Solving a Constraint Satisfaction Problem through Iterative Reordering Augmentation

Wee-Kit Ho\textsuperscript{1} and Andrew Lim\textsuperscript{2}

\textsuperscript{1}School of Computing, Nat’l Univ of Singapore, 3 Science Dr 2, Singapore 117543
Dept of IEEM, HKUST, Clear Water Bay, Kowloon, Hong Kong
Email: howk@comp.nus.edu.sg; iealim@ust.hk

1. Introduction

Industrial scheduling, exam timetabling and nurse rostering are some of the many real-life problems that belong to an important class of problems, commonly known as Constraint Satisfaction Problems (CSP)\cite{1}. These problems involve the assignment of appropriate values from a limited domain to a set of variables, subject to a number of constraints that restrict the values that variables can take simultaneously. CSP is important to the fields of Artificial Intelligence, Operations Research, and Computer Science in general since many search problems can be expressed as CSPs.

Generally, CSP has three components: \textit{Variables}: a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of $n$ variables; \textit{Values}: Each variable $v_i$ takes its values from an associated finite domain $D_i$; \textit{Constraints}: A set $C$ of constraints restricting the values that the variables can take simultaneously.

The set of constraints may include both hard and soft constraints. A feasible solution to a CSP is an assignment of values to variables such that all hard constraints are satisfied. The relative quality of two feasible solutions is measured by some function of the soft constraints satisfied. The given constraint satisfaction problem may require finding one or all solutions, or an optimal/good solution given some objective functions.

One of the most common algorithms used for solving CSP is backtrack search. This is an exhaustive systematic search algorithm that attempts to extend partial solutions while maintaining overall feasibility. Unfortunately, the computationally intractable nature of general CSP means that applying backtrack search to its fullest extent would take an exponential amount of time in the worst case. This paper introduces an alternative method of solving CSP, called \textit{Iterative Ordering} search. Unlike backtrack search, which maintains a partial solution, Iterative Ordering search performs several iterations of variable assignment, varying the order in which the variables are assigned according to information obtained in the previous iteration. Our experiments have shown that Iterative Reordering outperforms backtrack search both in terms of solution quality and computation time in the university examination timetabling problem.

2. Backtrack Search

The key feature of CSP is that the domains are finite. Simple systematic search algorithms can be devised that eventually finds a solution, if such a solution exists, and terminates. A CSP can be solved using the generate-and-test paradigm (GT). This method systematically generates each possible value assignment and then tests to see if all constraints are satisfied. A more efficient and commonly used algorithm for performing systematic search is the backtrack search (BT). This method incrementally attempts to extend a partial solution towards a complete
solution by repeatedly choosing a value for a currently unassigned variable. If a situation occurs where there are no suitable values for all remaining unassigned variables, then the most recent variable assignment is undone, and another assignment is attempted. In effect, backtracking is done to correct erroneous assignment of values.

However, many problems posed in the constraint satisfaction problem model are computationally intractable (NP-hard). Hence, performing exhaustive search will take an amount of time exponential to the problem size in the worst case on many constraint satisfaction problems. One alternative way to address this issue of intractability may be to restrict the algorithm from backtracking relentlessly upwards. A certain maximum height value is specified so that if further backtracking beyond this height is required, it will be ignored and the variable at that level of the search tree will be considered unassigned while the search continues. However, this approach will not guarantee a solution even if one exists.

Backtrack search may also include some advanced constraint satisfaction techniques. There are basically two important issues that need to be addressed when implementing a search algorithm for constraint satisfaction, namely the early detection of conflicts and the order in which the variables are considered and values are assigned.

3. Consistency Algorithm
Late detection of inconsistency is a common problem in many CSP search algorithms as they are unable to detect a conflict before it occurs, that is, before assigning values to all the variables of the conflicting constraint. This may cause much redundant work to be done during a backtrack search. To cope with this, various consistency techniques [2][3][4][5][6][7] for constraint graphs have been devised to prune the search space. A consistency-enforcing algorithm makes any partial solution of a small sub-network extensible to some surrounding network. Thus, inconsistency is detected as soon as possible. Consistency techniques range from simple node consistency and the very popular arc consistency to full but expensive path consistency. Forward-check and Look-ahead search algorithms are two such algorithms that embeds a consistency algorithm within a backtracking algorithm.

