A Random Jump Strategy for Combinatorial Search

Hantao Zhang*
Computer Science Department
The University of Iowa
Iowa City, IA 52242
hzhang@cs.uiowa.edu

December 3, 2001

Abstract

Recent work on the combinatorial search has provided experimental and theoretical evidence that randomization and restart strategies proposed by Gomes, Selman, and Kautz, can be very effective for backtrack search algorithms to solve some hard satisfiable instances of SAT. One difficulty of effectively using the restart strategy is its potential conflict with the branching heuristic. It is well-known that a completely random branching heuristic yields very poor performance. To support the restart strategy, the branching heuristic has to be random (with limitation); otherwise, every restart is a repetition of the first run. In this paper, we propose a new randomization strategy which offers the same advantage of the restart strategy but it can be used with any branching heuristics. The basic idea is to randomly jump in the search space to skip some space. Each jump corresponds to a restart in the restart strategy but there is no repetition. We ensure that no portion of the search space is visited twice during one run and the search will be going on until the allotted time is run out or the search space is exhausted. This new strategy is implemented in SATO, an efficient implementation of the Davis-Putnam-Loveland method for SAT problems. Using the new strategy, we are able to solve several previously open quasigroup problems, which could not be solved using any existing SAT systems.

1 Introduction

In the last decade, significant progress has been made in solving combinatorial problems, including both the constraint satisfaction problems (CSP) and the propositional satisfiability problems (SAT). It has been recognized that unpredictability in the running time of complete search procedures can often be explained by the phenomenon of "heavy-tailed cost distributions" [8, 9, 10, 11]. It was proposed in [9] that random restarts can effectively eliminate this phenomenon. That is, randomization is introduced into a complete backtrack search algorithm at two places:

- Introducing certain randomness in the branching heuristic. The amount of randomness may affect the selection of variables and their values for branching and backtrack in the search space.

*partially supported by the National Science Foundation under Grant CCR-0098093.

- Defining a *cutoff* value in the number of backtracks, and repeatedly starting a randomized (using the first idea) complete search procedure at the root of the search tree, each time limiting the maximum number of backtracks to the imposed cutoff value.

These two techniques have been shown very useful for solving some hard combinatorial search problems, including round-robin tournament scheduling [10]. These two techniques can also be combined with other advanced techniques such as lemma learning to solve hard real-world satisfiability problems [1].

To support the restart strategy, the branching heuristic has to be random; otherwise, the restart is a repetition of the first run. It is well-known that a completely random branching heuristic yields very poor performance [14]. That is why Gomes, Selman, and Kautz suggested to use a limited randomization in the branching heuristic.

A potential problem with the restart strategy is that some restarts may repeat the same search space. This happens when the branching heuristic is particular powerful; it may rarely assign more than one choice the highest score. It is suggested in [9] that we have to weaken the branching heuristic to expand the choice set for random tie-breaking. This technique still does not eliminate totally the repetition, especially when restarts are very frequent. For instance, if a restart first picks x, then y and another restart first picks y, and then x, both restarts are exploring the same search space after the first two choices.

In this paper, we suggest a new random jump strategy with the following goals in mind.

- Like the restart strategy, the new strategy should allow the search to jump out of a "trap" when it appears to explore a part of the space far from a solution.
- The new strategy will never cause any repetition of the search performed by the original search algorithm.
- The new strategy will not demand any change to the branching heuristic, so that any powerful heuristic can be used without modification.
- If a search method can exhaust the space without finding a solution, thus showing unsatisfiability, the same search method with the new strategy should be able to do the same.

The main idea of our new strategy can be described as follows: Suppose a search program is allotted time t to run on a problem. After the program runs for x time, where $0 < x < t$, we compare the percentage of the remaining time, i.e., $(t - x)/t$, against the percentage of the remaining search space. If the percentage of the remaining space is larger than that of the remaining time, we say the remaining space is *sufficiently large*. If this is the case, we may skip some unexplored space to avoid some traps, as long as the remaining space is still sufficiently large. Of course, it is impossible to compute the percentage of the remaining space exactly. In our case, an estimate will be okay as long as we do not over-estimate it, because if we over-estimate the space, we may skip too much space and run out of the space before the allotted time is used up.

