

Limited Discrepancy Search Revisited

PATRICK PROSSER and CHRIS UNSWORTH, Glasgow University

Harvey and Ginsberg's limited discrepancy search (LDS) is based on the assumption that costly heuristic mistakes are made early in the search process. Consequently, LDS repeatedly probes the state space, going against the heuristic (i.e., taking discrepancies) a specified number of times in all possible ways and attempts to take those discrepancies as early as possible. LDS was improved by Richard Korf, to become improved LDS (ILDS), but in doing so, discrepancies were taken as late as possible, going against the original assumption. Many subsequent algorithms have faithfully inherited Korf's interpretation of LDS, and take discrepancies late. This then raises the question: Should we take our discrepancies late or early? We repeat the original experiments performed by Harvey and Ginsberg and those by Korf in an attempt to answer this question. We also investigate the early stopping condition of the YIELDS algorithm, demonstrating that it is simple, elegant and efficient.

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.1 [Discrete Mathematics]: Combinatorics—*Combinatorial algorithms*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Heuristic mistakes, limited discrepancy search, performance improvement

ACM Reference Format:

Prosser, P. and Unsworth, C. 2011. Limited discrepancy search revisited. *ACM J. Exp. Algor.* 16, 1, Article 1.6 (August 2011), 18 pages.

DOI = 10.1145/1963190.2019581 <http://doi.acm.org/10.1145/1963190.2019581>

1. INTRODUCTION

In tree-based search, such as depth first search, performance is heavily dependent on variable and value ordering heuristics. Heuristics advise the search process as to what decision to make next, for example, what variable to consider and what value to assign to that variable. If a bad decision is made early on in a search, a large subtree may be explored before this decision can be reversed. It is commonly believed that heuristics tend to be less reliable at the top of search than deep in search where many decisions have been made and inferencing has taken place. Limited discrepancy search (LDS) [Harvey and Ginsberg 1995] attempts to address this. Initially, the search process goes with heuristic advice, traversing the left branch of the search tree. If this fails, then search is restarted and the process is allowed to take a single discrepancy (i.e., it is allowed to go against heuristic advice at most once, but in all possible ways). If one discrepancy fails to find a solution, then two are allowed in all $\binom{n}{2}$ ways, then three in all $\binom{n}{3}$ ways, and so on, up to a maximum number. In LDS, discrepancies are taken as early as possible, high up in search where it is assumed costly errors have been made.

Author's address: Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland; email: Patrick.Prosser@glasgow.ac.uk.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2011 ACM 1084-6654/2011/08-ART1.6 \$10.00

DOI 10.1145/1963190.2019581 <http://doi.acm.org/10.1145/1963190.2019581>

```

0.  LDS(node,n)
1.    for k = 0 to n
2.    do begin
3.        result = LDSProbe(node,k)
4.        if result != nil
5.        then return result
6.    end
7.    return nil

0.  LDSProbe(node,k)
1.    if isGoal(node) then return node
2.    if failed(node) then return nil
3.    if k == 0
4.    then return LDSProbe(left(node),0)
5.    else begin
6.        result = LDSProbe(right(node),k-1)
7.        if result == nil
8.        then result = LDSProbe(left(node),k)
9.        return result
10.    end

```

Fig. 1. Harvey and Ginsberg's limited discrepancy search (LDS).

When taking k discrepancies, LDS revisits search states with $k - 1$ discrepancies and less. This lead Korf [1996] to propose an *improved* version of LDS, namely ILDS. However, in Korf's description of LDS and ILDS, discrepancies are taken as late as possible, contrary to the motivation behind LDS (and we believe that many subsequent version of LDS have faithfully reproduced this error). Therefore, we ask the following question: Does it matter if we take discrepancies late or early? We put this to the test by replicating Korf's experiments, using ILDS over number partitioning problems, taking discrepancies late and early. We then repeat Harvey and Ginsberg's [1995] experiments over job shop scheduling problems, and the job shop experiments in Karoui et al. [2007], again taking discrepancies late and early. Finally, we examine randomly generated independent set decision problem. In all cases, we restrict ourselves to problems where the decision variables have binary domains, allowing us to preserve the simplicity of LDS and ILDS.

LDS and ILDS perform poorly when problems are unsatisfiable. This is due to the search process performing redundant probes. In Karoui et al. [2007], an early stopping condition was proposed in the YIELDS algorithm. We present this stopping condition in isolation, prove that it is sound, and show how it can be easily incorporated into ILDS. We call this YLDS, a lite version of the YIELDS algorithm. We show empirically that the YIELDS stopping condition has a significant effect on unsolvable problems, with no measurable penalty when problems are solvable.

We now present again LDS and ILDS, and in the section after that, we describe the experiments performed and present our results. We then revisit the YIELDS stopping condition and demonstrate the performance improvements due to this. Finally, we conclude.

2. LIMITED DISCREPANCY SEARCH

Harvey and Ginsberg's [1995] LDS is described in Figure 1. In $\text{LDS}(\text{node}, n)$, the integer n is the maximum number of discrepancies that can be made, and it is typically the number of decision variables, assuming binary domains. The parameter *node* is the

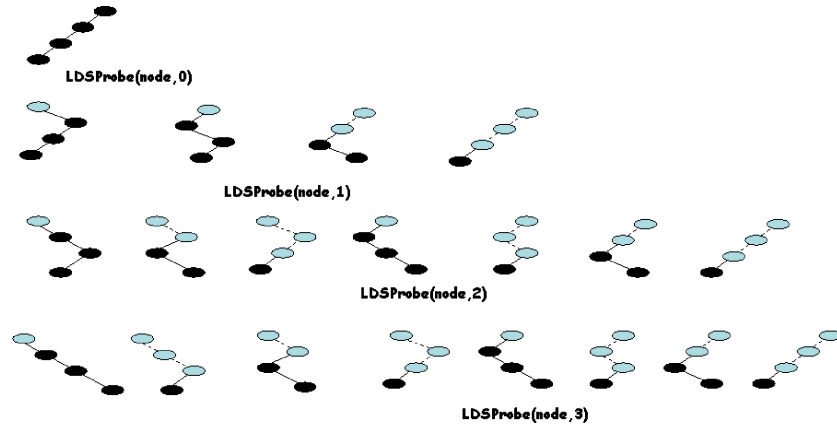


Fig. 2. Cartoon of the LDS search process. Going left with the heuristic, right against.

current search state. LDS then calls `LDSProbe` with increasing number of discrepancies k until a solution is found or maximum discrepancies have been taken. `LDSProbe` makes a limited discrepancy search with k discrepancies. In line 1, if we have reached our goal, then the current state is returned, and in line 2, if the current state cannot be extended, *nil* is delivered. It is assumed that the search can then go left or right. Going left means going with the heuristic, and going right against the heuristic while taking a discrepancy. In `LDSProbe`, the call `left(node)` delivers a new node resulting from going with the heuristic from the current node, and the call `right(node)` delivers a new node resulting from going against the heuristic from the current node. If there are no discrepancies allowed (line 3), search goes with the heuristic (line 4). If discrepancies are allowed (lines 5 to 10), then search takes a discrepancy and goes against the heuristic (line 6), and if this fails, then search does not take a discrepancy and goes with the heuristic (line 8).

