Extended analysis of intelligent backtracking algorithms for the maximal constraint satisfaction problem

Srinivas Padmanabhuni
Department of Computing Science
University of Alberta, Edmonton
Alberta, Canada, T6G 2H1
srinivas@cs.ualberta.ca

Abstract

Overconstrained systems refer to sets of soft constraints which do not permit a solution satisfying all the constraints. The overconstrainedness in such soft constraint systems can be manifested in a wide variety of structures including weighted constraints, partially ordered constraints, constraint hierarchies and references. The simplest among these formalisms is the maximal constraint satisfaction problem, where a solution is sought which satisfies the maximum number of constraints. In this treatise, backtracking algorithms and their intelligent versions used in ordinary constraint satisfaction(CSP) context, are studied in context of maximal constraint satisfaction problem. The algorithms by Freuder and Wallace in [1] for depth-first branch and bound and backjumping, are extended to conflict-directed backjumping. A theoretical analysis of the problem of application of intelligent backtracking algorithms for maximal CSP is provided.

1 Introduction

A constraint satisfaction problem (CSP)[4] refers to the problem of finding values to a set of variables, subject to constraints on the acceptable combination of values. A solution to a CSP is a set of variable-value assignments, which satisfies all members of the set of constraints in the CSP. In some situations, it is not possible to find a solution satisfy all the constraints belonging to a CSP. Such problems are termed as overconstrained problems.

The absence of a solution satisfying all the constraints in the CSP, prompts the need to relax the requirements of the CSP. An overconstrained problem typically involves standard ways of relaxing the requirements of the constraints, to allow for a reasonable solution. The common mode of relaxation includes embedding a priority information among the set of constraints, so that more important constraints are satisfied over the less preferred ones. This priority information can be embedded in the constraints in the form of a partial order on the constraints, or in the form of assignment of weights to the constraints or in the form of constraint hierarchy.

Orthogonal to the embedding of priority information in the constraints, another school of thought in overconstrained systems assumes all constraints to be of equal weight and search is made for a solution satisfying a maximal subset of the set of constraints. This overconstrained CSP specification is termed as the maximal constraint satisfaction problem (max-CSP). In max-CSP, the set of constraints satisfied by any solution of the problem cannot be expanded further without an inconsistency.

Unless otherwise specified, in this treatise, the discussion shall be limited to binary CSP's. By binary CSP's, it is meant the constraints in the CSP involve only two variables.

2 Intelligent backtracking and its applicability to max-CSP

2.1 Background

In this section, the background material concerning application of branch and bound methods and its variants to the maximal CSP are considered [1]. The material in this section is cited from [1]. In naive backtracking for constraint satisfaction problems(CSP's), a partial assignment of values to the variables is consistently expanded till a dead-end is encountered. A dead-end refers to a situation when any value assigned to the latest variable does not lead to a solution. In such a case, backtracking occurs and the next possible value for the immediately previous variable is tried till a value assignment to all variables is reached which satisfies all the constraints in the CSP.

2.1.1 Depth-first branch and bound (DFBB)

Backtracking clearly indicates that we need to go through an exponential number of nodes in the worst case to solve the CSP. This prompts the possibility of extension of backtracking based algorithms for the problem of computing one solution to the max-CSP. It is clear from the definition of the max-CSP, that naive backtracking will ultimately end for a max-CSP, without returning any solution because the CSP is known to be insoluble. Thus backtracking cannot be used directly for the maximal constraint satisfaction problem (MCSP)[1].

In [1], it was shown that the natural analogue of the naive backtracking algorithm for maximal CSP was the depth-first branch and bound algorithm. In depth-first branch and bound(DFBB) for max-CSP, similar to the process in backtracking, a partial solution is expanded along the way. But unlike the backtracking algorithm, a partial solution is expanded even on encountering of an inconsistent solution. A counter N is kept for the current best solution during the solving process, and a counter C keeps track of the number of constraints violated by the current assignment of values. We may begin with a large N, close to infinity. Then on encountering the first assignment of values to all variables, this figure N is updated by C, the number of constraints updated by the current solution. Then search backtracks and proceeds with a different value for the preceding variable and C is recomputed. On encountering a partial assignment of values during the search process, which violates $\geq$ N constraints, search along that path is cut short and backtracking occurs to the next value of the preceding variable. This is because any further assignment along the same path cannot lead to a better solution.

