

A Totally Unimodular Description of the Consistent Value Polytope for Binary Constraint Programming

Ionuț D. Aron, Daniel H. Leventhal, Meinolf Sellmann

Brown University, Department of Computer Science
115 Waterman Street, Providence, RI 02912, U.S.A.
{ia,dleventh,sello}@cs.brown.edu

Abstract. We present a theoretical study on the idea of using mathematical programming relaxations for filtering binary constraint satisfaction problems. We introduce the consistent value polytope and give a linear programming description that is provably tighter than a recently studied formulation. We then provide an experimental study that shows that, despite the theoretical progress, in practice filtering based on mathematical programming relaxations continues to perform worse than standard arc-consistency algorithms for binary constraint satisfaction problems.

Keywords: *cost-based filtering, hybrid methods, mathematical programming.*

1 Introduction

As a result of the growing interaction between the mathematical programming and constraint programming communities, it has now become standard to use mathematical programming tools to derive information useful both for domain filtering and for guiding the search. On real-world constraint satisfaction problems (CSPs), and especially optimization problems, hybrid methods have been shown to outperform pure solution approaches. As a result of a decade long research, a rich tool-box for hybridization is now available: from the idea of optimization constraints [7, 14, 17] and associated notions of relaxed or approximated consistency [5, 19], reduced-cost filtering [16], to sophisticated problem-dependent techniques based on Bender’s decomposition [9], Lagrangian decomposition [6, 18, 20, 21], or column generation [4, 11]. Also, specialized hybrid approaches have been developed for special problems like computing orthogonal Latin squares [2] or to solve the social golfer problem [22].

Despite these successes, in the past hybridization on binary constraint satisfaction problems (BCSPs) has been nothing less than disappointing. Many approaches that looked very promising on paper have failed to give real benefits. While this is common knowledge in the research community and has led to the common belief that mathematical programming techniques only pay off when a problem contains constraints that contain large numbers of variables where constraint programming (CP) propagation is weak, we are not aware of any paper that would state such a negative result.

Consequently, we frequently see that, despite prior experience that developing hybrid methods for BCSPs is not a promising research avenue, the undoubtedly tempting idea lures researchers into developing new hybrid filtering approaches for BCSPs.

A recent approach regarding hybrid filtering for BCSPs is presented in [12]. The authors of that paper suggest to use a relaxation of an equivalent integer programming (IP) formulation of a given BCSP for domain filtering. Two ideas were novel in that contribution: first, the idea to use a Lagrangian relaxation instead of the commonly used linear relaxations for filtering. And second, to use a formulation that specifically targets individual assignments.

We were intrigued by those two ideas and decided to investigate them further. We address two questions: First, can the Lagrangian relaxation suggested in [12] yield to more effective filtering than standard linear programming (LP) relaxations? And second, does it pay off to focus on individual assignments for filtering in a tree search where what matters is the trade-off between filtering effectiveness and filtering time?

In order to answer those two questions, we start out in Section 2 by discussing different models for BCSPs and how they can be translated into integer programs. Based on those models, in Section 3, we develop an LP relaxation that is *provably tighter* than the Lagrangian relaxation developed in [12]. While offering the prospect of more effective filtering, that LP relaxation can also be computed much faster than Lagrangian relaxations when using standard LP software like Cplex.

In Section 4, we then present numerical results on various CSP and BCSP benchmark classes. The experiments show that, once again, mathematical programming techniques are inferior to standard arc-consistency on feasibility problems.

2 CSP and IP Models

2.1 Positive and Negative Representations of BCSPs

A binary constraint satisfaction problem (BCSP) consists of a finite set of *variables* $\mathcal{V} = \{V_1, \dots, V_n\}$, a finite *domain* $D_i = \{v_1^i, v_2^i, \dots, v_{l_i}^i\}$ for each variable V_i , and a finite collection of *constraints* $\mathcal{C} = \{C_1, \dots, C_m\}$. Each constraint C is a constraint over two variables $\text{Vars}(C) \subseteq \mathcal{V}$. Every constraint C can be viewed as a subset of the Cartesian product of the domains of the variables in $\text{Vars}(C)$ (i.e. the set of tuples that *satisfy* the constraint). Alternatively, C could also be viewed as the *complement* of this product (i.e. the set of tuples that *do not satisfy* the constraint, which are commonly referred to as *no-goods*). As we will see later, although equivalent, these two views of constraints lead to very different linear models.

