tree before those at the bottom. This is the motivation behind depth-bounded discrepancy search and interleaved depth-first search.

Depth-bounded discrepancy search (DDS) biases search to discrepancies high in the tree by means of an iteratively increasing depth bound [14]. Discrepancies below this bound are prohibited. On the 0th iteration, DDS explores the leftmost branch. On the $i+1$th iteration, DDS explores those branches on which discrepancies occur at a depth of i or less. As in ILDS, we are careful not to re-visit leaf nodes visited on earlier iterations. This is surprisingly easy to enforce. At depth i on the $i+1$th iteration, we take only right branches since left branches would take us to leaf nodes visited on an earlier iteration. At lesser depths, we can take both left and right branches. And at greater depths, we always follow the heuristic left. Search is terminated when the increasing bound is greater than the depth of the deepest leaf node. Note that LDS, ILDS and DDS all re-visit interior nodes. For example, when exhaustively traversing a binary tree of depth n, ILDS and DDS both visit approximately 2^{n+2} nodes, compared to DFS which visits just 2^{n+1} nodes [14]. Of course, our hope is that a solution is found before the whole tree is visited.

Interleaved depth-first search (IDFS) also biases search to discrepancies high in the tree. IDFS searches in parallel several subtrees (called *active*) at certain tree levels (called *parallel*). IDFS traverses depth-first the current active subtree until it finds a leaf. If it is a goal, search terminates. Otherwise, the state of the current subtree is recorded so that it can be resumed later, and IDFS switches to the earliest parallel level, where it selects another active subtree and repeats the process. At each level active subtrees form a circular queue. There are two versions of IDFS. *Pure* IDFS interleaves search among all successor subtrees of any internal node at any level of the tree, i.e., all levels are parallel and, for each level, all sub-

[1] IIIA, CSIC, Bellaterra, Spain. pedro@iiia.csic.es. Supported by the Spanish CICYT project TIC96-0721-C02-02.

[2] Department of Computer Science, University of Strathclyde, 26 Richmond Street, Glasgow, Scotland, tw@cs.strath.ac.uk. Also a member of the cross-university APES research group, http://www.cs.strath.ac.uk/ apes/. Supported by EPSRC grant GR/K/65706.

Submitted to ECAI 98
January 22, 1998

Figure 1. Order in which leaf nodes are visited for a complete binary tree of depth 3.

3 DYNAMICITY

DFS starts at the leftmost leaf node in the search tree and moves in an ordered manner to the rightmost leaf node. Within a single iteration of DDS and ILDS, leaf nodes are also explored from left to right. By comparison, within a single iteration of LDS, leaf nodes are explored from right to left. Each iteration of DDS or ILDS can be seen as a simple restriction on the backtracking performed by DFS. In ILDS, we allow backtracking at any interior nodes whose discrepancy count is strictly less than the bound, and prohibit it once the bound is met. In DDS, we allow backtracking at any interior node above the depth bound, and prohibit it at deeper nodes. The pure and limited versions of IDFS are much more dynamic. They jump to leaf nodes both to the left and to the right.

If we ignore the order in which branches are explored *within* each iteration of DDS, then pure IDFS and DDS explore the tree in the *same* order. The only difference is that DDS trades space (pure IDFS uses exponential space) for time (unlike IDFS, DDS revisits interior nodes). It is easy to modify pure IDFS to require linear space, at the cost of revisiting nodes. However, this is very expensive in time. Successive branches searched by pure IDFS share no nodes in common other than the root. We therefore have to backtrack from each leaf node back to the root, and then follow a path down the other side of the tree. For a binary tree of depth n, the cost to search the tree exhaustively goes from $O(2^n)$ to $O(n.2^n)$.

4 HEURISTIC MODEL OF SEARCH

In [14], DDS, ILDS and DFS were compared using Harvey and Ginsberg's abstract model of heuristic search [4, 6]. In this model, nodes in a binary search tree are labelled good if they are above a goal leaf node and bad otherwise. The model has three parameters: m, the mistake probability (the probability that, at a good node, a randomly selected child is bad); p, the heuristic probability (the probability that, at a good node, the heuristic chooses a good node first); and n, the depth of the tree (which is assumed to be uniform). We therefore began our experimental studies by comparing the performance of IDFS on this model.

