Satisfaction Guaranteed *

Eugene C. Freuder and **Tom Carchrae** and **J. Christopher Beck**
Cork Constraint Computation Centre
University College Cork, Ireland
{e.freuder, t.carchrae, c.beck}@4c.ucc.ie

Abstract

A constraint satisfaction problem (CSP) model can be preprocessed to ensure that any choices made will lead to solutions, without the need to backtrack. This can be especially useful in an interactive context in which different users access the model to make choices on-line, e.g. in e-commerce configuration. The conventional machinery for ensuring backtrack-free search, however, adds additional constraints, which may require an impractical amount of space. A new approach is presented here that achieves a backtrack-free representation by removing values. This may limit the choice of solutions, but we are guaranteed not to eliminate them all. Experimental experience suggests that the trade-off for users between processing effort and solution loss may be worthwhile.

1 Introduction

A Constraint Satisfaction Problem (CSP) involves choosing values for variables that satisfy restrictions (constraints) on allowed value combinations. The basic method for solving such problems is backtrack search, which involves choosing values for variables in turn, backing up to make alternative choices when there is no way to proceed without violating a constraint.

Backing up is always costly, and in some situations may be impractical or even impossible. A customer configuring a purchase at an e-commerce website may abandon the purchase when forced to reconsider his choices. An on-line production process that chooses to mix in a chemical cannot "unmix" it. This paper is most directly motivated by work by NASA scientists seeking assurances that autonomous spacecrafts can commit to scheduling decisions in real time [8]. We would like to preprocess problems to obtain a "backtrack-free" representation that permits subsequent repeated use, where users can make different choices at "execution time", secure in the knowledge that none can lead to failure.

Early work on CSPs guaranteed backtrack-free search for tree-structured problems [4]. This was extended to general CSPs through k-trees [5] and adaptive consistency [3]; but these methods, of course, have exponential worst-case complexity. We would argue that "offline" preprocessing time is not a critical factor when we envision repeated "real time" reuse of the backtrack-free representation, while users make alternative choices (though these methods were not originally proposed in this context). However, these methods also have exponential worst-case *space* complexity, which may indeed make them impractical.

In this paper we propose preprocessing methods that can achieve a backtrack-free problem representation (BFR) without incurring any space penalty. For many problems the preprocessing time seems acceptable; in any event it is no worse than for conventional methods. Of course, there is a trade-off: some, though not all, of the solutions may be lost. This restricts the range of choice during subsequent use of the BFR, but experimental evidence suggests that in many cases this trade-off may lie within acceptable bounds.

Consider a simple example: a coloring problem on variables X, Y and Z. We wish to choose either red or blue as a color for each variable. We are constrained in that Z must be different in color from both X and Y. Suppose our user wants to choose colors for X, Y and Z in that order. There is a danger that he may choose red for X, blue for Y, and then be left with no choice for Z. The conventional way of "fixing" this would be to add a new constraint between X and Y specifying that the combination red for X and blue for Y is prohibited. The new constraint would prevent the user from getting in trouble in this way. However, the constraint requires additional space. This is a simple example of adaptive consistency; in general we may have to add constraints involving as many as $n-1$ variables for an n-variable problem.

Our basic insight here is so simple that it may at first appear simpleminded; but we are reassured by the observation that this is often the case with perfectly good ideas. We will "fix" the problem by removing the choice of red for X. The user cannot get in trouble by choosing red for X because the choice will not be there. Of course, this also removes a solution (red X, red Y, blue Z), but another remains (blue X, blue Y, red Z). If we also remove red as a value for Y we are left with a backtrack-free representation. (Of course, this also leaves us with no choices at all for this problem, just a single solution, but in general we will only restrict, not eliminate, choice.)

In Section 2, we present a basic algorithm, BFRB, for pre-

*This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075.