Let $y_{iu} \in \{\text{true}, \text{false}\}$ represent the truth value of assignment $V_i = u$ (i.e. $y_{iu} = \text{true}$ iff $V_i = u$). The two representations of C_{ij} , as described above, become:

1. Positive Representation: Tuples that satisfy C_{ij} .

$$(P_{CSP}) C_{ij} ::= \mathcal{R}_{ij} = \{(u, v) \in D_i \times D_j : (u, v) \text{ satisfies } C_{ij}\}$$

For any value $u \in D_i$, the set of tuples $\{(u, v) : (u, v) \in \mathcal{R}_{ij}\}$ can be seen as the logical implication:

$$y_{iu} \rightarrow \bigvee_{v:(u,v) \in \mathcal{R}_{ij}} y_{jv} \quad (1)$$

This states that once we have assigned value u to variable V_i , we must also assign (at least) one of the values v to variable V_j . For this reason, we will call this representation the *positive representation of BCSPs*.

2. Negative Representation: Tuples that violate C_{ij} .

$$(D_{CSP}) C_{ij} ::= \overline{\mathcal{R}_{ij}} = \{(u, v) \in D_i \times D_j : (u, v) \text{ violates } C_{ij}\}$$

In this case, for any value $u \in D_i$, the set of tuples $\{(u, v) : (u, v) \in \overline{\mathcal{R}_{ij}}\}$ can be seen as the logical implication:

$$y_{iu} \rightarrow \bigwedge_{v:(u,v) \in \overline{\mathcal{R}_{ij}}} \neg y_{jv} \quad (2)$$

This states that once we have assigned value u to variable V_i , we cannot assign to variable V_j any of the values v . We therefore refer to this representation as the *negative representation of BCSPs*.

Note that, when written as logical implications, there is nothing in the positive representation that prevents us from assigning multiple values to a variable (i.e. $y_{iu} = y_{iv} = true$ for $u \neq v$), just as there is nothing in the negative representation that says that we must assign values to variables (i.e. $y_{iu} = true$ for some $u \in D_i$). However, once we enforce the implicit constraints that each variable V_i must take one and only one value $u \in D_i$, it is not hard to see that:

Lemma 1. *In a BCSP, positive and negative constraint representations are equivalent.*

Proof. Let $s(P_{CSP}) = (y_{iu} \mid 1 \leq i \leq n, u \in D_i)$ denote a solution of the positive BCSP. If $y_{iu} = true$ in $s(P_{CSP})$, then by (1) for any j there exists a value v such that $(u, v) \in \mathcal{R}_{ij}$ and $y_{jv} = true$. Since V_j can only take one value, it means that for any other value $v_k \in D_j$, $y_{jv_k} = false$. In particular, for all v_k such that $(u, v_k) \notin \mathcal{R}_{ij}$, we have $y_{jv_k} = false$, which means that (2) also holds. If on the other hand $y_{iu} = false$ in $s(P_{CSP})$ then obviously (2) holds as well. Thus, $s(P_{CSP})$ is also a solution for the negative BCSP. Conversely, let $s(D_{CSP}) = (y_{iu} \mid 1 \leq i \leq n, u \in D_i)$ denote a solution of the negative BCSP. If $y_{iu} = true$ in $s(D_{CSP})$, then for any j , by (2), there exists no value v such that $(u, v) \in \overline{\mathcal{R}_{ij}}$ and $y_{jv} = true$. Since V_j must take at least one value, it means that there exists a value $v_k \in D_j$, with $(u, v_k) \notin \overline{\mathcal{R}_{ij}}$ such that $y_{jv_k} = true$. In other words, (1) also holds. If $y_{iu} = false$ in $s(D_{CSP})$, then (1) holds as well. Thus, $s(D_{CSP})$ is also a solution for the positive BCSP. \square

