A search based Sudoku solver

Tristan Cazenave

Labo IA
Dept. Informatique
Université Paris 8, 93526, Saint-Denis, France,
cazenave@ai.univ-paris8.fr

Abstract. Sudoku is a popular puzzle. In this paper we detail constraint satisfaction search algorithms used to solve Sudoku problems. We show that a good value ordering heuristic helps solving problems. We also show that Limited Discrepancy search is a good alternative to the traditional Forward Checking algorithm. We have also found a phase transition for 25x25 Sudoku.

1 Introduction

Sudoku is a popular puzzle, that was first published in 1979, and that became popular in Japan before its international development. The goal of Sudoku is to put numbers from one to nine on a 9x9 grid. The grid is pre-assigned with numbers. The final position must have each number only once in each row and each column. Moreover nine 3x3 squares are also part of the grid, and each number must appear once and only once in each square.

In this paper we test two search algorithms for Sudoku: Forward Checking (FC), and Limited Discrepancy Search (LDS) [1]. We also assess the difficulty of Sudoku problems relative to the number of holes they contain.

The contributions of this paper are :

– There is a clear phase transition for Sudoku problems of size 25x25.
– The min-domain-sum value ordering heuristic works better than the lexicographic ordering of values.
– Limited Discrepancy Search is a good alternative to the Forward Checking algorithm for Sudoku.

The second section presents the Sudoku problem, the third section details related search algorithms, the fourth section presents experimental results, the last section concludes.

2 Sudoku

This section starts with describing the quasi-group completion problem, the second subsection gives the additional constraints used to model Sudoku, the third subsection details how problems have been generated, the fourth subsection explains how redundant modeling can help to solve the problem.

2.1 The quasi group completion problem

A quasi-group is an $n \times n$ multiplication table which defines a Latin square. It is a matrix such that each row and each column contains n different elements. There are only n possible values for the cells of the matrix. The size of an $n \times n$ quasi-group is n.

The quasi-group completion problem (QCP) consists in completing a quasi-group with holes (unassigned variables).

QCP is an ideal testbed for constraint satisfaction algorithms as it has structure and it is easy to generate arbitrarily hard problem instances [2]. The problem has a phase transition phenomenon which follows an easy-hard-easy pattern, depending on the number of unassigned variables. This phase transition follows the backbones of the problems: i.e. the variables that are fully constrained and take the same value in all solutions.

QCP has also been used to discover important properties of search algorithms such as heavy-tailed behavior [3] and the associated rapid randomized restarts (RRR) strategies [4, 5].

Moreover, it is related to real world applications such as statistical design, error correcting codes, conflict free wavelength routing and timetabling [6, 7].

2.2 The Sudoku problem

Sudoku can be modeled as QCP with additional constraints. The usual size for Sudoku problems is 9x9. In this paper, we have also studied 16x16 and 25x25 Sudoku. In the usual version the board is partitioned in nine 3x3 squares. In the 16x16 version, there are sixteen 4x4 squares, and in the 25x25 version, there are twenty five 5x5 squares. The additional constraints used for the all the versions are that each square must contain all the possible values.

In previous works, Sudoku as already been modeled as a constraint satisfaction problem [8] and as a SAT problem [9]. The total number of possible Sudoku grids as also been computed [10].

2.3 Problem generation

In order to generate a problem, a limited discrepancy search is used to generate a complete Sudoku of a given size. Then variables are randomly selected among the assigned variables (initially all the variables) and are unassigned until the number of holes (unassigned variables) corresponds to the desired percentage of the total number of variables. For each size, and for each percentage, 50 problems have been generated.

Usually, Sudoku problems only have one solution. In the problem we have generated, they can have multiple solutions.