The LDS search process is shown graphically in the cartoon of Figure 2, taken from Harvey and Ginsberg [1995]. It is assumed that we have three zero/one decision variables. When going with the heuristic, search goes left, and going right against the heuristic. The broken lines and lightly colored nodes represent decisions that are not backtracked over. The first row corresponds to a call to `LDSProbe` with zero discrepancies, the second row with one discrepancy, the third row with two discrepancies, and the fourth row with the maximum of three discrepancies.

There are a number of points to note about LDS. First, and most significantly, when a discrepancy can be taken, it is taken as soon as possible (i.e., it is taken *early*). Consequently, the first discrepancy is taken at the top of search. This is consistent with the assumption that weak heuristic decisions are made early on in search. Second, a call to `LDSProbe(node, k)` will explore all leaf nodes with k or less discrepancies, consequently LDS re-explores leaf nodes, and this is shown graphically in Figure 2. This redundancy was addressed by Korf's [1996]. improved limited discrepancy search (ILDS). Pseudocode for improved `ILDSProbe` is given in Figure 3.

Korf's `ILDSProbe` (see Figure 3) takes an additional parameter, *rDepth*, the remaining depth over which discrepancies can be taken. In Figure 3 (line 4), the remaining depth is greater than the number of discrepancies k ; consequently, the search delays taking those discrepancies (line 5) and goes with the heuristic. If the probe in line 5 fails or there is insufficient remaining depth to delay taking discrepancies, then search makes a probe with a discrepancy, that is, search goes against the heuristic (line 7). If *rDepth* was not used and the probe of line 5 was performed unconditionally,

```

0.  ILDSProbe(node,k,rDepth)
1.    if isGoal(node) then return node
2.    if failed(node) then return nil
3.    result = nil
4.    if rDepth > k
5.    then result = ILDSProbe(left(node),k,rDepth-1)
6.    if k > 0 && result == nil
7.    then result = ILDSProbe(right(node),k-1,rDepth-1)
8.    return result

```

Fig. 3. Korf's improved limited discrepancy search (ILDS).

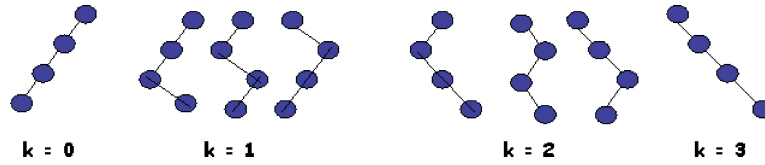


Fig. 4. Paths with 0, 1, 2, and 3 discrepancies. Going left with the heuristic, right against.

search would re-explore states with less than k discrepancies and behave like LDS. Note that when k is zero and no discrepancies are allowed, the condition of line 4 is satisfied and we get a left-only search. LDS becomes ILDS by substituting the call to `LDSPProbe(node,k)` in line 3 of the top half of Figure 1 with the call to `ILDSProbe(node,k,n)`.

Again, there are a number of points of interest in `ILDSProbe`. First, and most obviously, the redundancy in LDS is removed. However, as Korf [1996] notes, if the search process performs constraint propagation, the calculation of the opportunity for future discrepancies may be optimistic and redundancy may creep in. Finally, and most significant for our study, ILDS delays its discrepancies, taking its first discrepancy at maximum depth (i.e., it takes its discrepancies *late*, and this is graphically shown in Figure 4) [Korf 1996]. This is contrary to Harvey and Ginsberg's original motivation for LDS.

Subsequent reported enhancements to LDS, such as Walsh's [1997] depth-bounded discrepancy search, Meseguer and Walsh's [1998] interleaved and discrepancy-based search, and Beck and Perron's [2000] discrepancy-bounded depth first search either take discrepancies in the same order as Korf, or are not specific about the order that discrepancies are taken. Furcy and Koenig [2005] introduce limited discrepancy beam search over nonbinary domains, a search that combines beam search with LDS and takes discrepancies early, but does not incorporate Korf's improvement. Most recently, the YIELDS algorithm [Karoui et al. 2007] addresses nonbinary domains, taking i discrepancies for the i^{th} value in the ordered domain of a variable (the first value being in position zero). YIELDS incorporates a learning scheme that dynamically reorders variable instantiations based on past conflicts and has a simple and elegant early stopping condition when problems are unsolvable. However, YIELDS takes the first discrepancy at maximum depth, the same as ILDS. This then raises the question, should discrepancies be taken late or early, and if it makes no difference, why is that so?

3. EXPERIMENTAL STUDY

We perform three sets of experiments, the first over number partitioning problems (the problem domain studied in Korf [1996]) and the second on job shop scheduling

```

private static void genData(int n,int d){
    Random gen = new Random();
    long x = 1;
    for (int i=0;i<d;i++) x = x * 10;
    for (int i=0;i<n;i++){
        long y = gen.nextLong();
        while (y < 0 || y % x == 0) y = gen.nextLong();
        System.out.println(y % x);
    }
}

```

Fig. 5. Java code to generate n numbers uniformly at random to precision d .

problems (the problem domain studied in Harvey and Ginsberg [1995]). The third set of experiments is over randomly generated independent set decision problems. The purpose of the experiments is to determine if there is an advantage in taking discrepancies early (as in Harvey and Ginsberg [1995]) or late (as in Korf [1996]).

3.1. Number Partitioning

We replicate Korf's [1996] experiments on number partitioning, using ILDS and taking discrepancies late and early. In the number partitioning problem, we are given a multiset (bag) containing n positive integers. The problem is then to partition the bag into two bags such that their sums differ by at most 1. This problem is NP-complete.

This problem can be addressed by incorporating the Karmarkar Karp (KK) heuristic [Karmarkar and Karp 1982] into backtracking search. The KK heuristic works as follows: Initially, the input data is sorted into a list L in nonincreasing order (i.e., largest element first). The first two numbers in the list, X and Y , are removed from the front of the list. There are then two possible choices: (i) insert in order into L the difference $X - Y$, corresponding to placing the numbers in different bags, or (ii) push the sum $X + Y$ onto the front of L , corresponding to placing both numbers in the same bag. Of the two choices, option (i) is preferred (i.e., it is the heuristic choice). There are then three possible outcomes resulting from a choice: (1) $length(L) = 1$ and $head(L) \in \{0, 1\}$, or (2) $head(L) - sum(tail(L)) \leq 1$, or (3) $head(L) - sum(tail(L)) > 1$. In case (1), a perfect partition exists; in case (2), we can continue making choices; and in case (3), no perfect partition exists. In our model, we use the Choco constraint programming toolkit [Choco 2011], and have n 0/1 constrained integer decision variables, v_0 to v_{n-1} , and the list L as a reversible structure. The KK heuristic is encoded as a specialized constraint. If a variable v_i is assigned the value 0, we go with the heuristic, making choice (i), and if assigned the value 1, we go against the heuristic making choice (ii). If this results in outcome (1) search terminates successfully, outcome (2) search proceeds, outcome (3) search fails, and backtracking takes place.

Problem datasets were generated using the Java program segment given in Figure 5. Problem size n was varied from 25 to 100, in steps of 5, and for each value of n 100, problem instances were produced containing n numbers drawn uniformly at random from the range 1 to $10^d - 1$. In replicating Korf's experiments, d was set to 10, and of the numbers generated, about 90% were 10 digits long and about 10% were 9 digits long or less, as expected.