The space explored by a backtrack search procedure can be represented by a tree, where an internal node represents a backtrack point, and a leaf node represents either a solution or no solution. At a checkpoint, we look at the path from the current node to the root of the tree to see how many branches have been closed and how many branches are still open, to estimate the

percentage of the remaining space. If the remaining space is sufficiently large, we may jump up along the path, skipping some open branches along the way. After the jump, the remaining space should still be sufficiently large.

The checkpoints in our strategy can be decided as the cutoff points are decided in the restart strategy. The i^{th} checkpoint can be decided by the number $c(1+\alpha_i)$ of backtracks done after the last checkpoint, where $0 < \alpha_i < 1$ is a random number and c is a pre-selected constant as the cutoff value in the restart strategy.

The new randomization strategy is implemented in SATO (SAtisfiability Test Optimized) [24, 26], which is an efficient implementation of the Davis-Putnam-Loveland (DPL) method [5, 6]. The new strategy allowed us to solve a dozen of previously open problems in the study of quasigroups.

To simplify the presentation of this paper, we will limit the discussion of our idea on the DPL method, while the idea applies also to the CSP procedures. The search space explored by the DPL method can be represented by a binary tree. We assume that the left branch of each decision point is the heuristic choice.

2 Random Jump in the Search Space

The Davis-Putnam-Loveland (DPL) method has long been a major practical method for solving SAT problems. It is based on unit propagation (i.e., unit resolution and unit subsumption) and case-splitting. The search space related to the DPL method can be viewed as a binary tree. The case-splitting rule creates an internal node with two children in the search space. Without loss of any generality, if L is the literal picked at an internal node for splitting, we assume that the link pointing to the left child is marked with $\langle L, 1\rangle$ (called *left link*) while the link pointing to the right child is labeled with $\langle L, 0\rangle$ (called *right link*). A leaf node in the search space is marked with either an empty clause (a conflict) or an empty set of clauses (a model).

We can record the path from the root of the tree to a given node, called *guiding path* in [24], by listing the labels on the path. For the implementation of the DPL method, this guiding path is stored in a stack. We can use guiding paths to avoid repeated search so that the search effort can be accumulated. For instance, the guiding path $(\langle x_1, 1\rangle\ \langle \neg x_5, 0\rangle\ \langle x_3, 0\rangle)$ tells us that the literals are selected in the order of $x_1, \neg x_5, x_3$, and the subtree starting with the path $(\langle x_1, 1\rangle\ \langle \neg x_5, 1\rangle)$ is already complete; so is the subtree starting with $(\langle x_1, 1\rangle\ \langle \neg x_5, 0\rangle\ \langle x_3, 1\rangle)$.

For the standard DPL method, the backtrack is done by searching bottom-up for the first left link in the guiding path. If a left link is found, we replace it by its corresponding right link and continue the search from there. To implement the jump, we may skip a random number of left links in the guiding path as long as the remaining search space is sufficiently large.

2.1 Estimating Remaining Search Space

At any time during the execution of the DPL method, we can use the current guiding path to estimate the percentage of the remaining search space. That is, assuming the search space divides equally between left and right branches, given a guiding path

$$(\langle l_1, \delta_1\rangle \langle l_2, \delta_2\rangle \cdots \langle l_n, \delta_n\rangle),$$

we can compute the percentage of the remaining search space as $\sum_{i=1}^{n}(0.5\delta_i)^i \cdot 100\%$. For example, the percentage of the remaining space for $(\langle x_1, 1\rangle\ \langle \neg x_5, 0\rangle\ \langle x_3, 0\rangle)$ is 50%.

Since the search space does not always divide equally between left and right branches, in order to avoid the over-estimate, we may use $\sum_{i=1}^{n}(\beta\delta_i)^i \cdot 100\%$ for $\beta = 0.45, 0.4, 0.35, ...$, in practice. We can also adopt some learning scheme to dynamically adjust the value of β. Comparing this percentage with the percentage of the remaining time, we can decide if the search space is sufficiently large.

2.2 Proving Unsatisfiability

Backtrack search algorithms are said to be complete because they can be used to prove unsatisfiability, when the search space is exhausted. It is pointed in [9] that the restart strategy with a fixed cutoff value is not complete because it limits the search space. One simple solution as provided in [1] is to implement a policy for increasing the cutoff value. That is, after each restart the backtrack cutoff value can be increased by a constant amount. The resulting algorithm becomes complete by gradually letting the restart strategy faded out. This solution is obviously not effective to prove unsatisfiability because all the runs except the last are wasted. Recently, Baptista, Lynce and Marques-Silva [2] proposed a new strategy which records some lemmas learned, call *search signature*, between restarts, so that some repeated search can be avoided.