2.4 Redundant modeling

Redundant modeling has been used with success for QCP [6]. The primal and natural model for a QCP of order n is to take as the n^2 variables the cells of the matrix, the common initial domain for the variables being $D = \{k|1 \leq k \leq n\}$. The variables can be named x_{ij} where i is the row and j the column of the variable. The n^2 constraints for the rows are $x_{ij} \neq x_{il}$ with $j \neq l$, and the n^2 constraints for the columns are $x_{ij} \neq x_{lj}$ with $i \neq l$. The row dual model consists in considering in which column in a given row is a given color. There are n^2 constraints of the form $r_{ik} \neq r_{il}$ with $l \neq k$, and n^2 constraints of the form $r_{ik} \neq r_{jk}$ with $i \neq j$. The column dual model consists in considering in which row in a given column is a given color. There are n^2 constraints of the form $c_{jk} \neq c_{jl}$ with $k \neq l$, and n^2 constraints of the form $c_{jk} \neq c_{lk}$ with $j \neq l$. The row channeling constraints are: $x_{ij} = k \Leftrightarrow r_{ik} = j$. The column channeling constraints are: $x_{ij} = k \Leftrightarrow c_{jk} = i$. This is exactly the redundant modeling used in [6]. A good value ordering heuristic associated to this redundant modeling is the *min-domain-sum value selection heuristic* (vdom+). It consists in choosing the value whose corresponding two variables have a minimal sum of domains sizes. It is also interesting to note that Forward Checking with redundant models is almost equivalent to Arc Consistency for QCP, except for the order of instantiation of the singleton variables.

An alternative modeling of the problem is to use $2n$ all different n-ary constraints [11].

We have reused the redundant modeling with the min-domain-sum value selection heuristic for the Sudoku problem.

3 Search Algorithms

The search algorithms we have tested are Forward Checking and Limited Discrepancy Search which are presented in this order in this section.

3.1 Forward Checking

The Forward Checking algorithm consists in verifying, after each assignment of a value to a variable, all the constraints in which the variable appears. It helps reducing the domain of the free variables that appear in these constraints.

For example, in Sudoku, each time a value is assigned to a variable, the value is removed from the domain of the free variables that are either in the same line, in the same column or in the same square as the assigned variable.

3.2 Limited discrepancy search

Limited Discrepancy Search (LDS) was proposed by Harvey and Ginsberg [1]. A discrepancy is a node in the search tree where the algorithm does not follow the heuristic. Given a leaf in a search tree, its discrepancy order is the number of discrepancies that have been necessary to reach it. LDS explores leaves in increasing discrepancy orders. Korf improved LDS with Improved Limited Discrepancy Search (ILDS) [12] which avoids revisiting leaf nodes at the depth limit with lower discrepancy orders. T. Walsh proposes. Depth-bounded limited discrepancy search (DDS) [13] which focus on branching decisions at the top of the search tree where solution are more likely to be wrong. A closely related algorithm is Interleaved and Discrepancy Based Search [14] which has been applied with success to the quasi-group completion problem.

4 Experimental results

Experiments use a Pentium 4 3.0 GHz with 1 GB of RAM. In all our experiments we have used the fail first heuristic for variable selection. We select the variable that has the smallest domain size in the primal model.

Table 1 gives the cumulated search time and the number of solved problems for one thousand Sudoku problems of size 9x9 for different algorithms. It means that the one thousand problems are all solved in less than a quarter of second, with an average of 0.00014 second per problem.

There are 50 different problems every 5%, starting at 1% of holes (50 problems with 1% of holes, 50 problems with 6% of holes, and so on until 96%).

The upper part of the table gives the results using the lexicographic value ordering heuristic, while the lower part gives the results for the $vdom+$ value ordering heuristic. Similar orderings between the algorithms arise with the two value ordering heuristics. LDS performs better than Forward Checking in both cases.

Table 1. Cumulated time for 50 Sudoku of size 9x9 every 5% with a timeout of 100s.

Algorithm	Size	Time	Solved
$FC(lex)$	9x9	0.24s	1,000
$LDS(lex)$	9x9	0.19s	1,000
$FC(vdom+)$	9x9	0.22s	1,000
$LDS(vdom+)$	9x9	0.14s	1,000

Table 2 gives similar results for Sudoku of size 16x16 with a timeout of 100 seconds.

It is noticeable that the savings are much more important in table 2 than in table 1. Here LDS is very clearly better than FC.