Experiments were run as background jobs and farmed over 10 processors, taking 7 days elapsed time, with most processor time spent on problems in the range $30 \leq n \leq 45$. Since a variety of processors were used, we do not report runtimes. Figure 6(left)

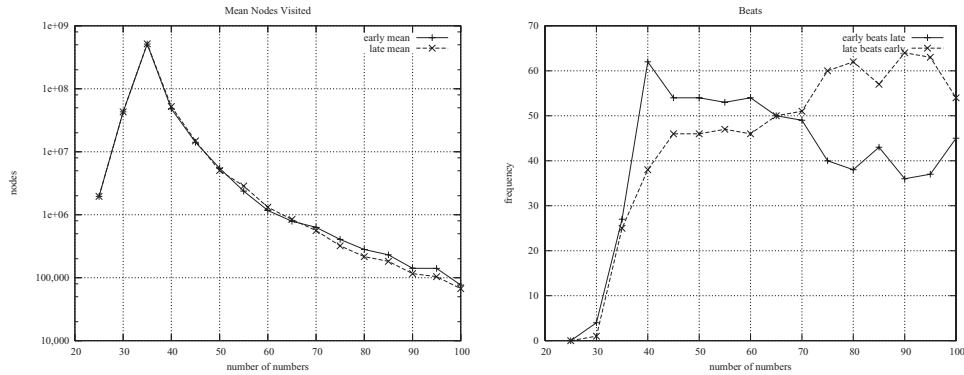


Fig. 6. On the left, log of average search effort (nodes visited) against problem size, solid contour for ILDS-early and broken contour for ILDS-late. On the right, the number of times ILDS-early beat ILDS-late (solid contour) and number of times ILDS-late beat ILDS-early (broken contour) against problem size.

shows, on a log scale, the average number of search nodes¹ explored against problem size n for ILDS taking discrepancies early (solid contour) and late (broken contour).

The contours generally agree with Korf's [1996]. Although not shown, search effort was plotted against constrainedness [Gent and Walsh 1998; Gent et al. 1996] $\kappa = \log_2(l)/n$, where numbers are drawn uniformly and at random from $(0,1)$. The complexity peak occurred at $\kappa = 0.95$ and problem satisfiability about 50%, as expected, that is, where half of our problem instances had perfect partitions.² Figure 6 (right) shows how often ILDS-early beat ILDS-late, and vice versa. If on a problem instance ILDS-early took less nodes than ILDS-late, then ILDS-early scores 1 point, if ILDS-late takes less nodes than ILDS-early, then ILDS-late scores 1 point; and if they both take the same number of nodes, there are no points.

From Figure 6, we can see that when problems are hard ($25 \leq n \leq 60$), it appears that it is better to take discrepancies early, and when problems are easy ($70 \leq n \leq 100$), late. However, Figure 6 shows that when problems are hard, the difference between late and early is relatively insignificant, and that when problems are easy, the absolute difference is insignificant. Therefore, it appears that there is nothing to choose between taking discrepancies late or early, and this raises the question, why should that be?

When using a static variable ordering, ILDS-early and ILDS-late must take the same number of discrepancies, to find a solution or prove that none exists. Furthermore, in a static variable ordering, if search terminates after k discrepancies, then both ILDS-late and ILDS-early must have explored the same states in all probes less than k . Consequently, the difference in the number of nodes visited can only be due to the final probe, and that is of cost $O(\binom{n}{k})$. When problems are unsatisfiable, ILDS must explore all discrepancies and call ILDSProbe $n + 1$ times, with $0 \leq k \leq n$. Each probe attempts to take k discrepancies in all possible ways; therefore, search effort for unsatisfiable instances will be $O(\sum_{k=0}^{k=n} \binom{n}{k})$, and this is $O(2^n)$. Consequently, ILDS-late and ILDS-early should have very similar search efforts when problems are mostly unsatisfiable, and identical search efforts when all instances are unsatisfiable, and this is what we see with $25 \leq n \leq 35$. But what happens when problems are mostly satisfiable?

Figure 7 shows the average number of discrepancies taken by ILDS-late and ILDS-early to find a solution or show that none exists. When problems have solutions ($n \geq 40$), we see that the number of discrepancies falls with problem size, in our instances, from

¹A node corresponds to a decision made by the search process, that is, assigning a value to a variable.

²Table V tabulates κ against n with $l = 10^{10} - 1$, that is, the largest integer allowed in our datasets.

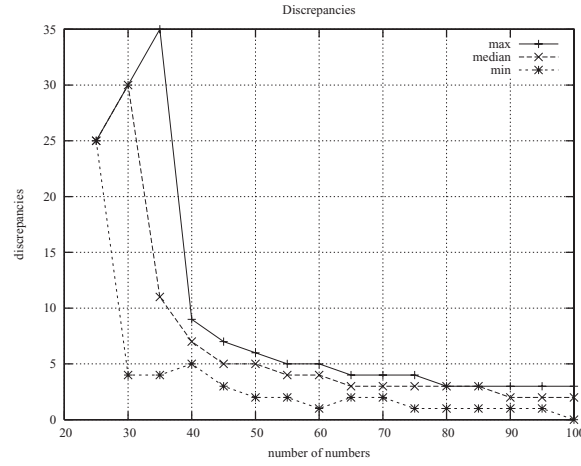


Fig. 7. The number of discrepancies taken by ILDS against problem size.

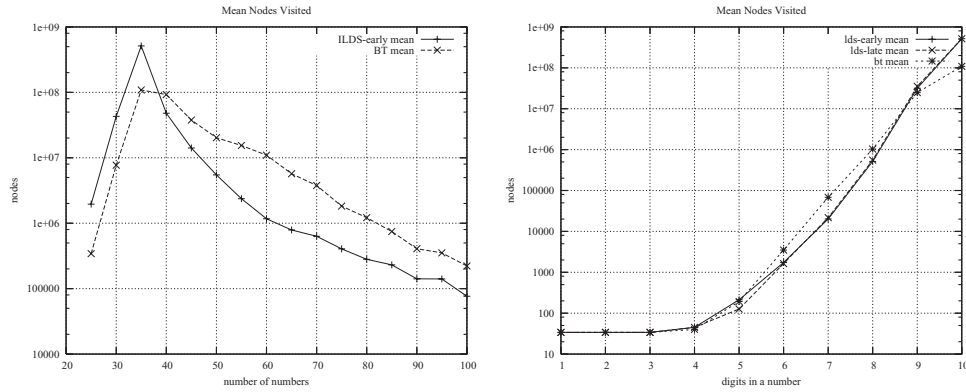


Fig. 8. On the left, log of average number of nodes against problem size: ILDS-early (solid contour) and BT (broken contour). On the right, log of average number of nodes against number of digits in a number, for BT and LDS late and early.

about seven discrepancies at $n = 40$ down to two when $n = 100$. The data was analyzed to determine how much search was devoted to this last probe. The percentage of search effort in the last probe (for late and early) at $n = 40$ was approximately 44%, steadily increasing to about 76% when $n = 100$. This came as a surprise. Although $O(\binom{n}{k})$ increases up to the point where $k = n/2$, we were expecting that the last probe would be relatively small when problems were satisfiable and hard $40 \leq n \leq 60$, and most search effort would be attributed to the sum of the previous probes. For our number partitioning problems, this was not the case, and the majority of search was in the last probe. We tabulate this data in Table V, along with all of our number partition results.

