Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439

ANL/MCS-TM-269

Local Search Strategies for Equational Satisfiability

by

Ken Keefe

Mathematics and Computer Science Division

Technical Memorandum No. 269

August 2004

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Prepared in partial fulfillment of the requirements of the Office of Science, DOE Student Undergraduate Laboratory Internship (SULI) Program under the direction of Dr. William McCune in the Mathematics and Computer Science Division at Argonne National Laboratory.

Argonne National Laboratory, a U.S. Department of Energy Office of Science laboratory, is operated by The University of Chicago under contract W-31-109-Eng-38.

Available electronically at http://www.osti.gov/bridge/

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Contents

Local Search Strategies for Equational Satisfiability

by

Ken Keefe

Abstract

The search for models of an algebra is an important and demanding aspect of automated reasoning. Typically, a model is represented in the form of a matrix or a set of matrices. When a model is found that satisfies all the given theorems of an algebra, it is called a solution model. This paper considers algebras that can be represented by using a single operation, by way of the Sheffer stroke. The characteristic of needing only one operation to represent an algrebra reduces the problem by requiring a search through all instances of a *single* matrix. This search is simple when the domain size is small, say 2, but for a larger domain size, say 10, the search space increases dramatically. Clearly, a method other than a brute-force, global search is desirable.

Most modern model-finding programs use a global search; instead of checking every possible matrix, however a set of heuristics is used that allows the search space to be dramatically smaller and thus increases the likelihood of reaching a solution. An alternative approach is local search. This paper discusses several local search strategies that were applied to the problem of equational satisfiability.

1 Introduction

Equational satisfiability problems have been difficult to solve efficiently because of their extremely large search space. A problem with a domain size of 10 contains 10^{100} possible distinct matrices. Such a problem is much too hard to solve by using a global search in which all instances of a matrix are checked. A different strategy is required. One solution is global searches that use heuristics; for example, William McCune's Mace4 [2] uses many heuristics in its model finding. By using various heuristics, global searches can solve equational satisfiability problems quickly. However, such searches run into difficulty as the problem size increases. An attractive alternative is local search because the search space involved in a local search is typically a small portion of the total global search space.

2 Simplifying Bases Using the Sheffer Stroke

An algebra is typically defined by several axioms in first-order logic. From these axioms every property of the algebra can be proven. Such a set of axioms is typically called a basis. Below is a 4–basis for modular ortholattices (MOLs) [3].

$$\begin{aligned}
(x \cup y) \cup z &= x \cup (y \cup z) \\
x \cup \neg(\neg x \cup y) &= x \\
x \cup \neg(x \cup \neg(x \cup y)) &= y \cup x \\
x \cup \neg(\neg y \cup \neg(x \cup z)) &= \neg(\neg(x \cup y) \cup \neg(x \cup z))
\end{aligned}$$

From these four axioms we can derive all the properties of modular ortholattices. However, it is better to represent the entire basis in terms of one operation instead of the two used above ($\cup$ and $\neg$). Using only one operation allows a model to be represented by a single matrix, which in turn reduces the search space significantly. The basis can be written in terms of the Sheffer stroke [3].

The Sheffer stroke works by rewriting any theorem containing the $\cup$ and $\neg$ operations.[1] The Sheffer stroke uses the following equation to transpose the theorems [3].

[1] A different variant of the Sheffer stroke can be used in theorems with $\cap$ and $\neg$.

$$f(x, y) = \neg x \cup \neg y$$

With the preceding definition of the Sheffer stroke, the following basis for MOLs can be obtained.

$$\begin{aligned} f(f(x,x), f(x,y)) &= x \\ f(x, f(x,x)) &= f(y, f(y,y)) \\ f(x, f(f(y,z), f(y,z))) &= f(z, f(f(y,x), f(y,x))) \\ f(x, f(y, f(x, f(z,z)))) &= f(x, f(z, f(x, f(y,y)))) \end{aligned}$$

Now models can be represented by using a single matrix.

3 Local Search

Local search is an attractive strategy for large search problems because it does not blindly go through every possibility. Instead, local search starts with an initial matrix, one believed to be close to a solution, and then tries to make small changes that move the matrix toward a solution.