Consider the max-CSP shown in Figure 1. Here we use two variables C and N and S which represent the number of constraints violated by the current assignment of values, the number of constraints violated by the current best solution and the current solution respectively.

The algorithm begins with the value N = infinity. Then we proceed along X1=1,X2=1 till X3=1 without any problem. But X3=1 is inconsistent with X1=1. In naive backtracking this would have resulted in a backtrack to X3=2. But here we proceed further and try X4=1. Here we have a possible solution which violates 4 constraints. Hence N is now updated by 4 and S, the current best solution is $\{X1 = 1, X2 = 1, X3 = 1, X4 = 1\}$. Next we backtrack and reach X4=2 and S now becomes $\{X1 = 1, X2 = 1, X3 = 1, X4 = 2\}$ as the number of inconsistencies in this solution is 3, which is less than 4, the current value of N, and N is updated to 3.

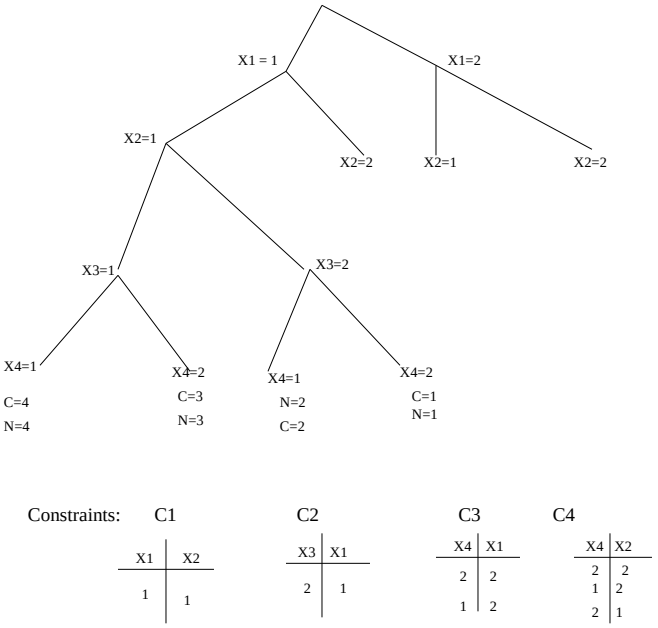

Figure 1: Depth First Branch and Bound for Problem 1 with constraints C1,C2,C3 and C4.

Further backtrack to X3=2,X4=1 gives an even better solution S=$\{X1 = 1, X2 = 1, X3 = 2, X4 = 1\}$, which gives N=2. In the next branch, this result is further improved with S=$\{X1 = 1, X2 = 1, X3 = 2, X4 = 2\}$ giving N=1. So this assignment violates just one constraint namely constraint C3, the constraint between X4 and X1.

Further backtrack leads us to X1=1,X2=2 which violates 1 constraints and thus any further expansion of this assignment cant lead to any better solution than the current best solution S. So we bound the search here and backtrack to the next level X1=2, and then proceed to X2=1. Here again the number of constraints violated is greater than or equal to the current N,namely 1. So we bound it here and face a similar condition at X1=2,X2=2.

This concludes the search process and we get the solution S=$\{X1 = 1, X2 = 1, X3 = 2, X4 = 2\}$ which violates just N = 1 constraints.