Table 2. Cumulated time for 50 Sudoku of size 16x16 every 5% with a timeout of 100s.

Algorithm	Size	Time	Solved
$FC(lex)$	16x16	10,021s	909
$LDS(lex)$	16x16	232s	1,000
$LDS(vdom+)$	16x16	131s	1,000

Figure 1 details the time performance of Forward Checking and Limited Discrepancy Search for Sudoku of size 16x16 for each percentage tested. The name finishing with lds.l.t is Limited Discrepancy Search with lexicographic ordering of values, lds.d.t is Limited Discrepancy Search with the $vdom+$ value ordering heuristic, and fc.l.t is Forward Checking with lexicographic ordering of values.

Figure 2 details the number of solved problems for the same algorithms.

We can observe a phase transition for Forward Checking in these two figures, starting at 61% of holes. However Limited Discrepancy search is almost unaffected by this transition and solves all problems before the timeout.

Table 3. Cumulated time for 50 Sudoku of size 25x25 every 5% with a timeout of 100s.

Algorithm	Size	Time	Solved
$FC(lex)$	25x25	37,309s	639
$LDS(lex)$	25x25	46,928s	563
$LDS(vdom+)$	25x25	27,914s	745

Table 3 gives the results for Sudoku of size 25x25 with a timeout of 100 seconds. On the contrary of 16x16 results, Forward Checking is here faster than Limited Discrepancy Search with lexicographic ordering of values. However Limited Discrepancy Search with the $vdom+$ value ordering heuristic is faster than both algorithms.

Figure 3 details the time performance of Forward Checking and Limited Discrepancy Search for Sudoku of size 25x25 for each percentage tested. Figure 4 details the number of solved problems for the same algorithms.

In these figures, we observe that all three algorithms have a very clear phase transition at 51% of holes. At 46% all problems are solved by all algorithms, whereas at 51%, no problem is solved by any of the algorithms.

5 Conclusion

We have shown a phase transition for 25x25 Sudoku at 51 % of holes. We have also shown that the min-domain-sum value ordering heuristic, and that Limited Discrepancy Search are useful for solving Sudoku problems.

Future work include using our search algorithm to generate Sudoku problems with a unique solution, finding the minimum number of pre-assigned variables necessary to have unique solutions, and improving the search algorithm.

References

1. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In Mellish, C.S., ed.: IJCAI-95, Montréal, Québec, Canada, Morgan Kaufmann (1995) 607–615
2. Achlioptas, D., Gomes, C.P., Kautz, H.A., Selman, B.: Generating satisfi able problem instances. In: AAAI/IAAI. (2000) 256–261
3. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfi ability and constraint satisfaction problems. Journal of Automated Reasoning **24** (2000) 67–100
4. Gomes, C.P., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. In: AAAI-98. (1998) 431–437
5. Walsh, T.: Search in a small world. In: IJCAI-99. (1999) 1172–1177

6. Dotu, I., del Val, A., Cebrian, M.: Redundant modeling for the quasigroup completion problem. In Rossi, F., ed.: Principles and Practice of Constraint Programming - CP 2003. Volume 2833 of Lecture Notes in Computer Science., Springer (2003) 288–302
7. Gomes, C.P., Shmoys, D.B.: The promise of lp to boost csp techniques for combinatorial problems. In: CPAIOR'02. (2002)
8. Simonis, H.: Sudoku as a constraint problem. In: CP Workshop on modeling and reformulating Constraint Satisfaction Problems. (2005)
9. Lynce, I., Ouaknine, J.: Sudoku as a SAT problem. In: 9th International Symposium on Artifi cial Intelligence and Mathematics. (2006)
10. Felgenhauer, B., Jarvis, F.: Enumerating possible Sudoku grids. www.shef.ac.uk/ pm1afj/sudoku/sudoku.pdf (2005)
11. Shaw, P., Stergiou, K., Walsh, T.: Arc consistency and quasigroup completion. In: Proceedings of ECAI98 Workshop on Non-binary Constraints, Brighton. (1998)
12. Korf, R.E.: Improved limited discrepancy search. In: AAAI-96. (1996)
13. Walsh, T.: Depth-bounded discrepancy search. In: IJCAI-97. (1997) 1388–1395
14. Meseguer, P., Walsh, T.: Interleaved and discrepancy based search. In: ECAI-98. (1998) 239–243