Local search can be abstractly defined in the following way:

1. Get initial guess (here we used a random matrix as the initial guess).
2. Make a small change, and see how it affects the score.
3. Apply or drop that change depending on the rules of the local search variation.
4. Repeat steps 2 and 3 until a solution is reached or the search is terminated based on the rules of the local search variation.

An excellent example of local search can be found in the root-finding problem. Graphing calculators have come into widespread use lately, and most people who have participated in an algebra class since the dissemination of graphing calculators have probably had to find the roots of a function by graphing it. Once the function has been graphed, the user can have the calculator search for the roots of the function. The calculator first asks the

user to define the bounds of the search and then asks the user to take a guess by moving the cursor to the point on the screen that appears to be the root. This guess is analogous to the initial matrix. The calculator next performs a local search by assuming that the function is continuous at least around the root and by moving to the left or right of the initial guess to see which direction takes it closer to zero.

A key piece of local search strategies is the scoring function. In the graphing calculator example above, the scoring function is the difference between 0 and $f(x)$ for any given x. In other words, it wants to move closer to zero, and it gauges whether it is approaching zero by scoring itself based on the distance (only in terms of the y-axis) from zero. Clearly, if the scoring function is misleading, the local search strategy will fail. If the graphing calculator uses a scoring function that uses the distance (in terms of the y-axis) from 5, for example, the search will not likely find the root. For the algebras discussed in this paper, an intuitive scoring function is used that simply counts the number of times a theorem fails with the current model. This scoring function is explained in more depth in Section 3.1.

3.1 Scoring Function

All of the strategies discussed in this paper use the same type of scoring function. The scoring function iterates over all possible combinations of values of the variables in each axiom. If the axiom does not hold for a combination of the variables, the score is incremented by 1. An important characteristic of this scoring function is that it is clear what the score of a solution matrix will be. If the score is 0, then no combination of the variables, when applied to the axioms, failed, and therefore the matrix is a solution. Take, for example, the single axiom for Boolean algebra [3].

$$f(f(x, f(f(y, x), x)), f(y, f(z, x))) = y$$

In order to score a given model, the scoring function will go through all combinations of x, y, z, for example {(0,0,0),(0,0,1),...,(N-1,N-1,N-1)}, where N is the domain size. It increments the score by one for each combination that does not hold in the current model. Hence, if in the example, $f(f(1, f(f(3, 1), 1)), f(3, f(2, 1))) = 1 \neq y = 3$, then the score will be incremented because the clause failed for that instance.

3.2 Strategies

We formulated and tested many different strategies. The following local search strategies were included, not necessarily because they found solutions frequently or were able to reduce the score, but because they all seem to have some merit. It is hoped that a description of these strategies may inspire new strategies or may encourage someone to apply one of these strategies to other problems.

3.2.1 Greedy Search

The greedy search works the way its name implies. Starting with an initial matrix, it makes every *single* possible change and looks at the change it made to the score. The search then applies the change that yielded the greatest improvement. The search continues until no further improvements can be made. The greedy search never makes a lateral or uphill move (a move that results in an increase in score or no change at all).

3.2.2 Immediate-Improvement Search

The immediate-improvement search works in the same manner as a breadth-first search does. Like the greedy search, it iterates through every cell of the matrix trying every possible value and observing the change in score. The difference between the greedy and immediate-improvement searches is that as soon as a change is made that creates any improvement, the immediate-improvement search applies that change and then starts over. Part of the motivation for this search was that it would perform similar to the greedy search but much faster, because the algorithm applies improvements as soon as they are discovered. Again, this search never makes a lateral or uphill move.

3.2.3 First-Greedy Search

The first greedy search was an attempt to make a hybrid between the greedy and the immediate-improvement searches. The search works in a way that is similar to the immediate-improvement search. It iterates through each of the cells trying every possible change. When it finds a cell that can be changed to improve the score, it tries every possible value of that cell and chooses the value that improves the score the most.

3.2.4 Greedy-Followed-by-Randoms Search

The greedy-followed-by-randoms search was an attempt to defeat the problems of local minima. When the greedy search is applied to a matrix, it usually improves the score significantly; however, the greedy search often is unable to reach a solution. To break away from a local minimum, the greedy-followed-by-randoms search does exactly what its name implies. The search begins by running a greedy search. Once the greedy search terminates, if the score is not zero, then a set of randomly chosen cells is randomized. This search has two parameters that can be adjusted. The first parameter is the number of cells that should be randomized after each greedy search. The second parameter is the number of times a matrix should be randomized. The second parameter acts as a termination condition.