Variable/Value Ordering
The second issue to be addressed is the order in which variables are considered as well as the order in which values are assigned. Experiments have shown that the ordering of variables and values has significant impact on the complexity of backtrack search. Thus choosing a good order may improve the efficiency of constraint satisfaction. Generally, two types of ordering schemes are used. Static ordering specifies the order of the variables/values to be considered/assigned before the search starts. In contrast, dynamic ordering chooses the next variable/value to be considered/assigned based on the current state of the search. Dynamic ordering may not be feasible in some search algorithms, in which extra information is not available during the search. However, a backtrack search that uses a consistency algorithm is well suited to dynamic ordering schemes. For instance, during the search process, some of variables’ domains are pruned as a result of the consistency checks. Given this information, a choice on the next variable to be considered may be made based on the new availability of the domain of the variables.

In view of these available existing techniques, the most commonly used method for constraint satisfaction today is to combine backtrack search with consistency check and dynamic variable ordering.
ordering.

4. Iterative Reordering

While getting good orderings can noticeably reduce the number of unnecessary backtracks (i.e. thrashing), there is no easy algorithm that obtains the best ordering that minimizes the number of backtracks. In addition, the efficiency of backtrack search depends very much on the selection of the first few variables. If the initial choices are not well chosen, the number of backtracks required may be tremendous as effective pruning can be limited. Even with dynamic ordering, backtrack search is likely to make poor choices especially on the first few variables where conflict detections and domain reductions are inadequate.

In view of the limitations of backtrack search for solving CSP and the importance of ordering and consistency checks, we introduce the Iterative Reordering Search technique for CSPs. This search algorithm is mainly based on the Squeaky Wheel Optimization (SWO) [8] framework.

In this simple algorithm, we first place all variables to be considered in a priority queue. The priority value of a variable is based on some predefined measure. For instance, we can choose a dynamic ordering measure in which variables with smaller domain availability are given higher priority, as it is likely to be more difficult to assign values to these variables. It should be noted that this ordering is dynamic because, when the current variable is being assigned, the consistency algorithm will reduce the domains of its related constraint variables, thereby increasing their priority values. Based on this ordering, we try to assign a value to each of these variables. However, in the process of assignment, the domain of some variables may be completely reduced by the consistency algorithm, resulting in no feasible solution. After the first round of variable assignment, we increase the priority values of those unassigned variables by a fraction. This will move those variables that failed to be assigned forward in the sequence so that they have a higher chance of being assigned in the subsequent rounds. We repeat the process of variable assignment for all variables using the new ordering. This process is performed iteratively until all variables are assigned or after a pre-specified maximum number of iterations. This process is illustrated in Figure 1.

Iterative Reordering search avoids the greatest pitfall of backtrack search, namely the inability...
to recover from a poor allocation early in the process. Since each pass of Iterative Reordering search begins from scratch with the newly adjusted priority values, a poor early allocation can be corrected. In effect, Iterative Reordering search offers a global view of the problem that backtrack search lacks. Another way to view Iterative Reordering search is to note that the purpose of backtrack search is to correct earlier mistakes in value assignment. Instead, Iterative Reordering search tackles the problem by changing the value assignment order so that the mistakes are not made in the first place.

Furthermore, experience has shown that the vast majority of the computation time of backtrack search is used correcting erroneous value assignments (i.e. the actual backtracking process). Since Iterative Reordering search eliminates the backtracking component, it would be reasonable to expect Iterative Reordering search to be significantly faster than backtrack search, provided that variable reordering is an efficient way of finding superior solutions to the problem.

5. The University Examination Timetabling Problem (ETTP)

The university examination timetabling problem (ETTP) is an instance of hard CSP. The task involves scheduling a set of examinations into a set of venue-session slots, subject to a number of constraints. Hard constraints include (1) two examinations that are taken by any particular student cannot be scheduled in the same session (i.e. time conflict constraint); (2) the total number of candidates taking the papers scheduled in a particular venue must not exceed the venue’s capacity; and (3) user-defined constraints.

The following experiments were performed as part of the development of a campus-wide automated university examination scheduling application [9], UTTSExam, for the National University of Singapore (NUS) [10]. NUS is divided into nine faculties, with approximately 24000 students taking 1000 examinations per semester. NUS introduced the modular academic course structure in 1993. This allowed students to choose the modules that they wished to study in order to complete their degree requirements. As a result of this added flexibility, the task of scheduling the examination timetables in NUS becomes more complex, especially in view of the increasing number of cross-faculty modules (i.e. modules that can be taken by student from different faculties).

In NUS, all main examinations are to be scheduled in a total of 34 sessions (3 sessions on weekdays, 2 sessions on Saturday) over a period of two weeks. The data used was from both semesters of the 2001/2002 Academic Year. Table 1 shows some of the statistics on the data for each semester. In our experiments, a pool of venues with a total of 4700 seats per session is available.