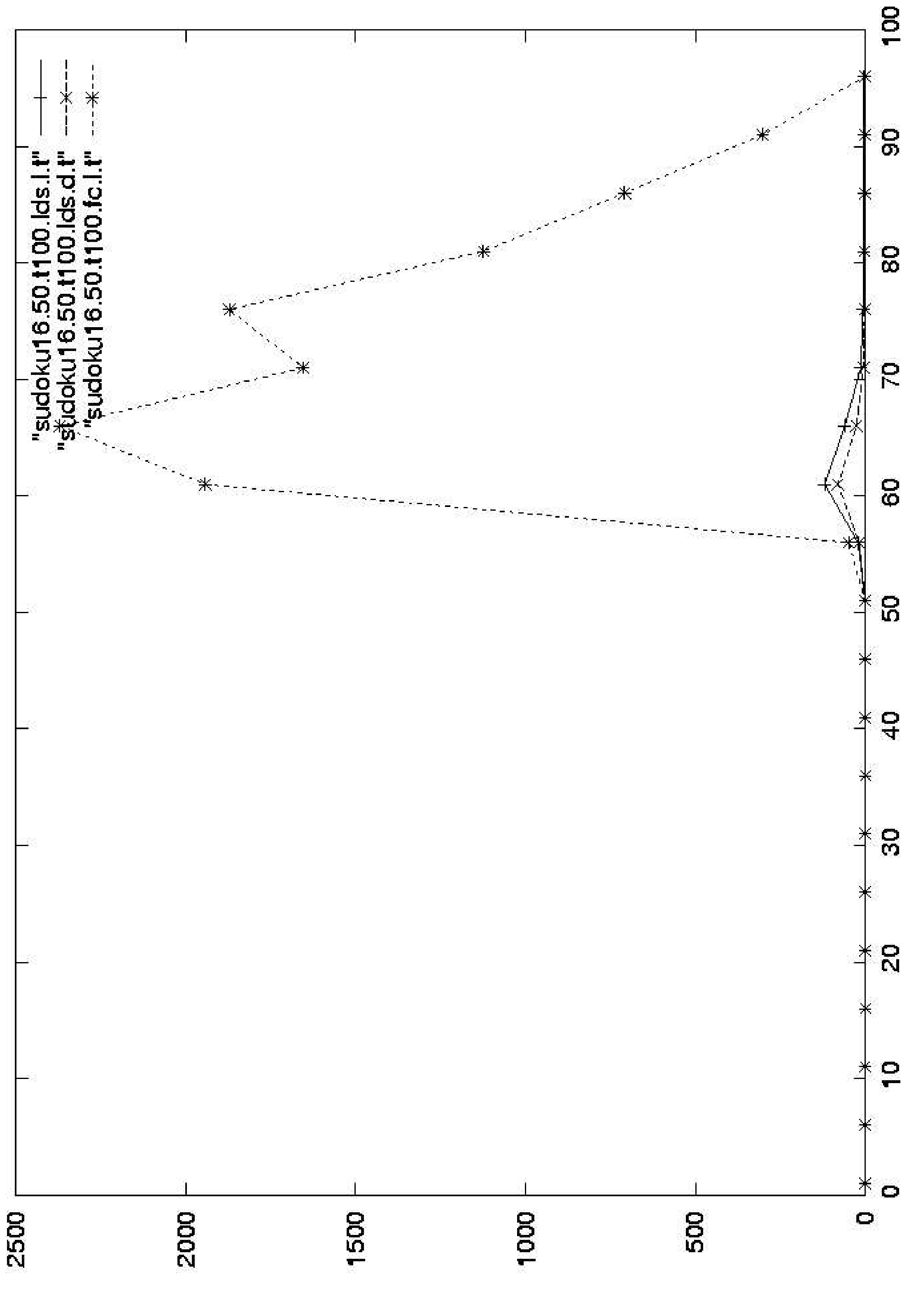

Fig. 1. repartition of time for 16x16 Sudoku