In Figure 2, as in [6, 14], we plot the probability of finding a goal for $m = 0.2$, $p = 0.95$ and $n = 30$ computed for an ensemble of 100,000 trees. Such trees have approximately a billion leaf nodes, of which just over a million are goals. With about 1 in 1000 leaf nodes being goals, the problems are relatively easy. On these problems, the performance of pure IDFS is very similar to that of DDS. Most of the success of limited IDFS comes from exploring the first branches in the active subtrees. Once we have expanded the first branch within each active subtree, backtracking between active subtrees only slowly increases the probability of success. If we have the space available, it may be more productive to increase the number of active subtrees. However, as we see shall see in our later experiments, we must be careful that the resulting increase in overheads does not outweigh the benefits.

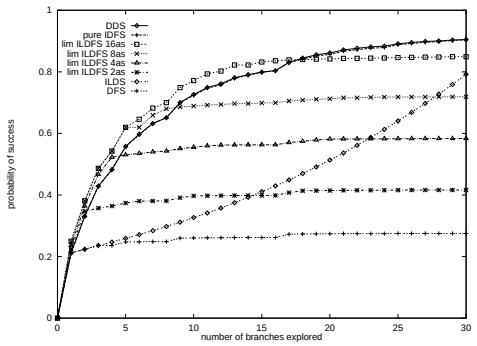

Figure 2. Probability of success on trees of height 30, with $m = 0.2$ and $p = 0.95$. With limited IDFS, "as" is the number of active subtrees.

We next considered a slightly more difficult search problem. In Figure 3, as in [6, 14], we plot the probability of finding a goal for $m = 0.1$, $p = 0.95$ and $n = 100$ computed from an ensemble of 10,000 trees. Such trees have approximately 10^{30} leaf nodes, of which about 1 in 38,000 are goals. DDS and pure IDFS again offered the largest probability of success. As before, increasing the number of active subtrees available to limited IDFS improves performance considerably. On this problem, limited IDFS performed as well as ILDS when the number of parallel levels was increased from 4 to 8.

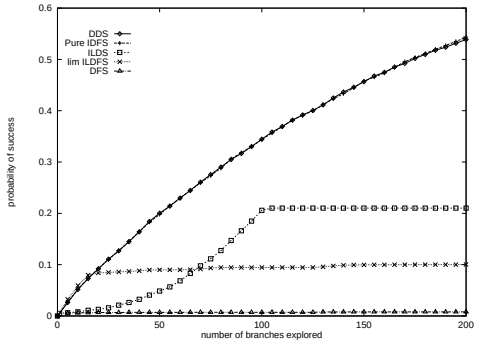

Figure 3. Probability of success on trees of height 100, with $m = 0.1$ and $p = 0.95$. Limited IDFS uses 16 active subtrees.

5 BINARY CSPS

As in [9], our first experimental problem domain is the random binary constraint problem (CSP) class defined by $\langle n, m, p_1, p_2 \rangle$ where n is the number of variables, m is the common cardinality of their domains, p_1 is the connectivity (the ratio between the number of constraints and the maximum number of possible constraints), and p_2 is the constraint tightness (the proportion of forbidden value pairs between two constrained variables). Each problem has exactly $p_1 n(n-1)/2$ constraints, each of which rules out exactly $p_2 m^2$ value pairs. The constrained variables and their nogoods are randomly selected from an uniform distribution [10].

We used a forward checking algorithm (FC), substituting the original DFS strategy with ILDS, DDS and limited IDFS using 2, 4 and 8 active subtrees at the first tree level. As in [9], we use the lowest support heuristic for variable selection and the highest support heuristic for value selection [8]. We solved instances of three qualitatively different classes from sparse 100 variable problems to highly connected 30 variable problems. For each, we have selected 5 different tightnesses to the left of the complexity peak, from very easy to quite difficult problems, and solved one hundred soluble instances of each problem type. Results are presented in Tables 1 to 3 in terms of the mean branches explored and the mean number of visited nodes. This last parameter is correlated with the mean CPU time.

