World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:10, 2007

A Hybrid Search Algorithm for Solving Constraint Satisfaction Problems

Abdel-Reza Hatamlou, and Mohammad Reza Meybodi

***Abstract*—**In this paper we present a hybrid search algorithm for solving constraint satisfaction and optimization problems. This algorithm combines ideas of two basic approaches: complete and incomplete algorithms which also known as systematic search and local search algorithms. Different characteristics of systematic search and local search methods are complementary. Therefore we have tried to get the advantages of both approaches in the presented algorithm. The major advantage of presented algorithm is finding partial sound solution for complicated problems which their complete solution could not be found in a reasonable time. This algorithm results are compared with other algorithms using the well known n-queens problem.

***Keywords*—**Constraint Satisfaction Problem, Hybrid Search Algorithm.

I. Introduction

MANY problems in the fields of artificial intelligence, network, database, engineering and other areas of computer science can be viewed as special cases of constraint satisfaction problems, including image processing, natural language parsing, routing, circuit design, scheduling and more. A CSP is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take [7]. The search space of CSPs is often exponential. Therefore a number of different approaches to the problem have been proposed to reduce the search space and find a feasible solution in a reasonable time. Based on the search space exploring and variable selection heuristics, those algorithms can be divided into two major groups: complete and incomplete algorithms.

Complete algorithms [2], [7] seek any solution or all solutions of a CSP. Or they try to prove that no solution exists. These algorithms also divided into subgroups. The consistency (or constraint propagation) techniques try to eliminate values that are inconsistent with some constraints. There are many consistency techniques ensuring different levels of consistency. The systematic search methods explore systematically (and exhaustively) the whole search space. Most common are so called tree search methods. They view the search space as a search tree. Each node represents mutually exclusive choices which partition the remaining search space into disjoint sub-spaces. This structure enables to remember with acceptable memory requirements, which parts of the search space have already been visited. Usually, a node corresponds to an assignment of a particular value to a particular variable. The efficiency of tree search methods may be significantly improved with the help of consistency techniques. There are two basic ways of employing consistency techniques together with tree search methods. Look back schemes are invoked when the algorithm is preparing to backtrack after encountering a dead-end. Look ahead schemes can be employed whenever the algorithm is preparing to assign a value to the next variable.

Incomplete search methods [4], [8] do not explore the whole search space. They search the space either non-systematically or in a systematic manner, but with a limit on some resource. They may not provide a solution but their computational time is reasonably reduced. They can not be applied to find all solutions or to prove that no solution exits. However, they may be sufficient when just some solution is needed. Another application is to seek a feasible solution of an optimization problem. There are two basic approaches for incomplete search. The constructive algorithms gradually extend a partial solution to a complete one. They are based on tree search algorithms for satisfaction problems and may benefit from constraint propagation. Iterative repair methods start with an initial solution, found by some other approach. Or it may be found randomly. They incrementally alter the values to get a "better" one, and eventually an optimal or at least a good enough solution. Their non-systematic nature generally voids the guarantee of "completeness", but they are often able to get quickly close to the optimal solution and overcome the algorithms based on tree search. These algorithms may be applied to satisfaction problems, as well. The search will move throughout all assignments and the quality of the assignment will be determined by the number of violated constraints. Important iterative repair methods are the so called local search methods [4], [8]. Local search is based on making small (local) changes in assignments to variables. A way of moving from one solution to another is called a move. A move is problem-specific. A set of all solutions that differ from the current one in only one move is called a neighborhood. At each iteration step, a solution is selected from the current neighborhood (typically a better one). And it

Manuscript received September 17, 2007. This work was supported by Islamic Azad University – Khoy branch.

Abdel-Reza Hatamlou is with Islamic Azad University, Khoy 58135-175, Iran (phone: +98 461 2225704; e-mail: reza_hatamloo@yahoo.com).

Mohammad Reza Meybodi is with the Computer Engineering Department, Amirkabir University of Technology Tehran 15875-4413, Iran (e-mail: mmeybodi@akt.ac.ir).

becomes the current solution. If the algorithm selected only better solutions all the time, it could get entrapped in so called local optimum. It is a solution whose neighborhood contains only solutions with equal or worse cost, and it is not the optimal solution (global optimum). To avoid this state, local search is often equipped with various meta-heuristics randomizing the search, i.e., allowing also worse neighbors to be selected under certain conditions the algorithm is usually bounded by some stopping criterion. It may be based on computational time, number of moves or some other indicator.