Therefore, in our number partitioning experiments, it appears that it does not matter if we take discrepancies early or late, but this must be put in context. If it makes no difference if we take discrepancies early or late—does that mean we should give up on discrepancy-based search? We, therefore, compare ILDS-early against chronological backtracking (BT) over the same datasets (Figure 8). The left of Figure 8 shows that in the region where problems are mostly unsatisfiable ($25 \leq n \leq 35$) chronological BT is almost one order of magnitude better than ILDS, but when instances

are satisfiable ($n \geq 40$), ILDS is the algorithm of choice, and this agrees remarkably well with the results in Korf [1996].

We also performed experiments holding n constant and varying the number of digits in numbers. This allows us to vary problem constrainedness while holding the problem size constant, where we measure problem size as the number of decision variables. This is shown on the right side of Figure 8. We again see that when problems are hard (9- and 10-digit numbers), BT is best, but when problems are satisfiable and easy, ILDS is the algorithm of choice. We should note that we get 1 point where ILDS-late appears to be significantly better than ILDS-early (five-digit numbers). This is due to a single data point skewing the mean, and in fact, ILDS-early and ILDS-late are not significantly different in this set of experiments.

3.2. Job Shop Scheduling Problems

Harvey and Ginsberg [1995] demonstrated the performance of LDS on job shop scheduling problems (JSSP), and we now do the same using ILDS-early and ILDS-late. In the JSSP, we are given a set of n jobs and a set of m resources. Each job is made up of a sequence of m operations, where each operation has an uninterrupted processing time on a single resource. Each job, and consequently each operation in a job, has a release date and a due date. Resources can only perform one operation at a time. The goal is then, given an overall completion time (i.e., make span), that all operations of jobs be sequenced on resources such that the make span is met, and there is an optimization variant where the make span is minimized.

Experiments were performed on 15 of the Lawrence [1984] JSSP instances, la01 to la15, available at ORLIB. The instances la01 to la05 are 10×5 (i.e., 10 jobs and 5 resources), la06 to la10 are 15×5 , and la11 to la15 are 20×5 . Further experiments were then performed over 40 of Sadeh's [1991] job shop satisfaction problems, each being 10 jobs and 5 resources with varying bottleneck resources. The scheduling problems were represented as a disjunctive graph, using the Choco constraint programming toolkit. That is, for a job shop instance with n jobs and m resources, there are $m \cdot n(n-1)/2$ zero/one variables to decide the order of operations on resources and $(n \cdot m + 1)$ bound integer variables to represent operation start times. Therefore, we have two distinct sets of variables: the set of zero/one variables that control disjunctive precedence constraints between pairs of operations that share a resource and the set of start times attached to operations. The slack-based heuristics of Smith and Cheng [1993] were implemented as a dynamic variable ordering heuristic and as a dynamic value ordering heuristic. That is, given a decision variable d_{ij} associated with an ordering between the pair of operations op_i and op_j on a shared resource, a slack value was computed for d_{ij} as the maximum slack remaining in the start times for operations op_i and op_j if op_i was sequenced before op_j or if op_j was sequenced before op_i . The zero/one decision variables were then ordered dynamically in nondecreasing order of slack, and values were ordered dynamically in nonincreasing order of slack. Consequently, an attempt is made to instantiate the most constrained variable with its least constraining value. After each decision, the search process establishes arc-consistency,³ that is, constraint propagation takes place through process plans, updating operation start times, and possibly enforcing sequencing decisions. Making a decision against the value ordering heuristics counts as a discrepancy. At the start of each probe, the decision variables

³Arc-consistency is at the heart of constraint programming. Arc-consistency is a polynomial-time process that filters out from the domains of variables values that cannot occur in any solution. Typically, arc-consistency is established after the instantiation of a variable. The first arc-consistency algorithms were proposed by Mackworth [1977]. For more details, see Rossi et al. [2006].

Table I. Lawrence Job Shop Scheduling Instances, la01 to la15

Instance	make span	ILDS-early			ILDS-late			BT	
		nodes	disc	time	nodes	disc	time	nodes	time
la01	666	42	0	0.05	42	0	0.05	42	0.04
la02	655	2,648	3	0.43	5,248	3	0.60	35,132	1.7
la03	597	53,552	6	4.1	42,345	6	3.3	6,103	0.67
la04	590	1,798	3	0.38	2,431	3	0.44	310	0.21
la05	593	91	0	0.06	91	0	0.06	91	0.05
la06	926	958	1	0.36	306	1	0.17	—	—
la07	890	3,660	2	1.1	8,024	2	2.5	1,044,950	41
la08	863	5,794	1	1.6	2,409	1	0.71	5,17,990	20
la09	951	760	1	0.32	6,616	1	1.6	—	—
la10	958	1,045	1	0.34	485	1	0.20	39,201,106	1,294
la11	1,222	2,090	1	2.0	757	1	0.79	—	—
la12	1,039	36,987	2	32	22,096	2	19	—	—
la13	1,150	4,117	1	3.4	14,669	1	9.4	—	—
la14	1,292	1,352	1	1.1	11,142	1	6.2	399	0.13
la15	1,207	11,1067,002	4	6743	7,194,189	3	431	—	—

Minimum make span (Second column) is posted as a constraint resulting in a decision problem. Tabulated is number of decisions (nodes), discrepancies taken, and runtime in seconds. Slack-based dynamic variable and value ordering heuristics were used. Results are reported for chronological backtracking (BT). The best results between ILDS-early and ILDS-late are in **bold**. A table entry of—corresponds to search termination after 100 million nodes (although 200 million nodes were allowed for la15).

are returned to index order. It should also be noted that we have kept our model relatively simple and that we did not exploit edge finding constraints [Vilim 2009].

3.2.1. Lawrence Benchmarks. The results of our experiments are shown in Table I. The Lawrence job shop instances were posed as decision problems, that is, the minimum make span (tabulated) was posted as a constraint, and the search process finds an ordering of operations on resources that satisfied that constraint. Table I shows the number of decisions made by the search process (nodes), the number of discrepancies required to find a solution, and the runtime in seconds.⁴ All experiments were done in a *safe mode*. That is, each decision variable was initialized with a unique identification number and prior to applying the heuristic to select the i^{th} variable, all uninstantiated variables were sorted with respect to their identification number. That way, there is no residual effect caused by previous variable orderings when backtracking. Unfortunately, this is costly to do and was suppressed for our hardest instance la15.

We also tabulate results for chronological backtracking (BT). BT was allowed 100 million nodes before search was terminated (tabulated as —) and 200 million nodes on la15. The most obvious thing of note is that ILDS is generally much faster than BT. However, what is less obvious is that ILDS-early and ILDS-late do not always take the same number of discrepancies to find a solution (see instance la15), and this is due to the dynamics of the variable ordering heuristic.