Of the three classes tested, ILDS and DDS dominate DFS on $\langle 100, 6, 305/4950, p_2 \rangle$, very sparse problems with a large number of variables. Heuristic mistakes are more likely to occur in large and very sparsely connected problems, for which the cost of recovering mistakes by DFS is very high even for easy problems. With the other two classes, DFS is not mistaken so frequently and at such high cost. For easy to medium difficulty problems, the three algorithms explore a similar number of branches, which implies that DFS visits a lower number of nodes. With the hard problems, ILDS and DDS make more mistakes than DFS.

IDFS is the best algorithm in this experiment, exploring fewer branches and visiting fewer nodes than DFS for all the problem types, Unlike discrepancy-based algorithms, which do not discriminate between non leftmost nodes at a tree level, limited IDFS keeps the heuristic ranking of nodes and selects active subtrees according to this ranking. Limited IDFS thereby concentrates search on some tree parts, while discrepancy-based algorithms perform a more scattered sampling of the search tree. If the quality of the heuristic is good, this focus appears to pay off compared to the more exploratory strategies like ILDS and DDS.

mean branches $36p_2$	DFS	ILDS	DDS	Limited IDFS 2 as.	Limited IDFS 4 as.
9	1	1	1	1	1
10	2130	1	1	1	1
11	5526	4	2	2	2
12	11764	56	44	19	16
13	56880	1525	8494	1049	470

mean nodes $36p_2$	DFS	ILDS	DDS	Limited IDFS 2 as.	Limited IDFS 4 as.
9	100	102	102	102	102
10	4669	112	116	109	115
11	15687	156	163	126	147
12	37731	874	1112	202	243
13	138454	13627	81496	4858	2358

Table 1. FC algorithm solving random instances of the $\langle 100, 6, 305/4950, p_2 \rangle$ class.

mean branches $144p_2$	DFS	ILDS	DDS	Limited IDFS 4 as.	Limited IDFS 6 as.
44	7	9	7	5	5
46	35	36	26	16	8
48	164	148	199	68	55
50	1030	1558	2063	662	528
52	14445	22620	67741	6977	7161

mean nodes $144p_2$	DFS	ILDS	DDS	Limited IDFS 4 as.	Limited IDFS 6 as.
44	74	162	199	114	132
46	199	507	601	169	156
48	774	1686	3447	402	370
50	4566	13694	25835	3006	2452
52	62031	146339	549928	30249	31070

Table 2. FC algorithm solving random instances of the $\langle 50, 12, 250/1225, p_2 \rangle$ class.

mean branches $225p_2$	DFS	ILDS	DDS	Limited IDFS 4 as.	Limited IDFS 8 as.
69	16	12	11	10	10
72	58	62	47	28	35
75	379	378	381	148	127
78	1584	2061	2334	1049	1175
81	8778	27385	67705	8447	9326

mean nodes $225p_2$	DFS	ILDS	DDS	Limited IDFS 4 as.	Limited IDFS 8 as.
69	89	139	188	95	122
72	259	584	635	172	236
75	1502	2909	3920	644	593
78	6120	12469	18179	4078	4596
81	32678	120507	332768	31562	34867

Table 3. FC algorithm solving random instances of the $\langle 30, 15, 174/435, p_2 \rangle$ class.

6 QUASIGROUPS

Our next problem domain is quasigroup completion. A quasigroup is a Latin square, a n by n multiplication table in which every element appears just once in each row and column. Quasigroup completion is the NP-complete problem of filling in the blank entries in a partially filled Latin square. There are several practical applications for quasigroup completion including the design of statistical experiments to reduce systematic dependencies, and time-tabling (e.g. scheduling a sports tournament). Gomes and Selman have recently proposed quasigroup completion as a benchmark domain for constraint satisfaction algorithms since it has many of the advantages of both random and structured problems [5]. The quasigroup provides a regular structure but this is perturbed by randomly filling some of the entries.