2.2 Linear Models of BCSPs

Our discussion of the two representations for BCSPs in Section 2.1, and in particular the formulation of constraints as logical implications provides the basis to model BCSPs as 0-1 integer linear programs: A logical formula written in *conjunctive normal form* (CNF) can be easily modeled as a set of inequalities involving 0-1 variables. Using the fact that $a \rightarrow b \equiv \neg a \vee b$, and that $a \vee (b \wedge c) \equiv (a \vee b) \wedge (a \vee c)$, we can write (1) and (2) in CNF in the following way:

$$y_{iu} \rightarrow \bigvee_{v:(u,v) \in \mathcal{R}_{ij}} y_{jv} \equiv \neg y_{iu} \vee \left(\bigvee_{v:(u,v) \in \mathcal{R}_{ij}} y_{jv} \right) \quad (3)$$

and

$$y_{iu} \rightarrow \bigwedge_{v:(u,v) \in \overline{\mathcal{R}}_{ij}} \neg y_{jv} \equiv \neg y_{iu} \vee \left(\bigwedge_{v:(u,v) \in \overline{\mathcal{R}}_{ij}} \neg y_{jv} \right) \equiv \bigwedge_{v:(u,v) \in \overline{\mathcal{R}}_{ij}} (\neg y_{iu} \vee \neg y_{jv}) \quad (4)$$

Let $x_{iu} \in \{0, 1\}$, $x_{iu} = 1$ iff $y_{iu} = \text{true}$. This allows us to rewrite (3) and (4) as linear inequalities in terms of x :

$$(1 - x_{iu}) + \sum_{v:(u,v) \in \mathcal{R}_{ij}} x_{jv} \geq 1 \quad (5)$$

and

$$(1 - x_{iu}) + (1 - x_{jv}) \geq 1, \forall v : (u, v) \in \overline{\mathcal{R}}_{ij} \quad (6)$$

Based on these formula, we are now ready to give the two IP formulations resulting from the positive and negative representations of a BCSP.

Positive IP model (P_{IP})

$$\begin{aligned} \max \quad & 0 \\ \text{s.t.} \quad & x_{iu} \leq \sum_{v:(u,v) \in \mathcal{R}_{ij}} x_{jv} \quad \forall i, \forall j, \forall u : (u, v) \in \mathcal{R}_{ij} \end{aligned} \quad (7)$$

$$\sum_{u \in D_i} x_{iu} = 1 \quad \forall i \in \{1, \dots, n\} \quad (8)$$

$$x_{iu} \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, \forall u \in D_i \quad (9)$$

Negative IP model (N_{IP})

$$\begin{aligned} \max \quad & 0 \\ \text{s.t.} \quad & x_{iu} + x_{jv} \leq 1 \quad \forall i, \forall j, \forall (u, v) \in \overline{\mathcal{R}}_{ij} \end{aligned} \quad (10)$$

$$\sum_{u \in D_i} x_{iu} = 1 \quad \forall i \in \{1, \dots, n\} \quad (11)$$

$$x_{iu} \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, \forall u \in D_i \quad (12)$$

The first set of constraints, (7) and (10) encode the constraints of the positive and negative BCSP and are equivalent to the inequalities (5) and (6), respectively. Constraints (8) and (11) state that each variable V_i must take one and only one value from its corresponding domain D_i . They are the same as the implicit constraints we discussed in Section 2.1 and recall that they are the ones that ensure the equivalence of the two models. The last set of constraints (9) and (12) forbid solutions in which x_{iu} take fractional values. These are of course the constraints that make solving both these IPs difficult and are commonly the ones that are relaxed first to solve such problems in operations research. The purpose of the following study is to show that the linear relaxation derived from the positive formulation is strictly stronger than the weakened Lagrangian relaxation of the negative formulation which is used in [12].

3 Integer Programming Relaxations and Filtering

Based on the positive and negative IP models developed in the previous section, we now investigate how they could be used for filtering. In [12] the research is based on the (N_{IP}) model. Two relaxation steps are taken: First, constraints (10) are aggregated and thereby weakened since new fractional solutions are introduced. This was done because the authors felt that the number of constraints in (10) were too many. Then, for a given potential assignment $V_p = q$, a Lagrangian relaxation is considered where all constraints in (10) that do not affect x_{pq} are softened by penalizing a violation rather than enforcing the constraints.