2.1.2 Backjumping

In the previous algorithm, the depth-first branch and bound method[1] was discussed. It was formulated as a variant of the naive backtracking algorithm for the maximal CSP. In backjumping proposed by Gashnig[2], the naive backtracking method for CSPs

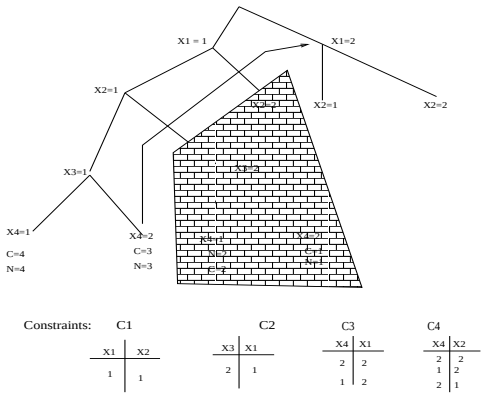

Figure 2: Backjumping with depth-first branch and bound fails to capture the maximal solution for the example in Figure 1. Shaded area is the area avoided by the backjumping algorithm.

was modified with a provision for some bookkeeping to achieve a significant reduction in the search space than the naive backtracking algorithm.

When backtracking occurs from a dead-end at a variable, the control passes to the chronologically previous variable. But it is possible that the variables responsible for the dead-end at that node are actually much higher up than the immediately previous variable. Let Xj be the deepest variable responsible for creation of the dead-end at a node Xk, and let Xi be the variable immediately chronologically prior to Xk. In naive backtracking the control passes on to Xi after Xk. But since Xj is the deepest variable responsible for the dead-end at Xk, any exploration of the nodes between Xj and Xk, will not prevent the dead-end at Xk. So if instead of jumping to Xi, if we jump directly to Xj, skipping across all variables between Xj and Xi, we still preserve the solution and save a lot of search space too. Thus backjumping results in a significant reduction in search space which overshadows the slight overhead involved in the bookkeeping of the deepest variable in conflict with the current node Xk.

But this technique cannot be directly translated to the max-CSP case, as was possible in the naive backtracking to depth-first branch and bound case. Consider the example in figure 2. This is the same as the example in figure 1. Here we follow depth first branch and bound in the same manner as explained in the previous section till we reach the node representing $\{X1 = 1, X2 = 1, X3 = 1, X4 = 2\}$. At this node we find that the node X1 itself is in conflict with both values of X4. So we can backjump from X4 to X1 directly instead of X3 or X2. So we can backjump to X1=2 directly. But in the process we avoid a significant portion of the search space, which is a considerable savings in the normal case. But from the figure it is also clear that the search space we are avoiding, actually contains the maximal solution,namely, $\{X1 = 1, X2 = 1, X3 = 12, X4 = 2\}$. So backjumping algorithm for CSP, when directly applied to the max-CSP, leads to incorrect results. The phenomenon occurs because of the fact that X3=1, was an inconsistent node, and in the avoided path, X3=2 was not an inconsistent node, and this lead to the possibility of a better solution along that path. The solution proposed in [1], is to keep track of the latest inconsistent node encountered in the current path, and backjump to the later of the two variables, the backjump point or the deepest inconsistent variable encountered. This ensures that the correctness of the algorithm is not compromised even though the savings in search space due to backjumping may be lost in some situations. Consider the situation above. At X4=2, we have to choose between the backjump point namely X1, and the last inconsistent point ,namely X3. Since X3 is deeper than X1, backjump occurs to X3 and we have no savings in space though we preserve the correctness of the algorithm.

3 Extension of the results to conflict-directed backjumping

In the previous section, the studies by Freuder and Wallace[1] in extending backtracking and backjumping algorithms to maximal constraint satisfaction problem was explained. It was shown that an extra variable to keep track of the deepest inconsistent variable at any time, is needed to decide the next backjump point, in case of backjumping. The backjump took place to the deeper of the two variables, the deepest variable in conflict with the current node or the deepest of the inconsistent nodes.