Algorithm 1: BFRB - computes a BFR

BFRB(n):
Obtains a BFR for P_n (maintained as a global variable)

1 **if** *domain of* V_n *is empty* **then**
2 report Failure
3 **if** $n = 1$ **then**
4 report Success
5 **foreach** *solution* S *to the parent subproblem that does not extend to* V_n **do**
6 Choose a value v in S and remove it from the domain of its variable.
7 **recursively** seek a BFR for P_{n-1}:
8 If successful, report Success.
9 If not, make one different choice of a value to remove, and recurse again.
10 When there are no more different choices to make, report Failure.

processing a problem to achieve a backtrack-free representation by removing values, and prove that it will find such a representation for any soluble binary CSP. We also suggest refinements, heuristics and alternatives. In Section 3 we describe algorithm instantiations for obtaining a BFR, and study their performance. We measure solution loss and preprocessing effort as well as the effort saved during subsequent search. In particular, we foresee the preprocessing as being of use in an interactive CSP setting, where human users make the value choices, e.g. for e-commerce product configuration, and consider the savings the users gain, in avoiding tedious undoing of choices or long processing delays, in return for restricting the choices available. In Section 4, we discuss our experimental result. In Section 5 we describe some interesting extensions to explore. In Section 6, we place this work in the context of several broad themes in constraint computation.

2 Algorithm, Alternatives, and Analysis

We describe a basic algorithm for obtaining a BFR by deleting values, prove it correct and examine its complexity. Given a problem P and a variable search order V_1 to V_n, we will refer to the subproblem induced by first k variables as P_k. A variable V_i is a parent of V_k if it shares a constraint and $i < k$. We call the subproblem induced by the parents of V_k the *parent subproblem* of V_k. P_n will be a backtrack-free representation if we can choose values for V_1 to V_n without backtracking. BFRB operates on a problem and produces a backtrack-free representation of the problem, if it is solvable, else reports failure. We will refer to the algorithm's removal of solutions to the parent subproblem of V_k that do not extend to V_k as *processing* of V_k.

The BFRB algorithm is quite straightforward. It works its way upwards through a variable ordering, ensuring that no trouble will be encountered in a search on the way back down, as does adaptive consistency; but here difficulties are avoided by removing values rather than adding (or refining) constraints. (Of course, removing a value can be viewed as adding/refining a unary constraint.)

However, correctness is not as obvious as it might first appear. It is clear that a BFR to a soluble problem must exist; any individual solution provides an existence proof: simply restrict each variable domain to the value in the solution. However, we might worry that BFRB might not notice if the problem is insoluble, or in removing values it might in fact remove all solutions, without noticing it.

Theorem 1 *If P is soluble, BFRB will find a backtrack-free representation.*

Proof: Proof by induction.

Inductive step: If we have a solution s to P_{k-1} we can extend it to a solution to P_k without backtracking. Solution s restricted to the parents of V_k is a solution to the parent subproblem of V_k. There is a value, b, for V_k consistent with this solution, or else it would have been eliminated by BFRB. Adding b to s gives us a solution to P_k, since we only need worry about the consistency of b with the parents of V_k.

Base step: P_1 is soluble, i.e. the domain of V_1 is not empty after BFRB. Since P is soluble, let s be one solution, with s_1 as the value for V_1. We will show that if it does not succeed otherwise, BFRB will succeed by providing a representation that includes s_1 in the domain of V_1. We will do this by demonstrating, again by induction, that in removing a solution to a subproblem, s_p, BFRB will always have a choice that does not involve a value of s. Suppose BFRB has proceeded up to V_k without deleting any value in s. It is processing V_k and a solution s_p to the parent subproblem does not extend to V_k. If all the values in s_p are in s, then there is a value in V_k that is consistent with them, namely the value for V_k in s. So one of the values in s_p must not be in s, and BFRB can choose at some point to remove it. (The base step for V_n is trivial.) Now since BFRB tries, if necessary, all choices for removing values, BFRB will choose eventually, if necessary, not to remove any value in s, including s_1. □

Theorem 2 *If P is insoluble, BFRB will report failure.*

Proof: Proof by induction.