THEOREM 3.1. *The order of instantiation of variables can influence the number of probes required to find a solution.*

PROOF. We use an existence proof. Assume we have a problem with three constrained integer variables $x \in \{1, 3\}$, $y \in \{1, 2\}$, and $z \in \{1, 2\}$ with the constraints $x \neq y$, $x \neq z$, and $y \neq z$. Furthermore, assume that the value ordering heuristic selects the first value in the domain of a variable, that after instantiation of a variable, arc-consistency is established [Rossi et al. 2006], and that the current probe has exhausted

⁴Note that all experiments were run on the same processor so we can reliably compare run times.

its discrepancies. The instantiation order (x, y, z) will fail, that is, x will take the value 1 and constraint propagation will remove that value 1 from the domains of y and z , y is then instantiated to 2 and propagation wipes out the domain of z , and search fails. Another probe will be required with an additional discrepancy, such that x takes the value 3, y the value 1, and z the value 2. However, the instantiation order (z, y, x) will succeed, with z taking the value 1, y the value 2, and x the value 3. \square

Clearly, the previously mentioned gadget (the mini-problem used in the proof) can be incorporated into any ILDS probe, where the variables x , y , and z are the last variables in the instantiation order, and these variables are reached when discrepancies have been exhausted. But this raises the question, how can the dynamic instantiation order change by going left and then right (i.e., take a discrepancy late) compared with going right and then left (i.e., take a discrepancy early)? This is due to the residual effect of our dynamic variable ordering. To select the variable with least slack, we compare decision variable d_i with d_j (where $i < j$), and if the slack associated with d_j is less than the slack of d_i , we swap d_i with d_j and then continue, comparing d_i with d_{j+1} , eventually terminating when $j = n - 1$. This residual disorder can then influence the subsequent search effort. Our *safe mode* avoids this, but it should only be used for empirical comparisons.

Returning to Table I, we note that there are six problems where ILDS-early is faster than ILDS-late (la02, la04, la07, la09, la13, la14) and seven problems where ILDS-late is faster than ILDS-early (la03, la06, la08, la10, la11, la12, la15). However, only la015 produces a significant difference in runtimes between ILDS-late and ILDS-early. Problem la15, is of size 20×5 and has 950 zero/one variables. ILDS-early took one more discrepancy than ILDS-late to solve la15, and this resulted in ILDS-early taking nearly 2 hours of CPU time and ILDS-late taking just over 7 minutes. But this is a single *unsafe* instance, and Table I shows no clear trend and no clear winner. That is, it appears that there is little to choose between taking discrepancies early and taking discrepancies late.

3.2.2. Sadeh Benchmarks. Experiments were performed over Sadeh's [1991] job shop scheduling satisfaction problems. There are four classes of instance: e0ddr1, e0ddr2, enddr1, and ewddr2.⁵ Each instance has 10 jobs and 5 resources, and individual jobs within an instance have specified release and due dates. The problem is then to satisfy resource and temporal constraints. The same model and slack-based heuristics were used in these experiments. Search nodes are tabulated for each instance in Table II, comparing ILDS-early against ILDS-late, both using the dynamic slack-based variable and value ordering heuristics. Experiments were performed on the same machine as described earlier, again in *safe mode*. A table entry in **bold** corresponds to a search that took at least one discrepancy, and a nonbold entry corresponds to a search that took no discrepancies, that is, the search process find a solution without making a mistake, and ILDS-early and ILDS-late take the same number of nodes. Only 6 of the 40 instances require discrepancies to be taken, and in 5 of those instances, ILDS-early beats ILDS-late. However, the relative difference is only significant in 3 of the 40 cases (e0ddr1-3, e0ddr1-8, enddr1-8), but with regard to runtimes, the differences are insignificant.

Experiments were then performed over the Sadeh benchmarks to explore the sensitivity of ILDS-early and ILDS-late to the slack-based heuristics (i.e., what would be the effect of turning the value and variable ordering heuristics off and on?). For each

⁵NOTE: These are the *complete* sets available from Sadeh's Web site, each complete set containing 10 instances. These instances should not be confused with the set at Christophe Lecoutre's benchmark page [Roussel and Lecoutre 2008]. Roussel and Lecoutre's instances are binary constraint satisfaction representations of Sadeh's originals, therefore some of the structural information is lost in translation.

Table II. Sadeh's Job Shop Scheduling Satisfaction Problems

id	e0ddr1		e0ddr2		enddr1		ewddr2	
	early	late	early	late	early	late	early	late
1	111	111	108	108	122	122	143	143
2	137	137	1,175	1,231	133	133	75	75
3	222	1,282	106	106	110	110	123	123
4	125	125	144	144	105	105	141	141
5	128	128	141	141	136	136	104	104
6	141	141	141	141	154	154	153	153
7	101	101	104	104	107	107	147	147
8	220	801	197	224	161	445	108	108
9	144	144	78	78	158	158	97	97
10	297	148	123	123	114	114	156	156

There are four complete classes (e0ddr1, e0ddr2, enddr1, ewddr2) with 10 instances in each class. Tabulated are nodes explored to find a solution. The **bold** entries correspond to instances where search took discrepancies. In all other cases, solutions were found merely by following the slack-based heuristic.

instance, we have eight possibilities: (take discrepancies early **or** late) **and** (use the dynamic slack-based value ordering **or** the static ordering 0 then 1) **and** (use the dynamic slack-based variable ordering **or** the static index order of the zero/one decision variables). The results of these experiments are shown in Table III, with a table entry of “–” corresponding to a trivial instance where no discrepancies were taken and ILDS-early and ILDS-late take the same number of search nodes.

Looking at Table III columns E-1-1 and L-1-1 (early versus late, using the dynamic value and variable ordering heuristic, essentially Table II), we see that absolute differences tend to be small, measured in hundreds of nodes. In fact, over all instances, late and early, runtimes were in the range 52 to 271 milliseconds. Looking at columns E-1-0 and L-1-0 (using the dynamic value ordering, but selecting variables in index order) ILDS-late is terrible, suggesting that value ordering heuristic error is significant high up in the search tree close to the root when we disable variable ordering and that taking discrepancies late is costly. Columns E-0-1 and L-0-1 (selecting value 0 before value 1, but using dynamic variable ordering) shows that there is a less reliable distinction between ILDS-early and ILDS-late, with ILDS-late being significantly worse on two instances (e0ddr1-3, e0ddr2-1). The final columns, E-0-0 and L-0-0, should be considered as a “straw man” where no heuristic information is exploited. We can see the result is a lottery with ILDS-late and ILDS-early both performing atrociously, and our choice is then between the lesser of two evils. In conclusion, Table III suggests that, over Sadeh's benchmarks, the slack-based value ordering heuristic in isolation (columns *-1-0) is less reliable at top of search than deep in search and that taking discrepancies early is preferable to late. When we ignore value ordering heuristic information (columns *-0-*), there is little to choose between late and early, as both perform equally poorly, and when we exploit all available heuristic information (columns *-1-1) again, there is little to choose between late and early, as they both behave extremely well. Therefore, it appears that the only time the choice between late and early is significant (over Sadeh's benchmarks) is when we have heuristic information but it is unreliable, and this supports the LDS hypothesis.