Systematic search algorithms suffer from the disadvantage of tree search. If a wrong decision is made early in the search tree, it is necessary to undone it by backtracking. Many so far assigned values are thrown away. Look-back and look-ahead schemes try to reduce the importance of this problem, but they can not solve it completely.

The local search algorithms decide upon complete solutions and the absence of systematicity allows them to modify assignments in any order. However, when a problem is tightly constrained, a local search algorithm may not get to a solution. Either the initial solution may be too far from it, or it may get entrapped in a local optimum.

While algorithms based on tree search are due to constraint propagation effective at finding solutions for tightly constrained problems with complex and overlapping constraints, the local search methods can be superior at optimization problems that are loosely constrained.

II. Hybrid Search Algorithm

The above mentioned considerations lead to conclusion that the different characteristics of systematic search and local search methods are complementary. Therefore promising hybrid algorithms trying to get the advantages of both approaches have been proposed recently [1], [5], [6].

The hybrid search algorithm, that we propose here, is based on ideas of local search methods [4], [8]. However, in contrast to classical local search techniques, it operates over feasible, though not necessarily complete solutions. In such a solution, some variables can be left unassigned. Still all hard constraints on assigned variables must be satisfied. Similarly to systematic search algorithms, this means that there are no violations of constraints. Working with feasible incomplete solutions has several advantages compared to the complete infeasible assignments that usually occur in local search techniques. For example, when the algorithm is not able to find a complete solution, an incomplete (but feasible) one can be returned, e.g., a solution with the least number of unassigned variables found. Moreover, because of the iterative character of the search, the algorithm can easily start, stop, or continue from any feasible solution, either complete or incomplete.

The search proceeds iteratively. See Fig. 1 for algorithm. During each step, an unassigned or assigned variable is initially selected. Typically an unassigned variable is chosen like in systematic search. An assigned variable may be selected when all variables are assigned but the solution is not good enough. Once a variable is selected, a value from its domain is chosen for assignment. Even if the best value is selected, its assignment to the selected variable may cause some conflicts with already assigned variables. At this point this algorithm decides to accept or reject the current variable and its value. If the number of conflicts was reasonable, such conflicting variables are removed from the solution and become unassigned and the selected value is assigned to the selected variable. But, if the number of these conflicting variables was great, current variable shall be rejected and a new variable shall be selected. The reasonability for number of conflicts is a heuristic problem. For presented algorithm we have defined 20 percentages for reasonability. This means that if the number of conflicts between current variable and assigned variables was great than 20% of all assigned variables, the current variable must be rejected. The algorithm attempts to move from one (partial) feasible solution to another via repetitive assignment of a selected value to a selected variable. During this search, the feasibility of all constraints in each iteration step is enforced by unassigning the conflicting variables. The search is terminated when the requested solution is found or when there is a timeout, expressed, e.g., as a maximal number of iterations or available time being reached. The best solution found is then returned.

```
procedure hybrid_search(unlabeled, answer, max_repeat, max_conflict)
    /* unlabeled is a list of un-labeled variables and answer is an  incomplete
                        answer (empty at the start) */
    repeats=0;
    while unlabeled not empty & repeats<max_repeat
      repeats ++;
      variable = Variable_Chooser(unlabeled, answer);
      unlabeled -= variable;
      value = Value_Chooser(variable, answer);
      conflicts = Conflict_counter(variable, value,answer);
      if conflicts < max_conflict then
        unlabeled += label(answer, variable, value);
          /* label the variable and return conflict variables */
      else
        unlabeled += variable;
    end while;
    return answer;
  end hybrid_search
```

Fig. 1 A kernel of the presented hybrid search algorithm

The above algorithm schema is parameterised by three functions: the variable chooser, the value chooser and the conflict counter.

There are several guidelines how to select a variable. In local search the variable participating in the largest number of violations is usually selected first. In systematic search algorithms, the first-fail principle is often used, i.e., a variable whose instantiation is most complicated is selected first. This could be the variable involved in the largest set of constraints or the variable with the smallest domain etc [3], [4], [7]. It is possible to select the worst variable among all unassigned variables but due to complexity of computing the heuristic value, this could be rather expensive in some cases. Therefore in the presented algorithm we select a subset of unassigned variables randomly and then select the worst variable from

this subset. The results will not be much worse and we can select the variable much faster.