$P_n = P$ is given insoluble. We will show that if P_k is insoluble, then after BFRB processes V_k, P_{k-1} is insoluble. Thus eventually BFRB will always backtrack when P_1 becomes insoluble (the domain of V_1 is empty) if not before, and BFRB will eventually run out of choices to try, and report failure.

Suppose P_k is insoluble. We will show that P_{k-1} is insoluble in a proof by contradiction. Suppose s is a solution of P_{k-1}. Then s restricted to the parents of V_k, s_p, is a solution of the parent subproblem of V_k, which is a subproblem of P_{k-1}. There is a value b of V_k consistent with s_p, for otherwise s_p would have been eliminated during processing of V_k. But if b is consistent with s_p, s plus b is a solution to V_k. Contradiction. □

The space complexity of BFRB is polynomial in the number of variables and values, as we are only required to represent the domains of each variable. The worst-case time complexity is, of course, exponential in the number of variables,

n. However, as we will see in the next section, by employing a "seed solution", we can recurse without fear of failure, in which case the complexity can easily be seen to be exponential in $(p + 1)$, where p is the size of the largest parent subproblem. Of course, $p + 1$ may still equal n in the worst case; but when this is not so, we have a tighter bound on the complexity.

3 Algorithm Instantiations

We can envision many potential improvements on the basic BFRB algorithm. In this paper, we will investigate some basic issues, describing more involved improvements in Section 5.

Using Seed Solutions. In the basic BFRB algorithm, it is possible to remove all the values in the domain of a variable, requiring the need to "backtrack" through the pruning choices for completeness. We can avoid the need to backtrack by finding an initial solution for the original CSP and using it as a seed. We specify that we cannot remove any values in that seed solution while searching for a BFR. There is a computational cost to obtaining the seed, and "protecting" it reduces the flexibility we have in choosing which values to remove; but we avoid thrashing when finding a BFR.

We performed preliminary experiments on the use of a seed solution to find a BFR versus finding one from scratch. Our results indicated that not only was using a seed significantly faster, it also tended to produce a BFR which preserves more solutions. Given the strength of these results, we only report here on finding a BFR using a seed.

Enforcing Consistency. We expect a second sets of improvements to arise from applying consistency algorithms while searching for a BFR. We could establish arc-consistency (AC) before starting the search for a BFR and/or every time we prune a value. Since non-arc-consistent values may lead to dead-ends, establishing AC will reduce the number of times that we must make a heuristic decision for pruning.

We experimented with two uses of arc consistency: establishing AC once in a preprocessing step and establishing AC whenever a value is pruned (line 6 of BFRB). The latter variation proved to incur less computational effort as measured in the number of constraint checks to find a BFR and resulted in BFRs which retained more solutions. In our experimental results, therefore, we only show results where AC is established whenever a value is pruned.

Using Pruning Heuristics. The selection of the value to be pruned to remove a dead-end may benefit from heuristics. It is unclear how the standard CSP heuristics (e.g., based on domain size and degree) will transfer to BFRB, but it is reasonable to expect that there will be some impact in preferring to prune different values. Two heuristics (line 6 of BFRB), together with their corresponding anti-heuristic, are tested in this paper based on the following characteristics:

Domain Size. As in heuristics for finding a solution to a CSP, we expect that the size of the domain of the variable whose value we remove will have an impact on the quality of the BFR produced. A value in the minimum domain is likely to participate in a larger proportion of both the remaining dead-ends and solutions than a value in a larger domain. Therefore, it is unclear whether the minimum domain heuristic *MinDom* or the maximum domain heuristic *MaxDom* should be expected to perform better. *Degree.* The degree of variables in the constraint graph is also a component of existing CSP heuristics. Removing a value from a variable of high degree will have an impact on more of the other variables in the problem than doing so from a variable of low degree. This impact may be to add or remove dead-ends. We cannot, a priori, predict whether the maximum degree heuristic, *MaxDeg*, or the minimum degree heuristic, *MinDeg*, will produce better BFRs.

To provide a baseline for comparison we also experiment with selecting the value to be removed randomly, in the *Random* "heuristic". For all heuristics, ties are broken lexicographically.