We argue that a complete search program using our random jump strategy does not compromise its completeness, even though our strategy may skip some search space. Suppose a program can exhaust the search without finding a model in time t. Given an allotted time a, this algorithm can prove unsatisfiability if $a \geq t$; otherwise, the search is incomplete. If $a \geq t$, using our strategy, at each checkpoint, the check will see that the remaining search space is not sufficiently large, so no search space will be skipped and when the search is complete, we know the search space is exhausted, thus proving unsatisfiability. If $a < t$, some search space will be skipped by our strategy but we know the search cannot be complete in the given time anyway.

3 Related Work

Our strategy is simply a modification of the depth-first search (DFS) procedure with backtracking. In terms of the traversal of the decision tree, our modified DFS acts very much like the standard DFS with intelligent backjumping (such as conflict-directed backjumping [20] or dependency-directed backtracking [3]), because the right branches of some decision points are skipped in both cases. The only difference is that the DFS with intelligent backjumping knows through analysis that the skipped right branches do not contain solutions, while the modified DFS just does not want to spend time on these branches. Of course, the new strategy can be adopted in the DFS with intelligent backjumping without any conflict.

Each restart in the restart strategy creates very likely a new search tree, because randomness is introduced in the variable selection heuristic. Each restart explores the space like a standard DFS, until the cutoff value is reached. In our strategy, randomness is introduced at two places: the selection of checkpoints, and the number of right branches to be skipped if the search space is found to be sufficiently large at a checkpoint. The goals of the restart strategy and our strategy are the same, i.e., to allow the search to jump out of a "trap" when it appears to spend too much effort at one place. Since there is no conflict between the randomness in both strategies, the two

strategies can be combined as follows: Each restart runs the modified DFS with our strategy; the cutoff value for the restart will be the allotted run time.

In the restart strategy, if the randomness in the branching heuristic is limited such that we always choose the variable selected by the original heuristic, but its values can be randomly selected, then each restart will explore the same search tree, with different (random) traversal orders. Furthermore, if we set the cutoff value to one, that is, each restart explores only one leaf node, then this special case of the restart strategy is a variant of the limited discrepancy search proposed by Harvey and Ginsberg [12, 13], and modified by Korf [15] and Walsh [21].

The motivation behind the limited discrepancy search approach is the same as that of the restart strategy and our strategy: Branching heuristics sometimes guide the search toward regions of the space that are unlikely to contain solutions, and we need a mechanism to get out of these regions before we exhaust our time. A discrepancy is any decision point in a search tree where we go against the heuristic. The limited discrepancy search explores the space systematically by allowing a limited number of discrepancies, i.e., not following the heuristic a limited number of points. The number of discrepancies goes from 0 to a given maximum limit [12] or from 0 to the maximum depth of the search tree [21]. All the proposed strategies in the limited discrepancy search involve different ways of systematically visiting all the space. It appears that Walsh's strategy works best because it has less redundancy than the others and uses discrepancies early in the top of the search tree, because it is believed that branching heuristics are more likely to be wrong at the top of the tree than at the bottom.

All the proposed strategies in the limited discrepancy search are deterministic in nature. The special case of the restart strategy discussed above (no randomness in variable selection and cutoff value set to one), can be viewed as a random discrepancy search. It has been claimed in both [12] and [21] that the limited discrepancy search can be expected to outperform existing approaches. However, we still need to see how this approach performs on hard problems of practical interests, for instance, the quasigroup problems introduced in the next section. One may doubt about the effectiveness of this approach because the experimental results [9] showed that the restart strategy works poorly when the cutoff value is very small.

When the limited discrepancy search stops prematurely because of time limitation, the unexplored regions are scatted over the whole space. On the contrary, the unexplored regions in the DFS with our strategy are always the right branches of some decision points. We may combine our strategy with the limited discrepancy search approach in at least two ways.

- Using the random discrepancy at decision points (with a probability in favor of the heuristic choice), so that some left branches (assuming the heuristic choice always leads to the left branch) will be skipped.
- Using the limited discrepancy search only for the top portion of the search tree. For instance, we may use Walsh's strategy for the nodes of depth 0 to 4 in the search tree, and then run our modified DFS at depth deeper than four. This way, the redundancy problem associated with the limited discrepancy search can be largely avoided.