3.3. Independent Set

We now present results on randomly generated problems, similar to experiments on random binary constraint satisfaction problems [Prosser 1996; Gent et al. 2001]. We choose independent set because we can model this exclusively with binary variables and do not have to complicate the algorithms with the burden of dealing with high

Table III. Sadeh's Job Shop Scheduling Satisfaction Problems

Instance	E-1-1	L-1-1	E-1-0	L-1-0	E-0-1	L-0-1	E-0-0	L-0-0
e0ddr1-1	—	—	788	1,722	201	98	448	200
e0ddr1-2	—	—	—	—	267	150	3968	1427
e0ddr1-3	222	1,282	4,190	64,855	6,06,229	15,63,201	3,58,049	4,38,228
e0ddr1-4	—	—	222	2,664	207	93	3,015	2,140
e0ddr1-5	—	—	—	—	—	—	743	445
e0ddr1-6	—	—	—	—	—	—	1,824	2,612
e0ddr1-7	—	—	231	1,157	—	—	239	114
e0ddr1-8	220	801	—	—	—	—	697	158
e0ddr1-9	—	—	254	1,383	—	—	698	123
e0ddr1-10	297	148	—	—	123	125	190	616
e0ddr2-1	—	—	8,782	7,359	1,326	8,997	4,56,864	1,00,969
e0ddr2-2	1,175	1,231	1,76,616	5,67,904	94	256	—	—
e0ddr2-3	—	—	—	—	96	96	—	—
e0ddr2-4	—	—	281	1,246	—	—	174	306
e0ddr2-5	—	—	—	—	—	—	—	—
e0ddr2-6	—	—	243	3,752	—	—	2,386	2,164
e0ddr2-7	—	—	6,415	23,015	—	—	1,214	166
e0ddr2-8	197	224	—	—	120	76	—	—
e0ddr2-9	—	—	312	2,004	5,911	4,226	56,991	10,976
e0ddr2-10	—	—	—	—	—	—	—	—
enddr1-1	—	—	—	—	—	—	9,581	31,246
enddr1-2	—	—	—	—	—	—	234	752
enddr1-3	—	—	—	—	1,007	534	685	1,458
enddr1-4	—	—	281	1,694	—	—	39,468	1,24,717
enddr1-5	—	—	—	—	—	—	—	—
enddr1-6	—	—	—	—	—	—	1,625	1,143
enddr1-7	—	—	—	—	—	—	3,179	2,182
enddr1-8	161	445	308	2254	929	491	3,488	8,128
enddr1-9	—	—	—	—	—	—	351	212
enddr1-10	—	—	—	—	239	772	2,521	13,478
ewddr2-1	—	—	—	—	—	—	600	1,162
ewddr2-2	—	—	247	4,971	—	—	12,025	3,879
ewddr2-3	—	—	—	—	118	131	613	957
ewddr2-4	—	—	—	—	—	—	—	—
ewddr2-5	—	—	—	—	—	—	1,946	254
ewddr2-6	—	—	655	4,658	—	—	1,097	211
ewddr2-7	—	—	227	4,776	—	—	2,882	2,776
ewddr2-8	—	—	—	—	93	112	8,800	2,258
ewddr2-9	—	—	—	—	—	—	61,419	5,69,337
ewddr2-10	—	—	—	—	123	161	1,397	661

Columns E-*. * are ILDS-early.

Columns L-*. * are ILDS-late.

Columns *-1-* use slack-based value ordering.

Columns *-0-* use the static value ordering, select 0 then select 1.

Columns *-*-1 use the slack-based variable ordering.

Columns *-*-0 select variables in index order.

A table entry of - signifies a trivial instance solved with zero discrepancies (see Table II).

arity domains and the difficulties associated with attributing discrepancies to values deep in those domains (as in Karoui et al. [2007] and Furcy and Koenig [2005]).

We are given a simple undirected graph $G = (V, E)$ and an integer k , and the problem is to determine if there is an independent set of size k or more in G , where a set of vertices $V' \subseteq V$ is an independent set if $\forall_{\{i,j\} \in V'} : \{i, j\} \notin E$, where E is the set of edges in G . The independent set decision problem is NP-complete.

The problem is again modeled in Choco. For each vertex v_i , we have a zero-one constrained integer variable x_i . If the variable x_i takes the value 1, then the vertex v_i is selected, and if x_i takes the value 0, the corresponding vertex is rejected. For every

Table IV. Random Independent Set

problem	count	ILDS-early		ILDS-late	
		mean	max	mean	max
$\langle 40, 0.20, 12 \rangle$	35	4,684	83,980	10,932	2,49,580
$\langle 40, 0.20, 13 \rangle$	45	29,259	5,85,202	92,563	2,097,341
$\langle 40, 0.20, 14 \rangle$	29	10,750	1,41,903	14,373	1,31,707
$\langle 40, 0.30, 9 \rangle$	22	3,060,207	67,143,322	9,473,309	2,08,197,948
$\langle 40, 0.30, 10 \rangle$	65	80,743	1,667,078	1,29,100	1,939,869
$\langle 40, 0.30, 11 \rangle$	34	66,060	9,78,348	1,48,775	1,691,877
$\langle 50, 0.15, 16 \rangle$	41	4,32,339	13,953,481	1,733,097	62,287,221
$\langle 50, 0.15, 17 \rangle$	60	7,838,786	4,08,722,788	10,139,148	4,81,195,132
$\langle 50, 0.15, 18 \rangle$	40	1,099,694	35,821,344	1,814,667	60,617,755

Given a random graph $G_{n,p}$ and an integer k , is there an independent set of size k or more? Only counted are instances, out of 100, where there was an independent set of size k or more and search took discrepancies.

edge $\{i, j\} \in E$, we constrain $x_i + x_j \leq 1$. The instance is solvable if $\sum_{i=1}^n x_i \geq k$, where $n = |V|$. The decision variables x are sorted into nondecreasing degree order such that the first variable to be instantiated corresponds to the vertex of lowest degree and the last variable instantiated corresponds to the vertex of maximum degree. This ordering is done at the top of search and is used as a static variable ordering. A static value ordering is also used such that the value 1 is considered as the heuristic choice, selecting a vertex of low degree and as a consequence rejecting a relatively small number of adjacent vertices. The value 0 is then a discrepancy corresponding to the rejection of a vertex of low degree.

All experiments were run on a machine with 8 Intel Xeon E5420 processors running at 2.50GHz, 32GB of RAM, with version 5.2 of linux. Experiments were performed over Erdos-Renyi random graphs, $G_{n,p}$, where n is the number of vertices and p is the edge probability. We present results for instances where search required discrepancies and those instances where solvable. The reason for this restriction is that if an instance can be solved with no discrepancies, then ILDS-late and ILDS-early must take exactly the same path to a solution and must have identical costs. Similarly, if an instance is insoluble, ILDS-early and ILDS-late will take n discrepancies to prove insolubility, and since a static variable ordering is used, it will take the same number of search nodes. Therefore, we exclude these instances from our results in Table IV. We classify problems with the triple $\langle n, p, k \rangle$, where n is the number of vertices, p is edge probability and, k is the size of the independent set. One hundred graphs were generated at each value of n and p . In Table IV, the column headed *count* gives the number of graphs that meet our restriction of solubility and discrepancies taken. These instances are soluble instances close to the phase transition, where we expect to find hard instances [Cheeseman et al. 1991; Prosser 1996; Gent et al. 1996].

In almost all cases, the mean is dominated by the maximum (maximums of hundreds of millions of nodes in some cases). Nevertheless, looking at the raw data, it is evident that even if we ignore these hard instances, ILDS-late is consistently worse than ILDS-early. This suggests that early heuristic mistakes are costly and should be corrected quickly. Independent set appears to fit Harvey and Ginsberg's initial hypothesis.