Without the first aggregation step, let us study the polytope of feasible solutions to the Lagrangian subproblem that evolves when we relax all constraints in (10) that do not affect x_{pq} . We call the LP relaxation of the following IP the *consistent value polytope*. Since it can be viewed as derived from the negative formulation, we denote it with $(CV - N)$:

$$(CV - N) : \begin{array}{ll} (1) \sum_{k:v_k \in D_i} x_{ik} = 1 & \forall 1 \leq i \leq n \\ (2) x_{pq} + x_{jl} \leq 1 & \forall 1 \leq j \leq n, l \in D_j, (q, l) \in \overline{R_{pj}} \\ (3) x \in \{0, 1\}^n & \end{array}$$

In [12], a large number of aggregated versions of these IPs with changing objectives need to be solved in order to compute the Lagrangian relaxation value. In $(CV-N)$ we did not aggregate any constraints, therefore we achieve a tighter relaxation. What is even more important, there exists a reformulation of $(CV-N)$ that is totally unimodular which allows us to solve these IPs by means of linear programming. Consider

$$(CV - P) : \begin{array}{ll} (1) \sum_{k:v_k \in D_i} x_{ik} = 1 & \forall 1 \leq i \leq n \\ (2) \sum_{l:(q,l) \in R_{pj}} x_{jl} \geq x_{pq} & \forall 1 \leq j \leq n, R_{pj} \in \mathcal{C} \\ (3) x \in \{0, 1\}^n & \end{array}$$

Of course, the reformulation of $(CV-N)$ above was motivated by what we called the positive formulation of BCSPs earlier. Formally, we can show:

Lemma 2. *The integer programs (CV-N) and (CV-P) are equivalent.*

Proof. When removing all constraints in the corresponding BCSP that do not involve V_p , then (CV-N) and (CV-P) are exactly the IPs that evolve from the negative and positive formulations of the resulting BCSP. Therefore, the proof of Lemma 1 shows that both IPs are indeed equivalent.

Even though the reformulation from a negative representation of constraints to their positive formulation appears academic at first, it has a very important consequence when the IP model is considered:

Theorem 1. *The integer program (CV-P) is totally unimodular.*

Proof. After eliminating all duplicate and all unit-vector columns from the constraint matrix, neither of which affect total unimodularity, we get the following structure:

$$\begin{array}{l}
 (1) \quad \left| \begin{array}{cccc} 1 & 0 & \dots & \dots & 0 \\ | & | & & & | \\ 0 & \dots & \dots & 0 & 1 \end{array} \right. \\
 \hline
 (2) \quad \left| \begin{array}{cccc} -1 & 1 & 0 & \dots & 0 \\ | & | & & & | \\ -1 & 0 & \dots & 0 & 1 \end{array} \right.
 \end{array}$$

We note that part (1) is now an identity matrix, so it does not affect total unimodularity and can also be eliminated. Then, in part (2), we can eliminate all unit vectors again and we are left with just one column where all entries are -1, i.e., every square submatrix of this column matrix is -1.¹ \square

As a consequence of Lemma 2 and Theorem 1, the linear relaxation of (CV-P) describes exactly the convex hull of feasible solutions to (CV-N). Consequently, the Lagrangian subproblem can be solved in polynomial time. This is hardly surprising from a CP perspective: the Lagrangian relaxation rids ourselves of all constraints that do not incorporate variable V_p . Consequently, polynomial arc-consistency methods perform perfectly in terms of filtering effectiveness on the relaxed BCSP.

From an IP perspective, the fact that we found a totally unimodular description of the polytope of the Lagrangian subproblems enables us now to solve a tighter Lagrangian relaxation than the one proposed in [12] simply by means of linear programming: It is a well-known fact that if the Lagrangian subproblem is totally unimodular (it is then sometimes also referred to as exhibiting the integrality property), then the Lagrangian relaxation and the linear continuous relaxation have the same value [1]. To make this point very clear: Theorem 1 states that the *Lagrangian subproblem* is TU. We can therefore solve the Lagrangian relaxation by means of linear programming. Then, the overall linear relaxation is of course not TU (which would indeed come as a big surprise as then the NP-hard BCSP was solvable in P).

In summary, we have shown that the linear relaxation on (P_{IP}) , while much easier to solve, is equivalent to the Lagrangian relaxation of (N_{IP}) . Consequently, it is *strictly better* than a Lagrangian relaxation on an aggregated version of (N_{IP}) . Therefore, the

¹ We owe the idea to this simplified proof to an anonymous referee.

filtering algorithms that we derive from the relaxation based on the positive model are more effective and faster than the one that is considered in [12]. Note that this improvement does not restrict the choice of objective function. We can, as it was suggested in [12], investigate specific assignments by maximizing different specific variables x_{pq} in turn, or we could choose a more global objective function and perform reduced cost filtering.