A feature of the new randomization strategy distinct from the other strategies discussed above is that it uses the allotted time as a parameter of the strategy. This idea was inspired from the work of Riazanov and Voronkov who considered the time as a parameter for designing efficient strategies for first-order theorem proving [19].

Table 1: Three strategies used for solving 1,000 instances of satisfiable 3-SAT problems at $L/N = 3.5$. A 10 minutes time slot is given for each problem. "DFS" is the standard Davis-Putnam-Loveland (DPL) procedure. "Restart" refers to the restart of the DPL procedure 10 times or until a solution is found; each restart is allowed to run for at most one minute. "Jump" is the DPL procedure with the new random jump strategy. The columns under "failed" give the numbers of failed cases (out of 1,000). The random jump strategy succeed in every run and the "failed" column is omitted from the table. The average time used and the leaf nodes explored are given under "time" (in seconds) and "nodes", respectively. The times were collected on a Linux PC (300MHz; 128M memory) running SATO. The column "R" provides the average number of restarts for each problem. The column "skipped" provides the average number of branches skipped by our strategy.

	DFS			Restart				Jump		
N	failed	time	nodes	failed	time	nodes	R	time	nodes	skipped
250	0	1.10	520	0	1.06	495	1.0	1.02	499	1
300	4	11.85	3906	0	10.27	3822	1.0	4.57	1295	37
350	12	41.63	12625	0	21.84	4307	1.2	11.94	3599	166
400	81	111.97	30867	18	53.43	6909	1.6	23.22	6898	411
450	282	235.11	60021	42	110.92	9458	2.6	35.03	9030	2021
500	476	339.01	71561	163	239.37	18504	4.5	50.94	10727	3959

4 Experiments

We have tested the new randomization strategy in SATO with various problems, including many problems in the DIMACS benchmarks. To avoid frequent checking, we set the default checkpoint value to $c = 2^{10}$. SATO can find a solution for many DIMACS benchmark problems without skipping any branches, because either it found a solution before the first checkpoint or the remaining search space was not sufficiently large. However, only using the new strategy, SATO could solve hanoi5 for the first time after a run of 20 hours. In the following, we concentrate on two sets of the SAT problems: random 3-SAT and quasigroup problems.

4.1 Random 3-SAT

As in [12] and [21], we use the random 3-SAT problems to test the strategies discussed in the paper. It is reported in [18] that a phrase transition occurs for this model at a clause to variable ratio, L/N, of approximately 4.3. In [21], the ratio $L/N = 3.5$ is used because most problems at this ratio are satisfiable. Walsh tested the problems for variables N from 50 to 250. These problems cannot be used to show the difference of our strategies because they are too easy to be solved. Instead, we generated 1,000 instances each for N from 250 to 500. As in [21], each problem is solved using the Davis-Putnam-Loveland procedure, branching on a literal in the shortest clauses.

It is clear from the table that our strategy works best for this set of 3-SAT problems because all of the problems were solved by the new random jump strategy and it took the least average time to solve them. The standard DFS started to fail on some problems of $N = 300$. The restart strategy started to fail when $N = 400$. For $N = 250$, almost no branches were skipped by our strategy because the remaining search space was not sufficiently large for such problems. All the three procedures performed the same on this set of the problems. In fact, for easy problems like

$N = 250$, there is no need to use any fancy strategies. The merit of our strategy can be only seen when attacking hard problems.

4.2 Solving Open Quasigroup Problems

Our real interests for developing the new strategy are on solving open quasigroup problems [22]. The simple cases of the quasigroups are much better benchmarks than randomly generated SAT problems for testing constraint-solving methods. Quasigroup problems are real problems, have fixed solutions and simple descriptions that are easy to communicate. More importantly, open problems in quasigroups have attracted the interest of many researchers and became a challenge for friendly competition. The usefulness of general automated reasoning techniques to attack open quasigroup problems was successfully demonstrated by various reasoning programs [7].

A quasigroup is a cancellative finite groupoid $\langle S, * \rangle$, where S is a set and $*$ a binary operation on S. The cardinality of S, $|S|$, is called the *order* of the quasigroup. The "multiplication table" for the operation $*$ forms a Latin square indexed by S, where each row and each column is a permutation of S. Many classes of quasigroups are interesting, partly because they are very natural objects in their own right and partly because of their relationships to design theory [22].