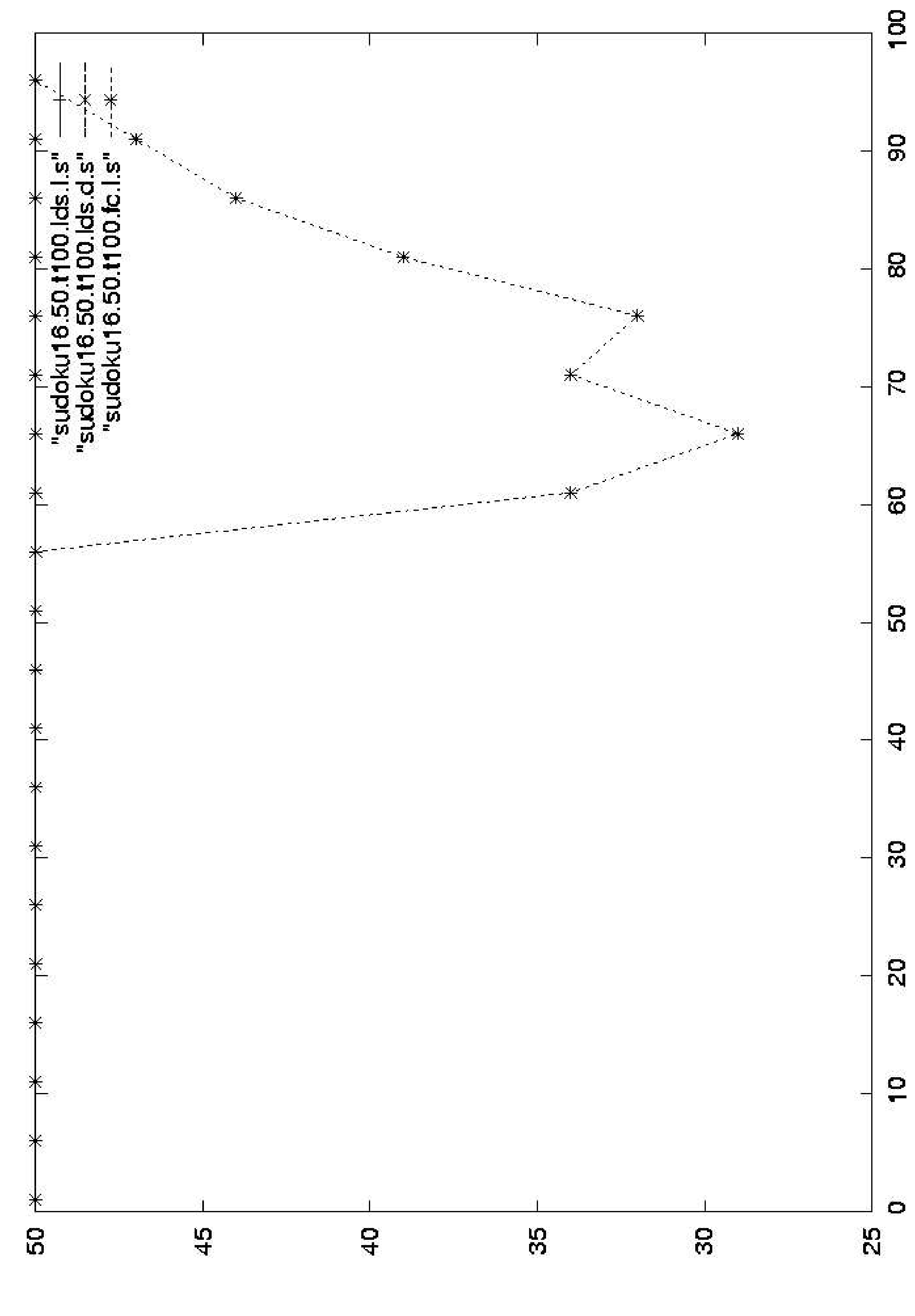

Fig. 2. repartition of solved problems for 16x16 Sudoku