What we view as even more important here is that in the positive model we have found a way to formulate *binary constraints as collection of integer constraints with tighter linear programming relaxations*. Consequently, we have found an improved formulation that we can use when binary constraints constitute a part of the constraint structure of an optimization problem, where it is well-known that it is essential to exploit tight global bounds on the objective.

4 Experimental Evaluation

In our experimental study, we focus purely on feasibility problems and the idea presented in [12] to base an efficient filtering algorithm for BCSPs on mathematical programming methods. In order to base a filtering algorithm on the relaxations that we studied in the previous section, first we follow the second main contribution that was made in [12]. It consists in the introduction of an objective function that is assignment specific. In [12], the authors compute upper bounds on (N_{IP}) augmented by an objective that tries to maximize the value of one single variable x_{pq} . Clearly, if that upper bound drops below 1, then this implies that V_p cannot consistently take value q , and the value is removed from D_p .

In our first series of experiments, we try to reproduce the results reported in [12]. We follow their approach and solve a series of linear relaxations of (P_{IP}) with changing objectives to maximize x_{pq} for the different variables. As a result of Section 3, we know that this filtering technique is at least as effective as the one presented in [12].

The following first set of experiments was run on a 2.4 GHz Intel Hyperthreading processor with 2 GB RAM. In order to provide a close comparison with [12], we use randomly generated BCSPs as our benchmark set. The problems were generated using the random uniform BCSP generator available at [3]. For each experiment, we generated 200 random instances. The test programs were implemented using ILOG Concert 2.0 to interface with Solver 6.0 and CPLEX 9.0 [10]. We generated problems of comparable size and structure: 16 variables, 8 values per domain, and 32 allowed pairs per constraint. We varied the number of constraints from 10 to 120 in increments of 10. In order to verify the validity of the observed trends, we also used a second class of problems, smaller in size, with the following characteristics: 10 variables, 10 values per domain, 32 allowed pairs. For these problems, we varied the number of constraints from 5 to 50 in increments of 5.

To assure that our experimentation is correct, first we solved all problems in our test set to completion using ILOG Solver 6.0 and looked at the distribution of feasible instances. The results are shown in Figure 1, and it is clear that, as the number of constraints approaches 120 for the larger problems and 50 for the smaller ones, the number of feasible instances drops sharply. This is a typical phase transition phenomenon, and

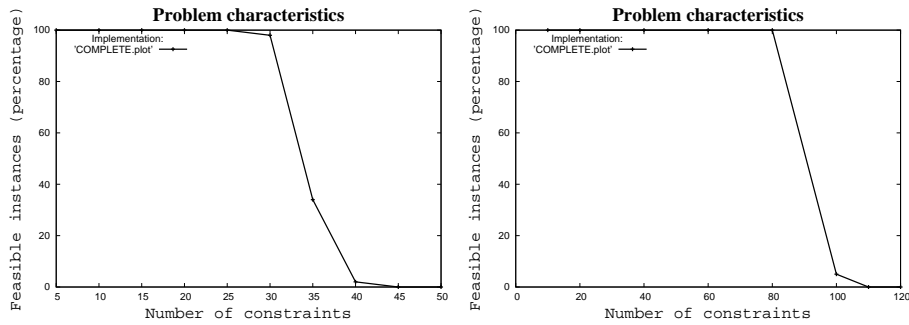


Fig. 1. Percent of solvable instances over the number of constraints for the small (left) and large (right) instances

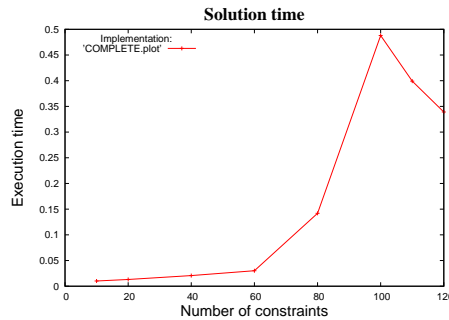


Fig. 2. Time (sec) required by a pure CP solver to solve the BCSPs

an easy-hard-less hard partition is visible by the time required by the solution algorithm on these instances. Figure 2 shows the time needed by a standard CP solver for the large benchmark. It is clearly visible that the hardest instances are those around the phase transition.