After selecting the variable we need to find a value to be assigned. Typically, the most useful advice is to select the best-fit value [3], [4], [7]. So, we are looking for a value, which is most preferred for the variable and also which causes the least trouble. It means that we need to find a value with minimal potential future conflicts with other variables. To remove cycling, it is possible to randomize the value selection procedure. For example, it is possible to select five best values for the variable and then choose one of them randomly.

III. Results

In this section we will present the efficiency of the described hybrid search algorithm on the N-queens problem. We will compare the achieved results with another local search and complete search algorithm. All presented results were measured on Intel Pentium IV 2.4GHz, 256MB RAM, Windows XP.

The N-queens problem is to place n queens on a n×n chessboard so that no queen is under a direct attack from any other one.

The algorithms which we used in this comparison are min-conflicts [4], [7], [8] and full look ahead [7]. Min-Conflicts algorithm is one of the typical local search algorithms and it is highly efficient on the N-queens problem and full look ahead algorithm is one of the complete search algorithms.

The time for solving the N-queens problem is given in the Table I. It is not surprising that the local search algorithms are much more efficient than complete search on this problem. So, let's focus only on the differences between the presented hybrid search algorithm and the minimal conflicts algorithm.

As we can see on Fig. 2, the presented algorithm is nearly as fast as the min-conflicts algorithm.

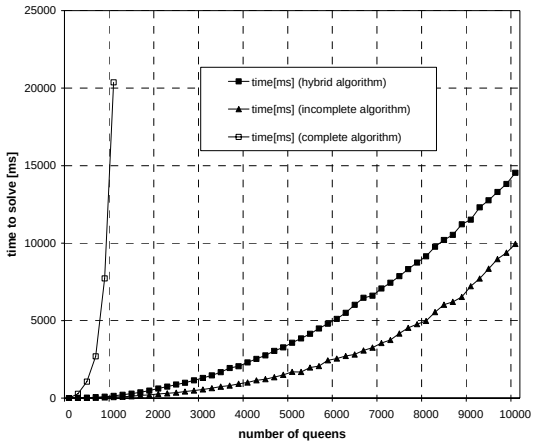

Fig. 2 Comparison of the time for the described algorithms; mean value from 10 measurements

TABLE I
Time to Solve the N-Queens Problem; Mean Value from 10 Measurements

Number of Queens	Hybrid Algorithm	Incomplete Algorithm	Complete Algorithm
1000	0.150 s	0.070 s	20.372 s
2000	0.623 s	0.271 s	---
3000	1.305 s	0.562 s	---
4000	2.037 s	1.013 s	---
5000	3.575 s	1.696 s	---
6000	5.115 s	2.551 s	---
7000	7.072 s	3.552 s	---
8000	9.149 s	4.982 s	---
9000	11.516 s	7.220 s	---
10000	14.543 s	9.952 s	---

IV. Conclusion

We have presented a hybrid approach for solving constraint satisfaction and optimization problems which combines ideas of local search and systematic search algorithms. The basic motivation was to design an algorithm for solving complicated problems which their complete solution could not be found in a reasonable time. We have compared the efficiency of the described hybrid search algorithm with another local search and systematic search algorithm on the N-queens problem. The achieved results show that the presented algorithm is nearly as fast as the local search algorithm. We believe that this algorithm can be successfully applied to many constraint satisfaction problems especially when working with sound incomplete solutions is needed in general.

References

[1] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. Artificial Intelligence, 139(1):21–45, 2002.
[2] Vipin Kumar. Algorithms for constraint satisfaction problems: A survey. AI Magazine, 13(1):32–44, 1992.
[3] K. Marriot, P. J. Stuckey. Programming with Constraints: An Introduction. The MIT Press, 1998.
[4] Z. Michalewicz and D. B. Fogel. How to Solve It: Modern Heuristics. Springer-Verlag, 2000.
[5] W. Ruml. Incomplete tree search using adaptive probing. In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), 2001.
[6] Andrea Schaerf. Combining local search and look-ahead for scheduling and constraint satisfaction problems. In Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pages 1254–1259, Nagoya, Japan, 1997.
[7] E. Tsang. Foundations of Constraint Satisfaction. Academic Press, 1993.
[8] Stefan Voß. Meta-heuristics: State of the art. In Alexander Nareyek, editor, Lcal search for planning and scheduling: revisited papers, pages 1–23. Springer-Verlag LNCS 2148, 2001.