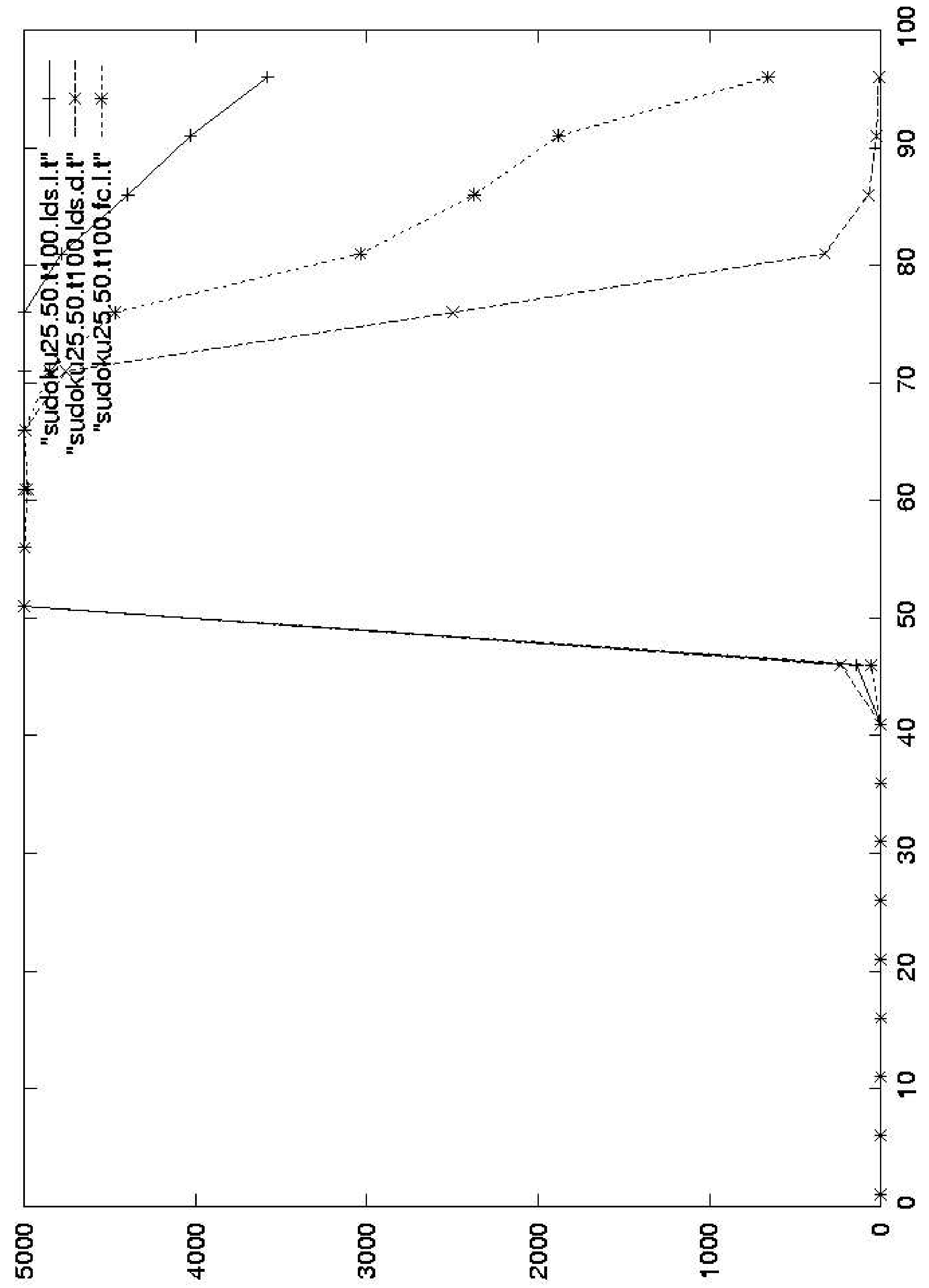

Fig. 3. repartition of time for 25x25 Sudoku