A quasigroup completion problem can be represented as a binary CSP with n^2 variables, each with a domain of size n. As the search tree is $O(n^{n^2})$ in the worst case, it quickly becomes prohibitively to explore the tree completely. Indeed, Gomes and Selman have shown that search costs with DFS are best characterized by a heavy-tailed distribution that has no moments (that is, one with an infinite mean and variance) [5]. Are interleaved and discrepancy based search strategies able to eliminate such heavy-tailed behaviour?

To help answer this question, we solved some quasigroup completion problems with the FC algorithm adapted to search with ILDS, DDS and IDFS. We used the Brelaz heuristic to choose a variable to assign [1], and Geelen's promise heuristic to pick a value for it [3]. At each problem size, we generated 1,000 problems and cut off search if more than 10,000 branches had been explored. In Tables 4 and 5, we report results for

quasigroup completion problems with 30 percent of the initial entries assigned. We obtained similar results with 10 and 20 percent of entries assigned.

n	DFS		ILDS		DDS		Limited IDFS	
	95%	99%	95%	99%	95%	99%	95%	99%
8	6	122	5	13	3	8	8	59
10	63	*	10	24	3	7	4	2826
12	1101	*	14	29	5	8	5	22
14	*	*	24	41	7	11	7	21
16	*	*	31	56	10	16	11	35
18	*	*	43	76	15	24	22	71
20	*	*	59	95	25	40	49	161

Table 4. 95 and 99 percentiles in the branches explored by the FC algorithm to complete a n by n quasigroup with 30 percent of entries pre-assigned. Limited IDFS used 2 parallel levels with n^2 active subtrees. * indicates that the branch limit was reached.

n	DFS		ILDS		DDS		Limited IDFS	
	95%	99%	95%	99%	95%	99%	95%	99%
8	74	717	96	193	84	172	101	545
10	425	*	198	334	177	279	180	21989
12	7654	*	334	678	368	573	361	662
14	*	*	636	1257	696	1115	673	1006
16	*	*	1010	1869	1415	2287	1405	2195
18	*	*	1721	3509	2688	4468	2643	3786
20	*	*	2728	5220	5535	8643	4896	6584

Table 5. 95 and 99 percentiles in the nodes visited by the FC algorithm to complete a n by n quasigroup with 30 percent of entries pre-assigned. * indicates that the branch limit was reached, and over 60,000 nodes are visited.

All the problems in these experiments were soluble. Indeed, most problems were very easy to solve. The median number of branches explored using DFS was typically 1. Nevertheless, DFS failed to solve a considerable fraction of problems within the branch limit. Both ILDS and DDS reduce significantly (if not eliminate completely) the long tail in the distribution of search costs seen with DFS. Limited IDFS also reduces the long tail, although results for the 99 percentile suggest that it may still persist. However, increasing the number of active subtrees reduces (if not eliminates completely) the poor performance in the 99 percentile. We conclude therefore that interleaved and discrepancy based search strategies are all effective at tackling search problems with heavy-tailed distributions.

7 NUMBER PARTITIONING

In [7], Korf compared ILDS, LDS and DFS on number partitioning problems using the CKK algorithm. We thus used number partitioning as the final problem domain in our experimental comparison. Given a bag of n numbers, we wish to find a partition into two bags that minimizes Δ, the difference between the sums of the bags. For problems with perfect partitions (that is, those in which $\Delta \leq 1$), Korf observed that both ILDS and LDS outperformed DFS, with little to choose between them [7]. For problems without perfect partitions, we must traverse the entire search tree to prove optimality. As a consequence, ILDS and DDS search the same number of branches as DFS and limited IDFS, but increase the number of nodes visited. Because LDS re-visits leaf nodes, it does even worse, increasing both the number of branches and the number of nodes visited. We therefore restricted our experiments to under-constrained problems which have a perfect partition and on which discrepancy based strategies may offer advantages. For over-constrained problems which lack a perfect partition, DFS and limited IDFS inevitably search the least number of nodes, and are clearly the search strategies of choice.