3.2.5 Random-Random Search

The random-random search was implemented after a conversation with Zac Ernst of Florida State University and Branden Fitelson of the University of California - Berkeley [1]. The strategy is simple: Choose a cell of the matrix at random; choose a value at random; if applying that value to that cell yields a score improvement, do it; otherwise, skip it and start over. The termination method used with this strategy was similar to the one used in the simulated annealing search (Section 3.2.6): After a certain number of iterations where no changes are made to the matrix, terminate. Obviously, many possible termination methods can be used.

3.2.6 Simulated Annealing Search

The simulated annealing search strategy was adapted to this problem. Simulated annealing has been mentioned in many papers, including [4]. The simulated annealing search has two parameters: initial temperature (T) and cooling factor (c). The way simulated annealing works is that it chooses a random cell and sets that cell to a random value. It determines the change in score, δ. If the score has improved or has not changed ($\delta \leq 0$), then the new value is kept. Otherwise, that new value is kept only a percentage of the time equal to $e^{-\delta/T}$. The higher the temperature, the more likely an uphill

move will be made. Every time an uphill move is made,[2] the cooling factor is applied to the temperature, usually by setting $T = T * c$. Typically the cooling factor is simply a decimal between 0 and (e.g., 0.9). However, the cooling factor may itself be a function of some other characteristics. Also, if the cooling factor is set to 1, then the temperature will simply remain constant.

The simulated annealing search can be terminated in various ways. Only one method of termination was used in these experiments. When a certain number of uphill moves were cancelled in a row, the search terminated. This seems to be an effective method because it occurs only after the temperature has been reduced enough to allow several uphill moves to be cancelled in a row. Obviously, the search is immediately terminated upon discovery of a solution.

3.2.7 Propagation Search

Many forms of propagation search were tested. The basic method starts with an initial matrix and selects a set of cells known to be correct, for example a set of cells that contain the same values in a solution matrix as they do in the initial matrix. From these "known" cells, the search extrapolates other cells, marks them as "known," and then attempts to use the now larger set of "known" cells to extrapolate more. The hope is that the number of cells that have to be initially "known" will be minimal.

This kind of search is typically applied to single axiom candidates such as

$$f(f(y,x),f(f(f(x,x),z),f(f(f(f(f(x,y),z),z),x),f(x,u)))) = x.$$

The axiom is written in a decomposed fashion that can tell, for given values of (x, y, z, u), whether the cells involved on the left side of the axiom are all "known" cells. When the axiom is decomposed and simplified to the form of $f(a, b) = x$, the cell located at (a, b) cannot be a "known" cell, unless its value already equals x. If the value does not equal x and the cell at (a, b) is not considered "known," then the value is set to x, and the cell at (a, b) is marked as "known."

[2] While this was the method used in these experiments, one can easily imagine that the cooling factor may be applied to the temperature after every iteration, uphill or not.

Clearly, the significant obstacles with this search strategy are how to know what cells to pick and how to know whether they are in a solution. An idea on how to overcome these obstacles is mentioned in Section 5.

3.2.8 Dependencies

Selman, Kautz, and Cohen [4] discuss the desirability of strategies targeting certain cells that contribute to the failure of an instance of the variables. By decomposing a single axiom, cells can be implicated in the failure of an instance. From this information, targets can be chosen that are a direct cause of failing instances.

One search that was experimented with briefly used a *dependency matrix* to choose which cells to change. An evaluation of the matrix produced a dependency matrix. Each cell in the dependency matrix contains a tally of the number of times that its sister cell in the main matrix participated in an instance that failed. The search then uses this dependency matrix to target the cell that is the worst offender and change its value to another value that yields the best score improvement. Interesting, the search strategy did not do very well because typically when that worst offender cell was changed, making the best possible change, the next iteration showed that it was the worst offender again, and usually worse than before.

3.2.9 Least Number Heuristic

The least number heuristic was described by Zhang [5]. This heuristic is more typically used in global search, but an effort was made to see whether it could help the strategies described here.