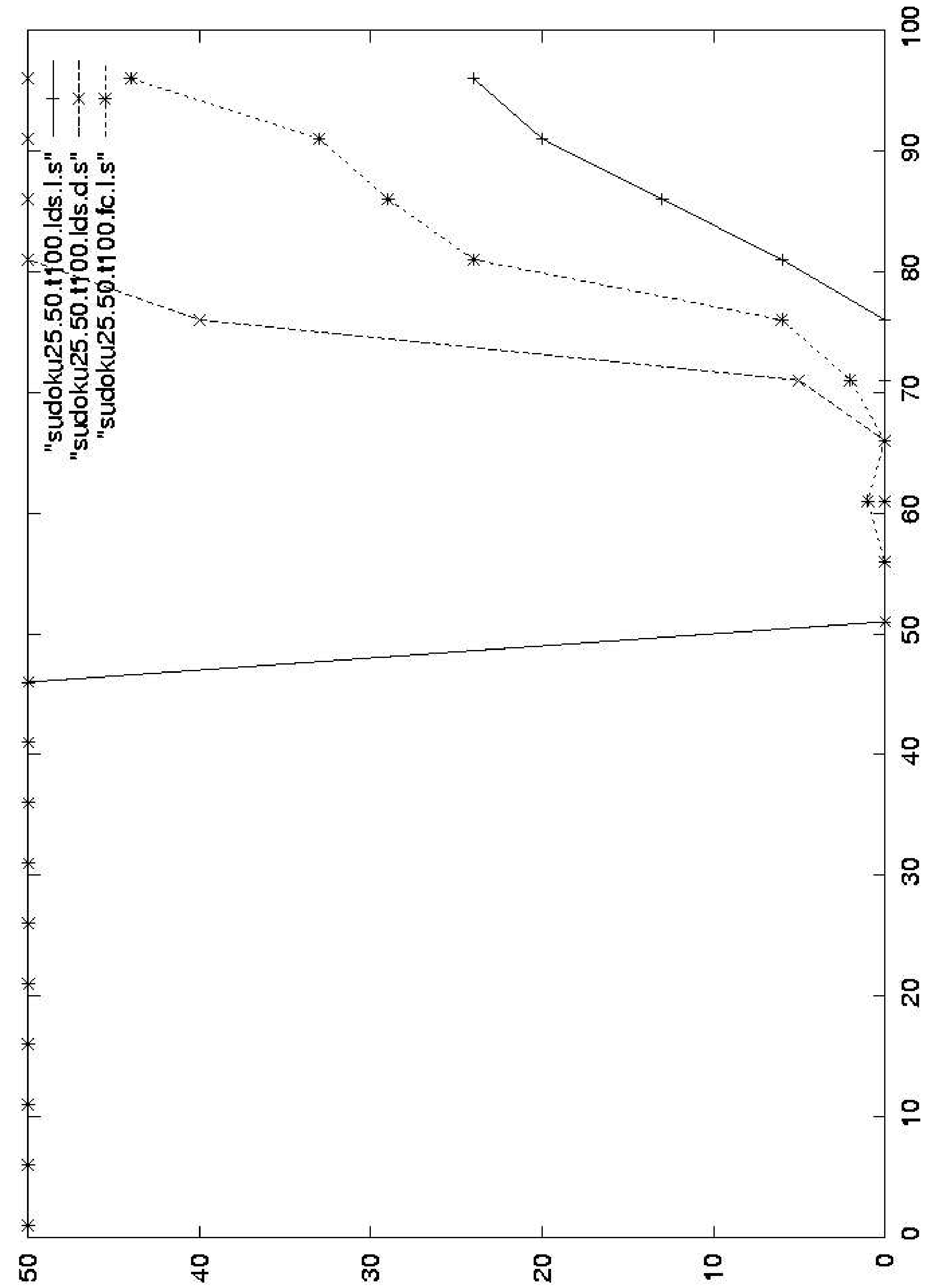

Fig. 4. repartition of solved problems for 25x25 Sudoku