In contrast to backjumping, in conflict-directed backjumping, the backjump point is not just dependent upon the variable at the current level but also upon the variables at levels below. At each level of x_i an array conflict-set, keeps track of all the past variables in conflict with the current level. Every time an inconsistency is encountered between x_i and some past variable, the past variable is added to the conflict set of x_i. In the event of all values of x_i being exhausted, the algorithm backjumps to the deepest variable x_h in the conflict-set of x_i. In the process of this backjump, the variables in the conflict-set of x_i (excluding x_h) are added to the conflict-set of x_h, because none of these valuations can lead to a successful solution. Thus the bookkeeping in conflict-directed backjumping (CBJ) is more involved than that in plain BJ case.

We show that similar to BJ, CBJ algorithm cannot

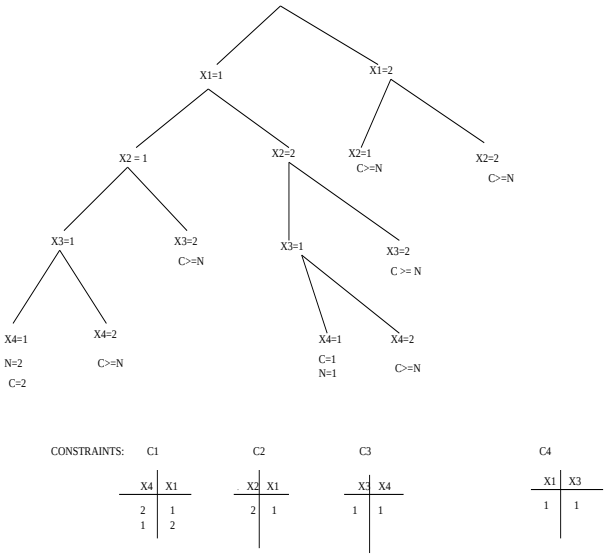

FIGURE 3: Depth first branch and bound for example 2 with constraints C1, C2, C3 and C4 above.

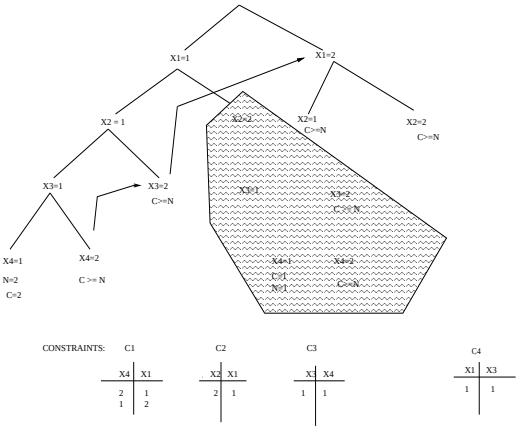

FIGURE 4: CBJ for example 2 with constraints C1, C2, C3 and C4 above. The shaded area is the area avoided by the CBJ and contains the critical solution.

be applied as it is to the depth-first branch and bound algorithm. Consider the example in Figure 3. In figure 3, the depth-first branch and bound traversal for the CSP is shown. The depth-first branch and bound algorithm traverses the search space and returns with the maximal solution $\{X1 = 1, X2 = 2, X3 = 1, X4 = 1\}$.

Now, consider the same example with conflict-directed backjumping. In CBJ, at X4=1 in the first pass at $\{X1 = 1, X2 = 1, X3 = 1, X4 = 1\}$ the conflict-set consists of the set $\{X1\}$. Next at $\{X1 = 1, X2 = 1, X3 = 1, X4 = 2\}$, the conflict-set now becomes $\{X1, X3\}$ for X4. So backtrack occurs to the deepest variable in the conflict-set namely $\{X3\}$. At the same time the conflict-set of X3 is reassigned $\{X1\}$, since that is the only remaining element in the conflict-set of X1 other than X3 itself. So the search proceeds along $\{X1 = 1, X2 = 1, X3 = 2\}$. But at this point we reach abound and now need to backtrack or backjump to the deepest variable in the conflict-set of X3, which is X1. Next search will proceed to X1=2 onwards. So in this process, we have jumped across a major portion of search space. But this search space contains the actual maximal solution found by DFBB. So in applying CBJ to the branch and bound algorithm, the correctness of the algorithm is lost.