3.4. Summary of Results

In number partitioning, it did not matter if we took discrepancies late or early. In the Lawrence job shop problems, it was the same, with little to choose between ILDS-late and ILDS-early. Sadeh's benchmarks showed that when heuristic information was very good or very bad, it did not make any difference if we took discrepancies late or early. It was only when we had incomplete and unreliable heuristic information that it really

mattered, and in that case, we should take discrepancies early, as recommended by Harvey and Ginsberg. In the random independent set problems, ILDS-early was the algorithm of choice, and this might suggest that our heuristics were indeed unreliable high up in search.

4. IMPROVING PERFORMANCE ON UNSATISFIABLE INSTANCES

In Section 3, we have seen that chronological backtracking dominates limited discrepancy search when problems are unsatisfiable. One of the reasons for this is that LDS and ILDS repeatedly probe the search space with discrepancies k , from 0 to n , to prove unsatisfiability, but this is unnecessary. Assume a probe is made with a quota of k discrepancies and this returns *nil*, and that the next probe with a quota of $k + 1$ discrepancies also delivers *nil*, but that probe never managed to take more than k discrepancies. In that case, the problem is unsatisfiable and search can terminate.

This observation was made in the YIELDS algorithm of Karoui et al. [2007]. To quote from their paper (page 103): “*In contrast, if the allowed discrepancies are not consumed, it is not necessary to continue to reiterate LDS with a greater number of discrepancies even if no solution has been found.*” This is incorporated into the YIELDS algorithm via detecting that “*The process of learning comes to an end*”, that is, a YIELDS-iteration terminated and the weight vector used in learning was not updated. The YIELDS stopping condition is both simple and elegant, and we now prove that is correct, show how it can be easily incorporated into LDS and ILDS, and empirically investigate the benefits to be had from this.

THEOREM 4.1. *If a probe terminates without consuming its quota of discrepancies, the problem is unsatisfiable and search can terminate.*

PROOF. Assume that a probe with a quota of $k - 1$ discrepancies returns *nil*. Furthermore, assume that the next probe with a quota of k discrepancies also returns *nil* but never manages to take more than $k - 1$ discrepancies. All subsequent probes with a quota of δ discrepancies, for $k < \delta \leq n$, will also be unable to take more than $k - 1$ discrepancies and, therefore, must also return *nil*. Consequently, we can terminate search after a probe if that probe returned *nil* and failed to take its full quota of discrepancies. \square

We now present YLDS, a simplified version of the YIELDS algorithm. YLDS is based on ILDS, takes discrepancies early, deals only with binary domains, and incorporates the YIELDS early stopping condition. To describe YLDS, we introduce a global boolean variable *adt*, for *all discrepancies taken*. The variable *adt* is set to false in the calling procedure YLDS, line 3. Procedure Probe sets *adt* to true if the full quota of k discrepancies have been consumed, in line 3 of Probe. On returning from the call to Probe, in line 5 of YLDS, if a solution has been found (*result* \neq *nil*) or the Probe failed to take all discrepancies, search terminates.

If we allow a probe to set *adt* to true unconditionally, YLDS then behaves as ILDS-early. Furthermore, if a problem instance is satisfiable, YLDS will behave identically to ILDS-early. That is, we only see our improvement on unsatisfiable instances, and that is where LDS and ILDS perform badly. YLDS was applied to the number partitioning problems of Section 3. Table V shows the performance of all the algorithms over these problems. We show, in thousands, the average number of nodes visited to find a solution or show that none exists and in brackets the average number of discrepancies taken. We also show the percentage of search effort devoted to the last probe in YLDS, the percentage of instances that were satisfiable, and the measure of constrainedness κ . It should be noted that, as expected, YLDS performs identically to ILDS-early over the satisfiable instances $n \geq 40$. YLDS should be compared against ILDS-early. When all

Table V. Average Search Effort in Thousands of Nodes, for the Number Partitioning Problems

n	ILDS-early	ILDS-late	BT	YLDS	last probe	% sat	κ
25	1,959 (25)	1,959 (25)	342	1,252 (9)	21%	0	1.329
30	42,958 (28)	42,986 (28)	7,714	29,482 (11)	18%	5	1.107
35	5,13,745 (21)	5,16,082 (21)	1,08,759	3,86,013 (11)	26%	52	0.949
40	48,087 (6)	52,291 (6)	92,720	48,087 (6)	44%	100	0.830
45	14,100 (5)	14,820 (5)	37,674	14,100 (5)	57%	100	0.738
50	5,467 (4)	5,040 (4)	20,281	5,467 (4)	60%	100	0.664
55	2,374 (4)	2,865 (4)	15,406	2,374 (4)	60%	100	0.604
60	1,171 (3)	1,319 (3)	10,984	1,171 (3)	67%	100	0.554
65	788 (3)	841 (3)	5,705	788 (3)	65%	100	0.511
70	631 (3)	561 (3)	3,778	631 (3)	69%	100	0.475
75	405 (2)	321 (2)	1,824	405 (2)	73%	100	0.443
80	281 (2)	216 (2)	1,216	281 (2)	71%	100	0.415
85	230 (2)	182 (2)	746	230 (2)	65%	100	0.391
90	141 (2)	116 (2)	408	141 (2)	71%	100	0.369
95	141 (2)	104 (2)	352	141 (2)	74%	100	0.350
100	76 (2)	67 (2)	222	76 (2)	76%	100	0.332

In brackets, we have average number of discrepancies. Column “last probe” gives the average percentage of search effort devoted to the last probe for YLDS (and ILDS-early when $n \geq 40$). Column “%sat” is percentage of instances at that size that were satisfiable, and last column is kappa for number partitioning [Gent and Walsh 1998].

```

0.  YLDS(node,n)
1.    for k = 0 to n
2.      do begin
3.        adt = false;
4.        result = Probe(node,k,n)
5.        if result != nil || !adt
6.          then return result
7.        end
8.      return nil

0.  Probe(node,k,rDepth)
1.    if isGoal(node) then return node
2.    if failed(node) then return nil
3.    if k == 0 then adt = true;
4.    result = nil
5.    if k > 0
6.      then result = Probe(right(node),k-1,rDepth-1)
7.    if rDepth > k && result == nil
8.      then result = Probe(left(node),k,rDepth-1)
9.    return result

```

Fig. 9. YLDS: improving performance on unsatisfiable instances by stopping when the required number of discrepancies k cannot be taken.

problems are unsatisfiable, YLDS shows a clear advantage over ILDS-early, being about 36% faster when $n = 25$. When instances are a mix of satisfiable and unsatisfiable, YLDS is about 31% faster at $n = 30$ and 25% faster when $n = 35$, where percentage satisfiable is 52%. The reason for these gains is due to the reduction in discrepancies required to prove unsatisfiability (shown in brackets): Typically, YLDS uses less than half of the discrepancies required by ILDS. On the satisfiable instances at $n = 35$, on average, eight discrepancies were taken, and 70,237 nodes were visited by both ILDS-early and YLDS. On the unsatisfiable instances at $n = 35$, ILDS-early took all

35 discrepancies and, on average, 994,213 nodes, whereas YLDS took, on average, 14 discrepancies and 728,103 thousand nodes, about a 27% speed-up.