The least number heuristic works by having a maximum constrained value (MCV), which is initially set to zero. The heuristic reduces the search space by systematically looking at each cell of a matrix and saying that the current cell cannot have a value greater than $MCV + 1$. If the current cell equals $MCV + 1$, then MCV is incremented. If the current cell is less than or equal to MCV, then the heuristic continues to the next cell. The sequence of cells looked at start in the upper left-hand corner and expand outward, in the order $(0,0), (1,0), (1,1), (0,1), (2,0), (2,1), \ldots$.

Matrices that do not adhere to the least number heuristic can be eliminated because they are isomorphic to at least one matrix that does adhere to the least number heuristic.

When this heuristic was applied to the local search strategies stated above, it did not appear to be beneficial and, in fact, usually significantly increased the average score.

4 Results

For this study, a large testbed was devised that allowed the user to experiment with a single matrix. The testbed also was capable of running multiple trials on a search strategy in order to look at the average performance of the various strategies, using thousands of initial matrices. Besides trials with many random initial matrices, a step-back experiment was devised to determine how close to a solution an initial matrix has to be for a strategy to reliably find the solution.

In the step-back experiments and the multiple trials, only the following set of axioms was used in testing. The first axiom was a single axiom candidate for Boolean algebra; the second was an axiom that contradicts commutativity. If a model could be found from the pair, it would prove that the first axiom is not a single axiom for Boolean algebra. The smallest solution has a domain size of 5.

$$\begin{aligned} f(f(y, f(f(x, z), y)), f(x, f(z, y))) &= x \\ f(A, B) &\neq f(B, A) \end{aligned}$$

4.1 Scores, Times, and Solutions

For this portion of testing, a matrix was generated randomly and processed by each of the search strategies. Final scores and times were recorded, and then the whole process was iterated many times.

For Table 1, the process was repeated 100,000 times. The average score of all of the initial matrices was 99.38. The "Avg Final" column displays the average score of the matrices after they were processed by each search. The "Avg Time (msec)" column displays the average time it took for a search to complete. The "Solutions" column displays the number of times a solution was reached by a search.

Table 1: Performance Comparison of Search Strategies

Search Strategy	Avg Final	Avg Time (msec)	Solutions
Greedy	36.89	13.13	7
Immediate Improvement	33.21	9.58	6
First Greedy	34.46	8.63	6
Greedy Followed By Randoms	36.92	13.19	612[a]
Random Random	36.07	10.36	4
Simulated Annealing[b]	10.72	12.6	5

[a]Twenty cells were changed after each greedy search. Also, the process of greedy search followed by randomization was repeated 100 times for each matrix. This explains the significantly larger number of solutions found.

[b]The initial temperature was set to 10 and the cooling factor was 0.9

As the table shows, all of the strategies found approximately the same number of solutions. Moreover, the strategies tended to find a solution from the same initial matrices. In other words, when the greedy search found a solution from initial matrix A, the other search strategies usually found a solution from the same initial matrix. Also, all of the search strategies tended to have approximately the same average final score, except simulated annealing, which had a significantly lower score. However, the *value* of a low score is not yet clear.

4.2 Step-Back Experiments

The step-back experiments started with a solution matrix, made one random change, and then performed the search on the new matrix. If the matrix returned to a solution, it was counted as a success. It then repeated the process of beginning with a solution and making a single change, many times. After it had gone through a set number of times, the process started all over, but now making two changes. The process continued making more and more changes each time until it was changing every cell in the matrix. The objective of this experiment was to indicate how close an initial matrix must be in order for a given search strategy to reach the solution.

Table 2 displays the results from a set of step-back trials that changed the same number of cells 1,000 times. The first row signifies the number of random changes made to the solution matrix. The values in the table indicate how many times the search successfully returned to a solution.