The phenomenon occurring here is similar to that occurring in the plain backjumping case as shown in [1]. The anomaly occurs because of the presence of inconsistent node X2, between the point to which conflict-directed backjumping occurs and the current level. The inconsistent node in the present branch leaves the prospect of a better solution in an alternative branch at the inconsistent node (X2 here). Here the maximal solution $\{X1 = 1, X2 = 2, X3 = 1, X4 = 1\}$ occurs in one such alternative branch.

The solution suggested by [1], can be extended to the CBJ case, but with additional bookkeeping. The idea is to backjump to the deepest of the two variables, the backjump point or the deepest inconsistent node prior to the current node. In BJ case, a variable was kept to keep track of the deepest inconsistent node encountered.

But in CBJ case, we need to keep track of the information of the deepest inconsistent node prior to every level, not just for the present level because in backjumping information requires only conflict information corresponding to the present level while in CBJ, the conflict information of the present level is passed on to the upper level to which the backjumping occurs. So the information pertaining to the inconsistent nodes should be kept corresponding to each level. Let inconsistent[i] correspond to the deepest inconsistent node prior to the level i. This information is maintained and passed on in the following manner. Initially all are initialized to 0. Then as the partial solution is expanded on in the CBJ algorithm, all entries still remain zero, till an actual inconsistent node say Xj is encountered. Then as this partial solution is expanded to Xj+1, then inconsistent[Xj+1] is made to Xj. The same value Xj is passed on to all entries till the next such inconsistent node is encountered,say Xc. Even inconsistent[Xc] is made to Xj. For any further expansion of this value Xc will be used. This goes on till an actual backjump takes place. At a level k, let Xi be the deepest of the two variables inconsistent[k], and the deepest variable in conflict-set[k]. Now when backjumping takes place to Xi, two changes are done. Conflict-set[i] is updated to the union of conflict-set[i]

and conflict-set[k] minus the variable Xi. In addition the values of inconsistent[j] is made to inconsistent[i] for all j > i. Because now the value of Xi is changed to a new variable and it may not be an inconsistent node. This minor additional bookkeeping ensures that the correctness of the CBJ is ensured when applied to the depth first branch and bound case.

4 Theoretical Analysis of branch and bound algorithms for the maximal CSP

In this section we shall explore some theoretical results associated with the branch and bound algorithms for the maximal constraint satisfaction algorithm. The results are analogous to ones discussed for plain constraint satisfaction problems in [3]. Here we observe that since in maximal CSP, we cannot usually satisfy all the constraints at one time, we need to concentrate on the problem of satisfying a subset of constraints at a time.

The main idea of this section is to present a characterization of static conditions under which a particular backtrack algorithm visits a node in the search. In doing so, it is possible to sometimes obtain a rough idea of the behavior of the backtrack algorithm. A characterization in terms of dynamic conditions is not of practical use since the checking of conditions in such case itself will be a costly process. By dynamic characterization, we mean a characterization in terms of a variable which dynamically changes during the search process.

The main difficulty in extrapolating the results from [3], to maximal constraint satisfaction problem is that the mere presence of an inconsistency at a node does not stop the search at that point in branch and bound algorithms.

Again, in the depth-first branch and bound method as outlined in a previous section, N,the number of allowed constraint violations varies as we proceed along the way in the search process. Thus it is impossible to obtain any static condition dependent on N for measuring the effectiveness of any branch and bound algorithm. So we need to fix our value of N, the necessary bound on the number of constraints violated in the maximal CSP, in order to get static quantified results pertaining to the performance of the individual branch and bound algorithms. So from now on in this section we concentrate on the problem of maximal CSP, with a predetermined N, for the conditions to explicable and applicable. In other words, in the modified formulation of the maximal CSP, we would like to find the best possible solution to the CSP subject to the condition that no more than N (predetermined or fixed) constraints are violated.

Based on a measure of N, we can now elucidate sufficient and necessary conditions for the different variants of the branch and bound methods to work for this version of the maximal constraint satisfaction problem.