Without loss of generality, let S be $\{0, 1, ..., (v-1)\}$, where v is the order of $\langle S, * \rangle$. The following clauses specify a quasigroup, or equivalently, a Latin square: for all elements $x, y, u, w \in S$,

$$x * u = y, x * w = y \Rightarrow u = w \quad : \text{the left-cancellation law} \tag{1}$$

$$u * x = y, w * x = y \Rightarrow u = w \quad : \text{the right-cancellation law} \tag{2}$$

$$x * y = u, x * y = w \Rightarrow u = w \quad : \text{the unique-image property} \tag{3}$$

$$(x * y = 0) \vee \cdots \vee (x * y = (v-1)) \quad : \text{the closure property} \tag{4}$$

A quasigroup may satisfy additional constraints, such as QG1 and QG2 below:

Name	**Constraint**
QG1	$x * y = u, z * w = u, v * y = x, v * w = z \Rightarrow x = z, y = w$
QG2	$x * y = u, z * w = u, y * v = x, w * v = z \Rightarrow x = z, y = w$

In the following, QG$i(v)$ denotes a quasigroup of order v that satisfies clauses (1)–(4) plus QGi for $S = \{0, ..., (v-1)\}$. In addition, the *idempotency* law, $x * x = x$, is assumed in every problem unless otherwise stated. The reader may find additional details on these problems in [22].

Propositional clauses are obtained simply by instantiating the variables in clauses (1)–(4) by values in S and replacing each equality $x * y = z$ by a propositional variable $p_{x,y,z}$.

Among the Latin squares satisfying these constraints, we are often interested in those squares with holes, i.e., some subsquares of the square are missing. A *holey quasigroup* of type $a^n b^1$ is a quasigroup with n disjoint holes of size a and a single hole of size b. We denote a holey quasigroup of type $a^n b^1$ satisfying QGi, $i = 1, 2$, by QG$i(a^n b^1)$. If $a = b$, we simply write it as QG$i(a^{n+1})$. Note that every idempotent QG$i(v)$ can be considered as a QG$i(1^v)$. The reader may find additional details on how to specify holey quasigroups in [22].

The following results are reported in [4, 25]:

Theorem 1 *There exists a QG1(h^n) if and only if $h \geq 1$ and $n \geq 4$, except $(n, h) = (6, 1)$ and except possibly $(n, h) = (6, 13)$.*

Theorem 2 *There exists a QG1($2^n 3^1$) if and only if $n \geq 5$, except possibly $n = 17, 18, 19, 21, 22$ and 23.*

Theorem 3 *For $2 \leq k \leq 5$, a QG2($1^v k^1$) exists if and only if $v \geq 3k + 1$, except possibly when $k = 4$ and $v \in \{35, 38\}$.*

Without the random jump strategy, given a week of run time, SATO could not solve any of the possible exceptions in the above theorems. With the strategy, SATO solved each of them in an overnight run. This clearly demonstrated the power of the new strategy. The above theorems are thus updated as the following new theorem:

Theorem 4 *(1) There exists a QG1(h^n) if and only if $h \geq 1$ and $n \geq 4$ except $(n, h) = (6, 1)$.*
(2) There exists a QG1($2^n 3^1$) if and only if $n \geq 5$.
(3) For $2 \leq k \leq 5$, a QG2($1^v k^1$) exists if and only if $v \geq 3k + 1$.

As a challenge to all the combinatorial search procedures, we point out that the existence of the following quasigroups are still unknown: QG1($2^5 1^1$), QG1($2^6 1^1$), QG1($2^7 1^1$), QG2(3^4), QG2(5^4), and QG2(7^4).

5 Conclusion

We have presented a simple randomization strategy which takes the allotted run time as a parameter and randomly set points to check if the remaining search space is sufficiently large comparing to the remaining run time; if yes, some space will be skipped. Like the restart strategy, it can prevent a backtrack search procedure from getting trapped in the long tails of many hard combinatorial problems and help it to find a solution quicker. Unlike the restart strategy, it never revisits any search space decided by the original search procedure. Unlike the restart strategy, it does not lose the ability of the original procedure to prove unsatisfiability; in this case, neither does it provide any help (the overhead of the strategy is very small and is ignorable). The four design goals listed in the introduction are clearly met. We demonstrated the power of the new strategy by solving several previously open quasigroup problems.