The percentage of values filtered using the relaxation of (P_{IP}) at the root node is plotted in Figure 3, for both sets of problem instances (small and large). This confirms the results reported in [12] where it was found that hybrid filtering is *far more effective* than standard arc-consistency algorithms *at the root node*. On our problems, at the root node arc-consistency is unable to filter any substantial number of values, which is why the corresponding line runs close to the 0% horizontal.

However, what is not made explicit in [12] is that the high percentage of filtered values when the number of constraints gets closer to 120 is actually due to the fact that most problems in that range are *infeasible* and that relaxation-based filtering is able to detect that at the root node! On infeasible problems, the filtering algorithm naturally reports 100% removal of values. It is solely due to this effect that the time per filtered value decreases so massively as it was reported in [12].

It is also important to mention that this performance is obtained only if we iterate the filtering process (i.e. solve relaxations of (P_{IP}) as long as we have at least one filtered

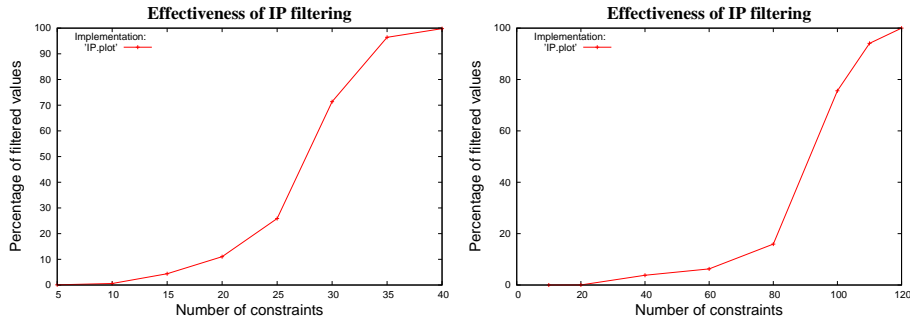


Fig. 3. Percentage of filtered values using the relaxation of (P_{IP}) for the small (left) and large (right) instances

value in a round). If we perform a single round of IP filtering (i.e. solve the relaxation of (P_{IP}) once for each variable x_{iu}), the number of filtered values grows only to about 30% as we approach 120 constraints. The large difference can be explained by the fact that, for most problems, the LP relaxation is unable to detect infeasibility in only one round. It typically does so after 4-5 rounds, and then the percentage of filtered values reported jumps to 100%. Obviously, an iterated application of the filtering algorithm increases the effectiveness — but of course it comes at the cost of more cpu time, which, as we will see shortly, is too much to make this kind of filtering worthwhile in the context of random BCSPs.

We also studied the effect of constraining the problem in a different way: namely by varying the number of allowed pairs per constraint instead of varying the number of constraints. For this experiment, we generated problems with 16 variables, 8 values per domain, 60 constraints and varied the number of valid pairs from 5 to 60 in increments of 5. The results are shown in Figure 4. Again, we observe the a clear phase transition, which happens at around 30 pairs per constraint, and that is supported by the problem characteristics observed in Figure 5.

So far we have been able to confirm the results reported in [12]. Now, we were of course curious to see whether the idea of iterated LP-based filtering with assignment specific objectives actually pays off within a tree search. After all, while the improvements in filtering effectiveness at the root node are quite good, what we are ultimately interested in is of course the time that it takes to complete the search and actually solve instances. Therefore, we study how fast the LP filtering is compared to that of the constraint solver. While for virtually every instance that we studied, the first propagation step of the constraint solver failed to remove *any* values from the domains of the variables, the performance of arc-consistency techniques *within a tree search* is far better: When comparing the time the constraint solver took to solve the *entire problem* with the time it took the LP approach just to filter *at the root node*, we see that the difference is hugely in favor of the constraint solver, by orders of magnitude. While at the phase transition (where more effective filtering should be of most importance) the time to filter according to (P_{IP}) only at the root-node peaks at around 150 seconds, standard arc-