4.1 Depth first Branch and Bound

In plain depth first branch and bound (DFBB) method, the search proceeds by assigning a value for a variable and then expanding the variable sets till either all variables are exhausted or we reach a point where the limit of N constraint violations is reached.

Because of the presence of the predetermined bound N on the number of inconsistencies, we can elucidate sufficient condition for the DFBB algorithm to visit a node.

Theorem 1 *If DFBB visits a node, its parent is a node with less than N inconsistencies.*

Proof 1 *The proof is very straightforward. If the parent of the node had greater than N inconsistencies, then by virtue of the algorithm the current node will never be reached and bounding will occur at the parent node itself.*

Unfortunately, it is not possible to give a static necessary characterization of the algorithm. Because during the course of the algorithm, if on the way, a solution is found with less than N inconsistencies, then N can be updated in such a case and this process can be dynamic. But if we know beforehand that any solution has a minimum of Nmin inconsistencies, then it would be possible to give a necessary condition similar as above:

Theorem 2 *If a node is such that its parent is node with $<$ Nmin inconsistencies, then the node is visited by DFBB.*

Proof 2 *The proof for the this theorem is straightforward.*

The above results show that the condition analogous to the consistency of a node in backtracking case[3], is the condition that the node have less than N inconsistencies. This is explained by the fact that in DFBB, a failure is triggered by a partial solution with greater than N inconsistencies while in backtracking the failure is triggered by an inconsistency.

4.2 Backjumping

Before we give a sufficient condition characterizing the branch and bound algorithm with backjumping, we state the following lemma.

Lemma 1 *Any node visited by BJBB to a_j after a_i such that $(i > j)$, is such that $(a1, a2, ..., a_j)$ along with any of the value of x_i will have $>= N$ inconsistencies.*

Proof 3 *The proof follows from the fact the fact that backjump occurs from the node x_i to x_j. Irrespective of the fact whether x_j represents the deepest inconsistent node before x_i or the deepest variable in conflict with x_i, no variable between x_j and x_j is inconsistent or has conflict with x_j. So any of these variable instantiations do not add any further inconsistencies caused by $a_1, ..., a_j$ with a_i. And we know that backtracking occurs at a_i, so it follows the $a_1, a_2, ..., a_j$ with a_i have $>= N$ inconsistencies.*

Based on the above condition the sufficient condition for the BJBB algorithm can be written as follows:

Theorem 3 *If BJBB visits a node, its parent node has less than N inconsistencies when combined with any other variable.*

The theorem directly follows from the lemma above.

Again, it is difficult to elucidate a necessary condition for the BJBB algorithm, because of the same problem as in case DFBB. We change the value of N whenever a partial solution having lesser number of inconsistencies is encountered. Once again if we are ensured that we have a measure of the best solution having Nmin inconsistencies, the necessary condition for the BJBB algorithm then can be stated as follows:

Theorem 4 *If a node is such that its parent has less than Nmin inconsistencies when combined with any other variable, then BJBB visits the node.*

5 Contributions and future work

This document studies the problem of maximal constraint satisfaction in detail with stress on the intelligent retrospective backtracking techniques. It extends the observations made in [1] to conflict-directed backjumping. The other contribution of this treatise is a detailed theoretical analysis of the intelligent backtrack algorithms in the domain of overconstrained problems, along the line pursued by [3] for CSPs.

This work shall be extended to include the incorporation of learning techniques in the backtrack techniques for maximal CSP. The learning algorithms have been studied as a major source of speedup for CSPs.

References

[1] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. In *Proc. Workshop on Over Constrained Systems,CP 95*, pages 63–109, 1995.

[2] J. Gaschnig. A general backtrack algorithm that eliminates most redundant checks. In *Proc. IJCAI*, page 457, 1977.

[3] Gregorz Kondrak and P. van Beek. A theoretical evaluation of selected backtracking algorithms. In *Proc. Fourteenth IJCAI*, pages 541–546, Montreal, Canada, August 1995.

[4] A. Mackworth. Constraint Satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 205–211. John Wiley and Sons, 1987.