References

[1] Baptista, L., and Margues-Silva, J.P., Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability, in Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP), September 2000.

[2] Baptista, L., Lynce I., and Marques-Silva, J.P., Complete Search Restart Strategies for Satisfiability, in the IJCAI'01 Workshop on Stochastic Search Algorithms (IJCAI-SSA), August 2001.

[3] Bayardo, R., Schrag, R., Using CSP look-back techniques to solve exceptionally hard SAT instances. In Proceedings of CP-96, 1996.

[4] Bennett, F.E, Zhang, H.: (1998) Existence of $(3, 1, 2)$–HCOLS and $(3, 1, 2)$–HCOLS, *J. of Combinatoric Mathematics and Combinatoric Computing*, **27** (1998) 53-64

[5] Davis, M., Putnam, H. (1960) A computing procedure for quantification theory. *Journal of the ACM*, **7**, 201–215.

[6] Davis, M., Logemann, G., and Loveland, D.: A machine program for theorem-proving. *Communications of the Association for Computing Machinery 5,* 7 (July 1962), 394–397.

[7] Fujita, M., Slaney, J., and Bennett, F.: Automatic generation of some results in finite algebra. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

[8] Gomes, C.P., Selman, B., and Crato, C.: Heavy-tailed Distributions in Combinatorial Search. In Principles and Practices of Constraint Programming, (CP-97) Lecture Notes in Computer Science 1330, pp 121-135, Linz, Austria., 1997. Springer-Verlag.

[9] Gomes, C.P., Selman, B., and Kautz, H.: Boosting combinatorial search through randomization. In Proceedings of AAAI98, Madison, WI, July 1998.

[10] Gomes, C.P., Selman, B., McAloon, K., and Tretkoff, C.: Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In Proceedings, AIPS98, Pittsburg, PA, June 1998.

[11] Gomes, C.P., Selman, B., Crato, N., and Kautz, H.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. J. of Automated Reasoning, Vol. 24(1/2), pages 67-100, 2000

[12] Harvey, W. D. and Ginsberg, M. L. Limited discrepancy search. In Proceedings of 14th IJCAI, 1995.

[13] Harvey, W. D. Nonsystemic backtracking search. PhD thesis, Stanford University, 1995.

[14] Hooker, J.N., Vinay, V. (1995) Branching rules for satisfiability. *Journal of Automated Reasoning*, **15**, No.3 359-383.

[15] Korf, R. Improved limited discrepancy search. In Proceedings of 13th AAAI, 1996.

[16] Lynce, I., Baptista, L., and Marques-Silva, J. P., Stochastic Systematic Search Algorithms for Satisfiability, in the LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT), June 2001.

[17] Marques-Silva, J. P., and Sakallah, K. A., GRASP: A Search Algorithm for Propositional Satisfiability, in IEEE Transactions on Computers, vol. 48, no. 5, pp. 506-521, May 1999.

[18] Mitchell, D., Selman, B., and Levesque, H., Hard and easy distributions of SAT problems. In Proceedings of 10th AAAI, 459-465, 1992.

[19] Riazanov, A., and Voronkov, A., Limited resource strategy in resolution theorem proving. To appear in J. of Symbolic Computation.

[20] Smith, B. M., Grant, S., Sparse constraint graphs and exceptionally hard problems. Proceedings of 14th IJCAI, 1995.

[21] Walsh, T., Depth-bounded discrepancy search. Proceedings of 15th IJCAI, (1997).

[22] Zhang, H.: (1997) Specifying Latin squares in propositional logic, in R. Veroff (ed.): Automated Reasoning and Its Applications, Essays in honor of Larry Wos, Chapter 6, MIT Press.

[23] Zhang, H.: (1997) SATO: An efficient propositional prover, Proc. of International Conference on Automated Deduction (CADE-97). pp. 308–312, Lecture Notes in Artificial Intelligence 1104, Springer-Verlag.

[24] Zhang, H., Bonacina, M.P., Hsiang, H.: PSATO: a distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation* (1996) 21, 543–560.

[25] Zhang, H., Bennett, F.E.: Existence of some $(3, 2, 1)$–HCOLS and $(3, 2, 1)$–HCOLS. *J. of Combinatoric Mathematics and Combinatoric Computing*, 22 (1996) 13-22.

[26] Zhang, H., Stickel, M.: Implementing the Davis-Putnam method, *J. of Automated Reasoning* 24: 277-296, 2000.