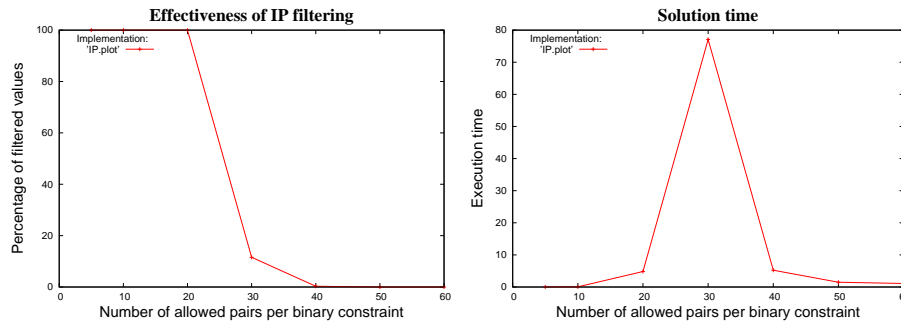


Fig. 4. Effect of the number of allowed pairs on the performance of \mathbf{P}_{IP_1} on the percentage of solvable instances (left) and the propagation time at the root node for the relaxation based filtering method (right)

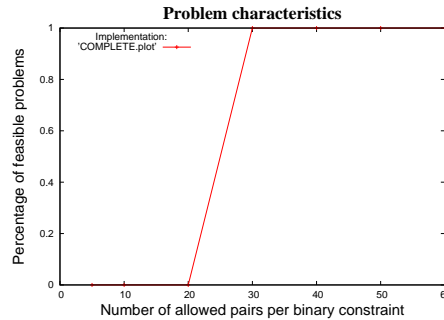


Fig. 5. Characteristics of the instances where we varied the number of pairs per constraint

consistency algorithms complete the entire search in half a second on average (compare Figures 6 and 2). Consequently, despite the far more effective filtering that they offer, the algorithms published in [12] are just not worthwhile to solve random BCSPs.

To summarize our findings to this point: filtering binary constraints based on mathematical programming relaxations is more effective than standard arc-consistency methods, thanks to the global view on the problem that the relaxation provides. However, even despite our strengthening the relaxation and speeding up its computation time by showing that an alternative LP relaxation dominates the Lagrangian relaxation introduced in [12], the idea to use an iterated procedure to filter every domain value individually is just far too costly to pay off within a tree search — no matter whether we compare at the under-constrained, over-constrained, or critically constrained region.

In order to improve the efficiency of LP-based filtering, we need to make it less expensive, even at the cost of losing some of its vast effectiveness. Therefore, we tried out two different kinds of weakened approaches: The first computes an initial LP-solution to the problem, then it chooses those assignments $X_p = q$ for which the continuous value of x_{pq} is lower than some threshold value $\epsilon > 0$, and finally it sets up a new objective for each of those variable with one filtering iteration only. We refer to this ap-

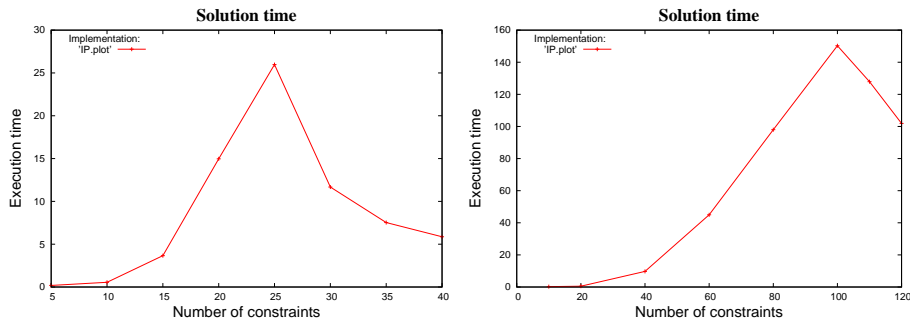


Fig. 6. Time (sec) required by the relaxation of (PIP) to complete filtering

proach as *LP-filtering*. The second approach sacrifices even more effectiveness by using the LP-relaxation just for pruning purposes. It just solves the initial LP once and backtracks if and only if that LP turns out to be infeasible. We denote this second approach with *LP-pruning*.