In Table 6, we report the median number of branches explored by the CKK algorithm to partition $2n$ numbers drawn at random from $(0, 2^n]$. At each n, we solved 1,000 problems. Results are similar for the mean number of branches, and for other percentiles. On the larger problems, ILDS explores an order of magnitude fewer branches than DFS. This supports the hypothesis underlying ILDS that branching heuristics often make just a few mistakes. On the larger problems, DDS explores even fewer branches than ILDS. This supports the hypothesis underlying DDS that these branching mistakes tend to occur high in the search tree. This is confirmed by the results for IDFS. Increasing the number of active subtrees, which increases the height of some of the discrepancies taken, decreases the number of branches explored.

n	DFS	ILDS	DDS	limited IDFS		
				4 as.	16 as.	64 as.
5	1	1	1	1	1	1
10	1	1	1	1	1	1
15	11	7	3	3	3	3
20	266	29	29	197	99	29
25	5325	443	354	4521	3249	2405
30	105647	6037	5531	94494	84020	69722

Table 6. Median branches explored by the CKK algorithm to partition $2n$ numbers drawn randomly from $(0, 2^n]$.

In Table 7, we report the median number of nodes visited by the CKK algorithm to solve the same problems as in Table 6. Results are similar for other percentiles. The time taken to solve a problem is closely related to the number of nodes visited. Although DDS explores the least number of branches of all the search strategies, in some cases more than an order of magnitude less than DFS, it actually visits the most nodes. How do we explain this?

n	DFS	ILDS	DDS	limited IDFS		
				4 as.	16 as.	64 as.
5	7	7	7	8	8	8
10	17	17	17	18	18	18
15	46	68	81	82	82	82
20	565	472	1007	568	742	1069
25	10691	5287	14553	10526	8117	8033
30	211343	69628	260654	213949	192230	160626

Table 7. Median nodes visited by the CKK algorithm to partition $2n$ numbers drawn randomly from $(0, 2^n]$.

To traverse exhaustively a binary tree of depth n, DDS and ILDS both visit approximately 2^{n+2} nodes compared to the 2^{n+1} nodes visited by DFS [14]. At worst, we might therefore expect DDS and ILDS to double the number of nodes visited. Given that DDS can explore over an order of magnitude fewer branches than DFS, this overhead might appear to be very worthwhile. Unfortunately, the overhead is very unevenly distributed. During the first few iterations, DDS expands many nodes and throws them away immediately. By comparison, DDS is much less wasteful during later iterations, and it is these iterations that dominate the asymptotic analysis given in [14]. The real wastefulness of DDS at the start of search is thus disguised. ILDS is just as wasteful as DDS when traversing

the entire tree. However, its overhead is more evenly distributed. As a consequence, ILDS often explores more branches than DDS but visits fewer nodes.

We observe a similar effect with limited IDFS. Increasing the number of active subtrees decreases the number of branches explored. However, this may have only a small impact on the number of nodes visited. Indeed, in a few cases, the number of nodes visited actually increases (e.g. at $n = 20$). Since each active subtree is expanded down to the bottom of the search tree, increasing the number of active subtrees increases the overhead. As with DDS, this overhead may outweigh much of the benefit of reducing the number of branches explored. In both interleaved and discrepancy based search, we see an interesting tension. On one hand, we want to bias search towards discrepancies high in the search tree as this tends to decrease the number of branches explored. On the other hand, biasing search to discrepancies high in the search tree increases the overhead, offsetting much of the benefit of reducing the number of branches explored. One way to tackle this problem would be to use parallel hardware. For instance, a parallel version of DDS with n processors exploring different branches would expand each branch in essentially constant time, eliminating completely the overhead

8 RELATED WORK

Speckenmeyer *et al.* prove that the expected number of solutions down one branch for random 3-SAT problems at a ratio of clauses to variables of 4 is much greater than down the other branch [12]. They argue that this non-uniform distribution of solutions explains the superlinear speedup observed for parallel backtracking on highly satisfiable problems in [13]. Motivated by this result, they show good performance for a sequential version of the Davis-Putnam procedure which, like IDFS, interleaves search between different subtrees.