Since we are using a seed solution, the pruning heuristic is restricted by the fact that a value in the seed solution is never removed. If the heuristically preferred value occurs in the seed solution, the next most preferred value is pruned. We are guaranteed that at least one parent will have a value that is not part of the seed solution or else we would not have found a dead-end.

Probing to Find Good BFRs. Since we want BFRs to retain as many solutions as possible, it is useful to model the finding of BFRs as an optimization problem rather than as a satisfaction problem. We envision a number of ways to do this, for example, by performing a branch-and-bound to find the BFR that retains the maximum number of solutions. In this paper, we will take advantage of the fact that we generate BFRs starting with a seed solution to introduce a simple probing algorithm. For a given seed solution, a BFR is generated and the number of solutions retained are counted. The search for a BFR is then restarted from a random seed solution. This process is continued until no improving BFR could be found in 1000 such iterations. This technique is, of course, incomplete, however, we are interested to investigate how much improvement we can achieve over the satisfaction algorithms.

4 Experiments

The purpose of this section is to evaluate the basic idea of finding a BFR through pruning values and to perform preliminary investigations of some of the algorithm variations noted above. Our basic interests are to look at the on-line processing effort that will be saved by using a BFR rather than the original problem representation, the solutions lost by removing values, and the effort required to find a BFR.

To evaluate our algorithm instantiations, we generated problems with 15 variables with 10 values in each domain. One problem set contains sparse instances (density = 0.3) while the other contains dense problems (density = 0.7). For each of these sets, we identified a range of tightness values that allowed us to span the phase transition region from the easy soluble problems, across the hardness ridge, to the (relatively) easy insoluble problems. Since a BFR is only well-defined in a soluble problem, for each combination of density and tightness we generated 100 soluble problem instances by filtering out the insoluble problems. Table 1 presents the

range of tightness values used in each of the problem sets.

We solved the problems with MAC [9] using lexicographic search order, which is effectively a random search order. In the interactive settings we envision the search order cannot necessarily be chosen for efficient search: the chemicals may need to be mixed in a specific order, the rockets may need to be fired in a specific order, choice may be based on preference or cost. Table 1 presents the mean and median number of backtracks in finding an initial solution. This initial solution is then used as a seed in the search for a BFR, which will allow users to avoid any backtracking when interactively seeking solutions. Even if we consider problems that do permit a more efficient search order to be used, even moderately difficult problems will still require far more backtracks than we can expect a human user to tolerate in an interactive setting.

Sparse (Density = 0.3)			Dense (Density = 0.7)		
Tightness	mean BTs	median BTs	Tightness	mean BTs	median BTs
0.5	9.64	3.5	0.30	7	4
0.51	6.24	3	0.31	12.49	7
0.52	4.89	3	0.32	19.05	15
0.53	6.93	4	0.33	26.59	20
0.54	5.52	4	0.34	31.81	23
0.55	6.21	4	0.35	24.59	19
0.56	4.28	3	0.36	24.78	20
0.57	2.93	2	0.37	17.19	14
0.58	2.98	2	0.38	13.46	11.5
0.59	3.05	3	0.39	8.90	8
0.6	2.16	1	–		

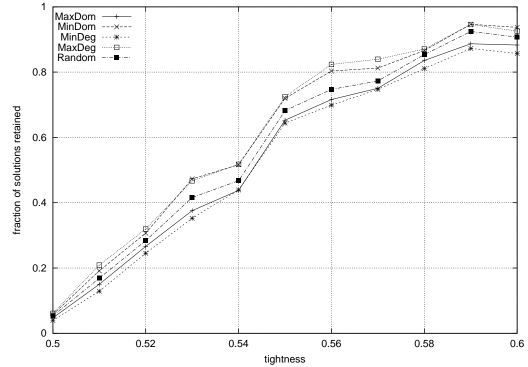
Table 1: The tightness of each problem sub-set for the sparse and dense problems and the mean and median number of backtracks required for MAC with lexicographic variable and value ordering to find a first solution.