<table>
<thead>
<tr>
<th>Table 1 Statistics of Examination Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Academic Year</td>
</tr>
<tr>
<td>Semester</td>
</tr>
<tr>
<td>Number of Students</td>
</tr>
<tr>
<td>Number of Examinations</td>
</tr>
<tr>
<td>Number of Student-Exam Tuples</td>
</tr>
<tr>
<td>Number of Time conflict Constraints</td>
</tr>
</tbody>
</table>

Kyoto, Japan, August 25–28, 2003
6. Experimental Details
In our experiments, we investigate two algorithms. The first algorithm is backtrack search embedded with a node-and-arc consistency algorithm and dynamic variable ordering (BT). In our backtrack algorithm, we set the maximum backtrack height to be 10. In addition, a composite measure used to dynamically order the examinations (variables) is as follows:

\[ \text{Exam-Priority} = \alpha \cdot \text{Measure1} + \beta \cdot \text{Measure2} + \gamma \cdot \text{Measure3} \quad \text{where } \alpha + \beta + \gamma = 1 \]

- **Measure1** is based on the number of students taking this exam
- **Measure2** is based on the constraint degree of this exam (i.e. the number of other papers affected by the scheduling of this paper)
- **Measure3** is based on the number of timeslots that cannot be used for scheduling this exam, due to one or more constraint conflicts

In **Measure1**, we believe that large examinations should be scheduled first. If large examinations are scheduled at a later stage, there will be a higher chance that there will not be any suitable timeslots available mainly due to venue capacity constraints. **Measure2** attempts to schedule highly constrained examinations first since they are likely to be the most difficult to find a value. **Measure3** is based on the ordering strategy of domain availability as described earlier.

The second algorithm is the Iterative Reordering (IR) search heuristic. Specifically, this heuristic also embeds a node-and-arc consistency algorithm and uses the same variable ordering method as the backtrack algorithm. As mentioned, variables that are not assigned in the current iteration will get an increase in priority value by a constant factor. Also, the maximum number of iterations is set to be 50.

Both the algorithms were implemented in Java Development Kit 1.3. These experiments were carried out on an AMD Athlon 1.33 GHz PC. All computational times are rounded to the nearest second.

7. Results and Analysis
Table 2 compares the experimental results when backtrack search and iterative reordering were used to schedule the examination timetables for the two semesters.

<table>
<thead>
<tr>
<th>Academic Year</th>
<th>2001/2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semester</td>
<td>I</td>
</tr>
<tr>
<td>Number of Examinations</td>
<td>1049</td>
</tr>
<tr>
<td>Algorithm used</td>
<td>BT</td>
</tr>
<tr>
<td>Number of assigned examinations</td>
<td>1046</td>
</tr>
<tr>
<td>Number of unassigned examinations</td>
<td>3</td>
</tr>
<tr>
<td>No of iterations used</td>
<td>-</td>
</tr>
<tr>
<td>Time Taken (seconds)</td>
<td>146</td>
</tr>
</tbody>
</table>

From Table 2, we can clearly see that Iterative Reordering outperforms Backtrack Search, both in terms of the number of examinations (variables) successfully scheduled as well as the amount of computational time required. The reason why Backtrack Search is unable to find a
feasible solution may be mainly due to the limitation of a maximum backtrack height of 10. However, if we increase the maximum backtrack height linearly, the computational time required by backtracking will increase exponentially.

On the other hand, not only did Iterative Reordering find feasible solutions for both semesters, it also requires substantially less computational time compared to backtracking. Furthermore, this small amount of computational time is mainly attributed to the fact that only a few reordering iterations were required to reach feasibility in our experiments, and also because the computation time increases linearly with the number of iterations performed. In addition, the few iterations needed also support the claim that ordering of variables can greatly influence the complexity of the overall search process. Even though Backtrack Search uses dynamic ordering, it may not be sufficient as mistakes made in choosing the earlier variables may only be detected when assigning variables that appear much later in the search. This again is closely related to our maximum backtrack height. Intuitively, if a mistake is made at level $m$, and is only detected at level $n$, then this mistake cannot be undone by backtracking if the difference of $n$ and $m$ exceeds the specified maximum backtrack height.

8. Conclusions
The question is: Is backtracking really needed in CSP? Backtracking may still be the preferred choice for simple CSP as it guarantees a solution if one exists. However, when dealing with hard CSPs, many other considerations should be made. Nevertheless, we should consider alternative approaches to solving CSP. In this paper, we have introduced an alternative algorithm, Iterative Reordering, which is a simple yet efficient heuristic in dealing with CSP. Though we made no attempt to provide a qualitative comparison with backtracking, nevertheless Iterative Reordering has shown great potential and consistently produces favorable results in our experiments.

References

Kyoto, Japan, August 25–28, 2003