Rao and Kumar also report superlinear speedup for parallel backtracking on a variety of different domains including the n-queens problem, the 15 puzzle, and test-pattern generation for digital circuits [11]. They prove that the average speedup is linear when solutions are distributed uniformly, and superlinear when solutions are distributed non-uniformly. They argue that average case superlinear speedup implies that parallel DFS time-sliced on a single processor will dominate sequential DFS. Our results with IDFS confirm that, with care in the choice of the number of active subtrees, such dominance can be achieved in practice.

9 CONCLUSIONS

We have performed a detailed experimental comparison of interleaved and discrepancy based search strategies, using a variety of different domains, and a mixture of random and structured problems. We have shown that interleaved and discrepancy based search strategies often reduce search, most especially on large and lightly constrained problems. In addition, they reduce (if not eliminate completely) the heavy-tailed distribution reported in [5].

Our results highlight a tension common to interleaved and discrepancy based search strategies. Biasing search to discrepancies high in the search tree tends to decrease the number of branches explored, but increases the overhead in expanding branches down to the bottom of the search tree. With under-constrained problems and search strategies like DDS or IDFS with many active subtrees, this overhead can outweigh the benefits. As a consequence, it may be more effective to use a search procedure which explores *more* branches but has smaller overheads (for example, ILDS or limited IDFS with just a few active subtrees).

REFERENCES

[1] D. Brelaz, 'New methods to color the vertices of a graph', *Commincations of ACM*, **22**, 251–256, (1979).

[2] J.M. Crawford and A.B. Baker, 'Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems', in *Proceedings of the 12th National Conference on AI*, pp. 1092–1097, (1994).

[3] P.A. Geelen, 'Dual viewpoint heuristics for binary constraint satisfaction problems', in *Proceedings of the 10th ECAI*, pp. 31–35, (1992).

[4] M. L. Ginsberg and W. D. Harvey, 'Iterative broadening', *Artificial Intelligence*, **55**(2-3), 367–383, (1992).

[5] C. Gomes and B. Selman, 'Problem structure in the presence of perturbations', in *Proceedings of the 14th National Conference on AI*, pp. 221–226, (1997).

[6] W. D. Harvey and M. L. Ginsberg, 'Limited discrepancy search', in *Proceedings of the 14th IJCAI*, (1995).

[7] R. Korf, 'Improved limited discrepancy search', in *Proceedings of the 13th AAAI*, (1996).

[8] J. Larrosa and P. Meseguer, 'Pptimization-based heuristics for maximal constraint satisfaction', in *Principles and Practices of Constraint Programming - CP95*, pp. 103–120, (1995).

[9] P. Meseguer, 'Interleaved depth-first search', in *Proceedings of the 15th IJCAI*, pp. 1382–1387, (1997).

[10] P. Prosser, 'Binary constraint satisfaction problems: Some are harder than others', in *Proceedings of the 11th ECAI*, pp. 95–99, (1994).

[11] V.N. Rao and V. Kumar, 'On the efficiency of parallel backtracking', *IEEE Transactions on Parallel and Distributed Systems*, **4**(4), 427–437, (1993).

[12] E. Speckenmeyer, M. Bohm, and P. Heusch, 'On the imbalance of distributions of solutions of CNF-formulas and its impact on satisfiability solvers', in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science Vol. 35*, ed., Du et al., 669–676, (1997).

[13] E. Speckenmeyer, B. Monien, and O. Vornberger, 'Superlinear speedup for parallel backtracking', in *Proceedings Supercomputing 1987 (ICS'87)*, pp. 985–993. Springer-Verlag LNCS 292, (1987).

[14] T. Walsh, 'Depth-bounded discrepancy search', in *Proceedings of 15th IJCAI*, (1997).