4.1 Satisfaction Results

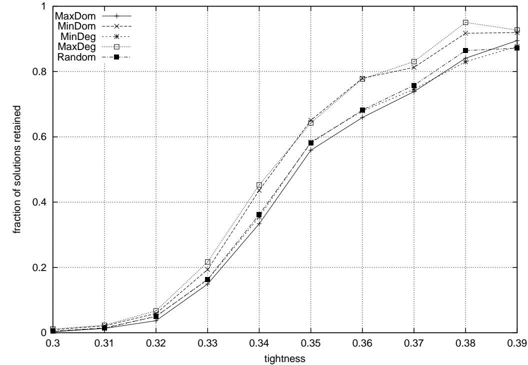
Using the first solution found, we then found a BFR with each of the one-seed algorithm instantiations. Since we are using a seed, we already have a BFR. However, for the purposes of comparison, we consider that a BFR has been found when algorithm BFRB terminates. We then count the number of solutions of the original problem, that are still solutions to the BFR: that is, the number of solutions that have value assignments that are in the domains of the BFR representation.

Figure 1(a) presents the relative number of solutions retained for each of the algorithm instantiations on the sparse problem set. On the tighter problems, the best heuristic is able to find BFRs which retains more than 80% of the solutions in the original problem. On the loose problems, which have many solutions, a small percentage are retained however even for tightness of 0.52, the best heuristic finds BFRs retaining more than 30% of the solutions. From the perspective of the heuristics, MaxDeg and MinDom out-perform MinDeg and MaxDom respectively. In general, the range from the best heuristic to the worse appears to be about 10% of the solutions in the original problem.

Figure 1(b) presents the relative number of solutions retained on the dense problem set. In general, it appears that the fraction of solutions retained is lower on the dense problem set than on the sparse one, however, for the tightest half of the problem set, on average at least 60% and up to 95%



(a) Sparse Problems



(b) Dense Problems

Figure 1: The mean number of solutions retained in the BFR representations of the sparse problem set relative to the number of solutions for the original problem.

of the solutions are retained with a single BFR. The relative performance of the heuristics is similar to that on the sparse problem set with MaxDeg and MinDom dominating.

The effort to find a BFR is assessed in Figure 2, where the number of constraint checks for each algorithm instantiation are displayed. The largest difference between heuristics is on the loose problems where MaxDom incurs over twice as many constraint checks as MaxDeg. Interestingly, the heuristics that retain more solutions also spend less effort in finding a BFR. The data is noisier for the sparse set, however, the relative performance of the heuristics is similar with the number of constraint checks being an order of magnitude lower. This is reasonable as the size of each subproblem is smaller.

4.2 Optimization Results

While successful in retaining a large percentage of the solutions, especially, on the tighter problems, the satisfaction algorithms did not attempt to find a good BFR: the first satisfying BFR was reported. Figure 3 demonstrates that substantially better performance can be achieved by using the heuristics within a simple probing algorithm. For the loosest problems, where the satisfaction algorithms performed worse, the optimization algorithms are able to find BFRs that on average retain an order of magnitude more solutions than the satisfaction algorithms. On the tighter problems, the advantage

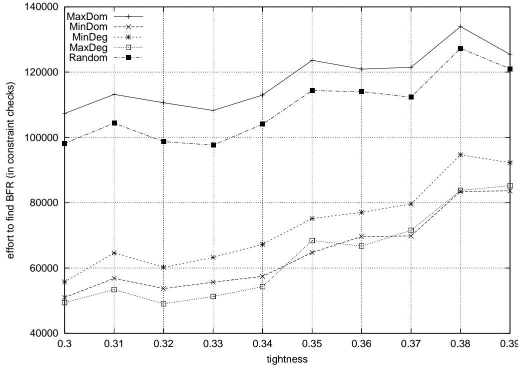


Figure 2: **Dense Problems** The mean number of consistency checks required to find a BFR starting from a seed solution.