YLDS was also applied to the Lawrence job shop scheduling problems, where the goal was to prove optimality. That is, the problem is to show that no schedule exists with a make span less than the optimal (i.e., the search process must show that the problem is unsatisfiable). Unfortunately, these problems are far too hard for limited discrepancy search and our relatively simple model. Considering instance la01, our easiest instance, ILDS would require 225 probes (the number of decision variables) to prove unsatisfiability, and we expect this would correspond to an astronomical amount of search effort. However, YLDS, proved optimality of la01 in 383 million nodes, taking 29 discrepancies, and 2 hours and 45 minutes of CPU time (but this compares poorly with BT's 29.6 million nodes and 11 minutes CPU time). YLDS was used in all the random independent set decision problem experiments in Table IV. Without YLDS, proof that an instance was unsatisfiable would take hours, and in some cases, days. YLDS reduced these runtimes to manageable amounts. Furthermore, in all of our experiments, when problems were solvable, the difference in runtime between YLDS and ILDS was minuscule (i.e., the YIELDS stopping condition incurs negligible overhead).

5. CONCLUSION

In moving from LDS to ILDS, we have inadvertently lost the assumption underpinning limited discrepancy search (i.e., that costly heuristic errors are made early on in the search process). We have put this assumption to the test with two variants of ILDS, one taking discrepancies early and one taking discrepancies late. In our number partitioning experiments, we see clear regions where early beats late, and that is when problems are hard and satisfiable, suggesting that Harvey and Ginsberg's assumption holds in that region. Conversely, late beats early when number partitioning problems are easy and satisfiable, refuting Harvey and Ginsberg's assumption in that region. However, in both regions, the improvements in performance are either relatively small or absolutely small. Since a static variable ordering is imposed, both versions must take the same number of discrepancies to find a solution, and thus the gain can only be found in the last probe. An analysis of the data revealed that in soluble instances, the majority of search effort occurs in the last probe. This was somewhat surprising.

In our job shop scheduling experiments, we used a dynamic variable ordering heuristic, and we observed that this can influence the number of probes required to find a solution. Over the Lawrence dataset, there was only one instance where there was a significant difference between taking discrepancies early or late and that was when search resulted in ILDS-early taking one more discrepancy than ILDS-late. However, over the instances examined, there was no clear trend and no clear winner. Sadeh's benchmarks succumbed to ILDS when using the variable and value slack-based heuristics combined, with no significant difference between taking discrepancies late or early. By turning heuristics off and on, we observed that there was no significant difference between ILDS-late and ILDS-early when we used both heuristics combined or none at all. It was only when using partial and unreliable heuristic information that there was a significant difference, and in that case, we should go back to Harvey and Ginsberg's original hypothesis and take discrepancies early.

The random independent set decision problems lend weight to our observations on Sadeh's benchmarks that our heuristic was unreliable at the top of search and that ILDS should take discrepancies early.

The observation was made in Karoui et al. [2007] that if a probe fails to take all of its quota of discrepancies, all subsequent probes will do likewise; consequently, search can be stopped early on unsatisfiable instances. A small modification was made to ILDS to reflect this giving us YLDS, a stripped-down version of YIELDS. This

algorithm performs like ILDS-early on satisfiable instances, but terminates sooner on unsatisfiable instances. Our experiments (in number partitioning, job shop scheduling optimization, and random independent set decision problems) have demonstrated that this modification leads to significant reductions in probes to prove unsatisfiability and this significantly reduces search effort.

Are there any simple “take home messages”? First, if you are using limited discrepancy search, take your discrepancies early (it never seems to hurt). Second, incorporate Korf’s improvement into your limited discrepancy search (it is surprising that some people do not). Third, make sure that you use the YIELDS stopping condition (it is simple, elegant, cheap, and effective). And finally, do some experiments. If those experiments show that there is a significant difference between taking discrepancies late and early, maybe your heuristic is really dumb. And even worse, if there is no difference between late and early, is your heuristic nothing more than a lottery?

ACKNOWLEDGMENTS

We would like to thank Richard Korf, Wafa Karoui, Toby Walsh, Chris Beck, Derek Long, Ian Gent, Alice Miller, Neil Moore, Matt Stallmann, and our reviewers.

REFERENCES

- BECK, J. C. AND PERRON, L. 2000. Discrepancy-bounded depth first search. In *Proceedings of the Constraint Programming, Artificial Intelligence and Operations Research*.
- CHEESEMAN, P., KANEFSKY, B., AND TAYLOR, W. M. 1991. Where the really hard problems are. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 331–337.
- CHOCO. CHOCO Solver. <http://www.emn.fr/x-info/choco-solver/>.
- FURCY, D. AND KOENIG, S. 2005. Limited discrepancy beam search. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.
- GENT, I. P., MACINTYRE, E., PROSSER, P., SMITH, B. M., AND WALSH, T. 2001. Random constraint satisfaction: Flaws and structure. *J. Constraints* 6, 345–372.
- GENT, I. P., MACINTYRE, E., PROSSER, P., AND WALSH, T. 1996. The constrainedness of search. In *Proceedings of the 14th National Conference on Artificial Intelligence*. AIII, Menlo, CA, 246–252.
- GENT, I. P. AND WALSH, T. 1998. Analysis of heuristics for number partitioning. *Comput. Intell.* 14, 3, 430–451.
- HARVEY, W. D. AND GINSBERG, M. L. 1995. Limited discrepancy search. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.
- KARMARKAR, N. AND KARP, R. M. 1982. The differencing method of set partitioning. Tech. rep. UCB/CSD/82/113, Computer Science Division, University of California, Berkeley.
- KAROUI, W., HUGUET, M.-J., LOPEZ, P., AND NAANAA, W. 2007. YIELDS: A yet improved limited discrepancy search for CSPs. In *Proceedings of the Constraint Programming, Artificial Intelligence and Operations Research*.
- KORF, R. 1996. Improved limited discrepancy search. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI’96)*. AIII, Menlo, CA.
- LAWRENCE, S. 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.
- MACKWORTH, A. K. 1977. Consistency in networks of relations. *Artif. Intell.* 8, 99–118.
- MESEGUER, P. AND WALSH, T. 1998. Interleaved discrepancy based search. In *Proceedings of the 13th European Conference on Artificial Intelligence*. John Wiley and Sons, Hoboken, NJ.
- PROSSER, P. 1996. An empirical study of the phase transition in binary constraint satisfaction problems. *Artif. Intell.* 81, 81–109.
- ROSSI, F., VAN BEEK, P., AND WALSH, T. 2006. *Handbook of Constraint Programming*. Elsevier, Amsterdam, The Netherlands.
- ROUSSEL, O. AND LECOUTRE, C. 2008. XML representation of constraint networks, format XCSP 2.1. Universite Lille-Nord de France, Artois, CIRL-CNRS UMIR 8188.
- SADEH, N. 1991. Look-ahead techniques for micro-opportunistic job shop scheduling. PhD thesis CMUCS-91-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

- SMITH, S. F. AND CHENG, C.-C. 1993. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence*. AAAI, Menlo, CA, 139–144.
- VILIM, P. 2009. Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*. Springer, Berlin, 802–816.
- WALSH, T. 1997. Depth-bounded Discrepancy Search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

Received June 2010; revised March 2011; accepted May 2011