Table 2: Step-Back Experiment Results

Search Strategy	1	2	3	4	5	6	7	8
Greedy	1000	1000	1000	1000	1000	1000	960	741
Immediate Improvement	1000	1000	1000	998	965	804	562	320
First Greedy	1000	1000	1000	999	963	820	617	371
Random Random	987	986	985	979	955	863	598	329
Simulated Annealing	1000	1000	997	959	809	514	268	130

Search Strategy	9	10	11	12	13	14	15	16
Greedy	431	160	29	11	0	0	0	0
Immediate Improvement	148	55	13	12	0	0	1	0
First Greedy	174	50	21	11	2	0	1	0
Random Random	143	48	17	2	4	1	0	0
Simulated Annealing	47	40	13	3	1	1	0	1

Search Strategy	17	18	19	20	21	22	23	24
Greedy	0	0	0	0	0	0	0	0
Immediate Improvement	0	0	0	0	0	0	1	0
First Greedy	1	0	0	0	0	0	0	0
Random Random	0	0	0	0	0	0	0	0
Simulated Annealing	0	0	0	1	0	0	2	0

From this data, one can see that greedy search gets to a solution reliably when the solution is eight or nine steps away. The other searches do not perform as well.

5 Conclusion and Future Work

The local search strategies studied here do not appear to be the "silver bullet" for equational satisfiability problems. These search strategies tended to find solutions only about 5 or 6 times out of 100,000. If a matrix was close to a solution, however, these strategies could be useful. This is actually the way local search is supposed to work: a guess is made that is close to the solution and the search finds the solution in the neighborhood of the guess. Choosing matrices randomly does not suffice as an acceptable guess.

Nevertheless, local search may still have merit. Work should be done with different algebras and different types of problems. One might investigate a way to generate a matrix that is known to be close to a solution. Also, the scoring function may be the problem. Perhaps a different mechanism is needed for determining whether the search is heading in the right direction. Many aspects play a role in the performance of the strategies, and each should be researched further.

More work should be done on the propagation search. The propagation search can be successful in the same way that the other local searches are successful. The problem appears to be in the initial factors, namely, which cells are "known." If there was a mechanism to determine that several cells are correct, the propagation search would at least be able to fill out a portion, if not the rest, of the matrix.

One possible solution is to start with a certain number of distinct initial matrices, perhaps 50. One could perform a greedy search in parallel on all 50. Looking at the 50 new matrices, one could compare each of the cells that are in the same positions in the matrices, determine which values in which cells occur most frequently, and then perform a merging operation that puts the most common value in each of the cells. For example, if among the 50 matrices the value 3 appears in the cell at (2,4) 46 times and the value 2 appears in that cell 4 times, then one might be able to infer that the value 3 is likely to appear in cell (2,4) in a solution matrix. So, once the matrices have been merged, each cell should be ranked by which cell has a value that is most likely to appear in a solution matrix. The propagation search could

then performed using the cells that are most "likely" as the "known" cells.

This idea is not without problems. One problem that immediately disables this strategy as it currently exists is isomorphism. If a set of axioms has a solution matrix, it actually has several. There exist several matrices that are isomorphic to the solution matrix, and they too are solutions. So, the problem with this parallel greedy search and merge idea is that 10 of the greedy searches may have gotten several cells right that exist in one isomorphic solution, another 10 matrices may have several cells that are correct for a different isomorphic solution, and so on. This appears to be a difficult problem to tackle.

Another option for the propagation search is to choose a set of cells and then iterate through every combination of values for those cells, trying to fill in the rest of the matrix. Experiments with this option showed some promise, but they usually required half of the cells to be marked as "known." However, this type of search is not really a *local* search strategy, so it is beyond the scope of this paper.

Further research should be done in this area. The problems are clearly difficult, and local search seems to have the makings of an elegant solution.

References

[1] Ernst, Z., Fitelson, B., Discussion, Argonne Workshop on Automated Reasoning and Deduction (AWARD-2003), July 13, Mathematics and Computer Science Division, Argonne National Laboratory, 2003.

[2] McCune, W., *Mace4: Models And Counter–Examples*, Argonne National Laboratory, 2003. `http://www.mcs.anl.gov/AR/mace4/`

[3] McCune, W., Rose, M., and Veroff, R., *Short Equational Bases for Ortholattices: Web Support*, Argonne National Laboratory, 2003. `http://www.mcs.anl.gov/~mccune/papers/olsax/`

[4] Selman, B., Kautz, H., Cohen, B., *Noise Strategies for Improving Local Search*, AT& T Bell Laboratories, Murray Hill, N.J., 1994.

[5] Zhang, J., *Constructing Finite Algebras with FALCON*, Journal of Automated Reasoning, vol. 17, no. 1, pp. 1–22, 1996.