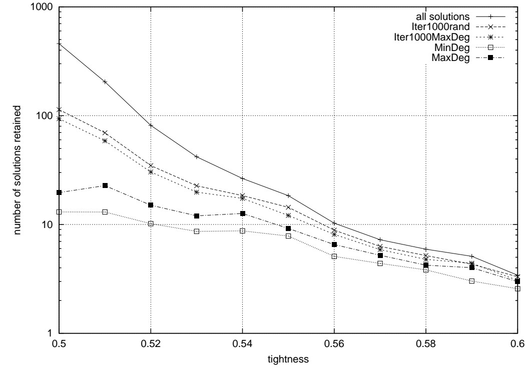
for the optimization algorithms continues to be apparent as almost all solutions are retained for the tightest problems.

5 Future Work

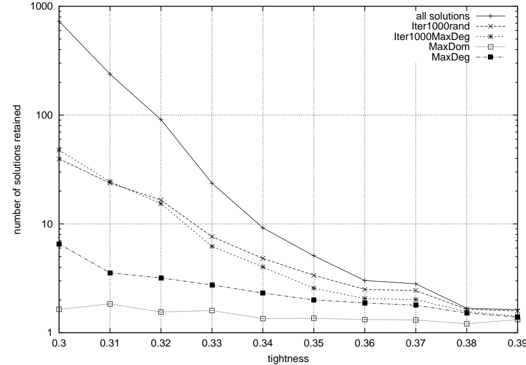
This paper represents initial experiments with BFRs. Clearly there are a number of extensions to be explored. More research is needed in the area of heuristics as we have demonstrated that much better BFRs exist than are discovered with the heuristics we used. More generally, though, by changing our assumptions about the on-line processing, we can also expand the set of techniques applied in finding the BFR. For example, so far we have assumed that the on-line algorithm is simple backtracking with no consistency enforcement. If we make assumptions that we will use forward checking or MAC on-line, we can remove fewer dead-ends off-line and lose less solutions. Dead-ends that will not be encountered by the on-line algorithm do not have to be removed off-line. This means, in fact, that a backtrack-free representation is backtrack-free with respect to the on-line algorithm: a BFR built for MAC will not be a BFR for backtracking (though the converse is true). It is a relatively easy transformation of BFRB to ensure that only those dead-ends that exist for a specific on-line algorithm will be pruned.

A different, though complementary, direction for further research is in storing more than one BFR for a problem instance. The tighter a problem is, the smaller the number of solutions can be represented in a single BFR. However, if we maintain a limited set of BFRs, we could still achieve polynomial space complexity, backtrack-free on-line search, and significantly broader solution coverage. For example, if we stored $n \times d$ BFRs, we could ensure that every globally consistent value of every variable was represented in at least one BFR.

Finally, in a real application some BFRs will be more preferred than others, perhaps, not simply based on the number of solutions retained. In a configuration application, it is likely that the vendor will have some guidance as to the attributes he/she would like to have in the most common solutions. This suggests that it would be very useful to be able to reason about soft constraints, preferences, or other optimization criteria while building a BFR or, more generally, to



(a) Sparse Problems



(b) Dense Problems

Figure 3: Using a log-scale we plot the mean number of solutions in the original problem, the mean number of solutions retained by the best and worst algorithm instantiations, and the mean number of solutions retained by two variations of the probing technique.

be able to find a BFR for constraint optimization problems.

6 Context

The work on BFRs presents a perspective on a number of fundamental dichotomies in constraint processing.

BFR vs. Adaptive Consistency. BFRB can be viewed as one extreme on a spectrum that has adaptive consistency [3] at the other end. Our BFR algorithm could be termed a 1-BFR algorithm, building a BFR by adding or altering unary constraints (removing values). A 2-BFR algorithm would add or alter binary constraints when possible, rather than unary ones, and could thus achieve a BFR while removing fewer solutions. In general, we have k-BFR, where k is the “induced width” in the case of adaptive consistency, which in the worst case is $n - 1$ for an n variable problem. As k increases, we suffer less solution loss, but incur a greater space cost. Further work is needed to explore the trade-offs here.

Inference vs. Search. As in many aspects of constraint computation, the axis that runs from inference to search is relevant for BFRs. The basic BFR algorithm allows us to perform pure search online without fear of failure. BFRs for online algorithms that use some level of inference require more online computation while still ensuring no backtracks and

preserving more solutions. It would be interesting to study the characteristics of BFRs as we increase the level of online consistency processing we are willing to do.

Implicit vs. Explicit Solutions. BFR models can be viewed along a spectrum of implicit versus explicit solution representation, where the original problem lies at one end, and the set of explicit solutions at the other. The work on "bundling" solutions provides compact representations of sets of solutions. Hubbe & Freuder [7] represent sets of solutions as Cartesian products, each one of which might be regarded as an extreme form of backtrack-free representation. If we restrict the variable domains to one of these Cartesian products, every combination of choices is a solution. All the solutions can be represented as a union of these Cartesian products, which suggests that we might represent all solutions by a set of distinct BFRs. As we move toward explicit representation the preprocessing cost rises. Usually the space cost does as well, but 1-BFR representations are an exception that lets us "have our cake and eat it too".

Removing values vs. Search. Removing values is related in spirit to work on domain filtering consistencies [2] though these do not lose solutions. Another spectrum in which BFRs play a part therefore is based on the number of values removed. We could envision BFRB variations that remove fewer values, allowing more solutions, but also accepting some backtracking. Freuder & Hubbe [6] remove solutions in another manner, though not for preprocessing, but simply in attempting to search more efficiently. Of course, a large body of work on symmetry and interchangeability does this.

Offline vs. Online Effort. BFRs lie at one end of an axis that increasingly incorporates offline preprocessing or precompilation to avoid online execution effort. These issues are especially relevant to interactive constraint satisfaction, where human choices alternate with computer inference, and the same problem representation may be accessed repeatedly by different users seeking different solutions. They may also prove increasingly relevant as decision making fragments among software agents and web services. Amilhastre et al. [1] have recently explored interactive constraint solving for configuration, compiling the CSP offline into an automaton representing the set of solutions.

"Customer-centric" vs. "Vendor-centric" Preferences. As constraints are increasingly applied to online applications, the preferences of the different participants in a transaction will come to the fore. It will be important to bring soft constraints, preferences and priorities, to bear on BFR construction to address the axis that lies between "customer-centric" and "vendor-centric" processing. For example, a customer may tell us, or we may learn from experience with the customer, that specific choices are more important to retain. Alternatively, a vendor might prefer to retain an overstocked option, or to remove a less profitable one.

7 Conclusion

We have presented an approach to obtaining a backtrack-free CSP representation (BFR) that does not require additional space. We investigated a number of variations on the basic algorithm for finding BFRs including the use of seed solutions, arc-consistency, and a variety of pruning heuristics. We have evaluated experimentally the cost of obtaining a BFR, the solution loss, and the execution time savings, for different problem parameters.

Overall, our results indicate that a significant proportion of the solutions to the original problem can be retained especially when an optimization algorithm that specifically searches for such "good" BFRs is used. We expect that such performance can be improved, particularly with the increased use of consistency algorithms, the maintenance of multiple BFRs, and the use of more sophisticated optimization techniques.

Finally, we noted that the BFR concept provides an interesting perspective on a number of theoretical and practical dichotomies within the field of of constraint programming. Given these dichotomies and the potential for BFRs established in this paper, we feel that we have opened up a rich vein of research to explore.

References

[1] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs – application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.

[2] R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, May 2001.

[3] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987.

[4] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of ACM*, 29(1):24–32, 1982.

[5] E.C. Freuder. Complexity of k-tree-structured constraint-satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 4–9, 1990.

[6] E.C. Freuder and P.D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 254–261, 1993.

[7] P.D. Hubbe and E.C. Freuder. An efficient cross-product representation of the constraint satisfaction problem search space. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 421–427, 1992.

[8] N. Muscettola, P. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Principles of Knowledge Representation and Reasoning*, pages 444–452, 1998.

[9] D. Sabin and E.C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 125–129, 1994.