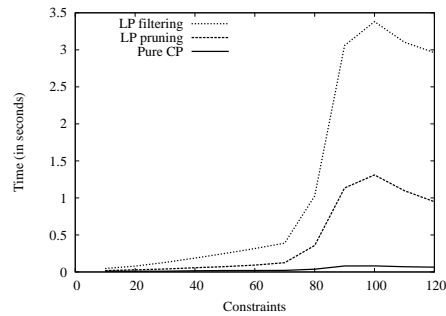


Fig. 7. Comparison of pure CP, LP-pruning, and LP-filtering on random BCSP instances

We performed a second set of experiments to compare the performance of LP-based BCSP propagation and pure CP. The following test results show the averages² over 30 runs per data point on a 2 GHz AMD Athlon processor with 512 MB RAM. Figure 7 visualizes the results of our experiments on the large benchmark of random BCSP instances with 16 variables, 8 domain values per variable, and 32 allowed pairs per binary constraint. Again, we see a clear easy-hard-less hard pattern. The comparison shows that a pure CP solver is orders of magnitude faster than LP-filtering and LP-pruning, whereby the latter, despite its weaker effectiveness, is still about twice as fast.

² Although we can only visualize averages in our plots, we would like to mention that we also checked the medians and variances to eliminate the possibility that some extreme outliers disproportionately bias the comparison.

For our last experiment, we were curious whether the good efficiency of pure arc-consistency methods was maybe caused by the unstructured character of our benchmark set. Therefore, we repeated the experiment in Figure 7 on a benchmark set that contains 13-queens instances with additional random binary constraints on the queens. We use the standard CP model where we add one queen-variable for each column and the values that they take correspond to the row index that the queen takes. All different constraints on rows, columns, and diagonals enforce the 13-queen problem. In Figure 8 we plot the percentage of feasible instances and the solution time by our three solvers over the number of (additional) binary constraints added to the problem.

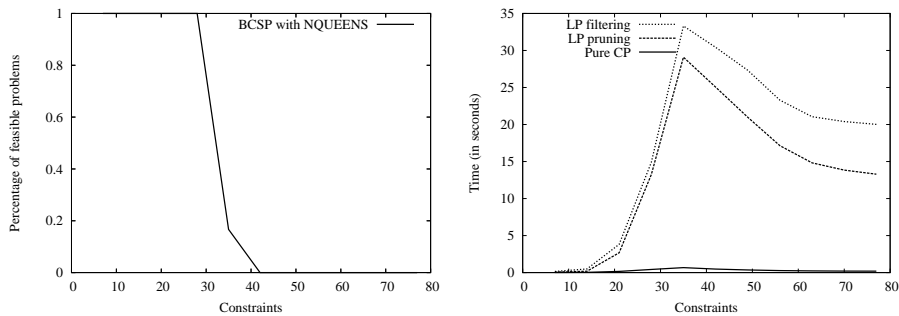


Fig. 8. Comparison of pure CP, LP-pruning, and LP-filtering on random BCSP instances

We see that LP-filtering is able to catch up with LP-pruning, but the comparison with the pure CP solver is devastating. We conclude that the idea of basing a BCSP filtering algorithm on mathematical programming just does not pay off within a tree search.

Of course, it is well-known fact that the use of relaxations is essential for many optimization problems. For this case, when binary constraints are part of the problem, we have introduced a linear programming formulation that approximates the convex hull of feasible integer solutions better than previously studied relaxations. However, for pure feasibility problems, we find that pure CP is the method of choice.

5 Conclusion

We presented a filtering algorithm based on linear programming (LP) models for BCSPs. The LP relaxations that we used are provably stronger than those developed in [12]. At the same time, filtering can now be based on standard linear programming technology which reduces the programming effort and speeds up the filtering process considerably. Our numerical results show that LP-based filtering for BCSPs leads to more effective filtering. In so far, we can confirm the findings in [12]. However, ultimately we are interested in solving BCSPs by search methods. And in the realm of search, what matters is not so much the effectiveness of filtering methods, but the trade-off between effectiveness and time, i.e. efficiency. Our experiments on random instances show

clearly that the additional time for filtering based on mathematical programming does not pay off for BCSPs when compared with standard CP arc-consistency techniques. We therefore reconfirm the common (yet to the best of our knowledge unpublished) belief that hybrid methods perform very poorly on BCSPs: for these problems, leaner and faster inference continues to be the right way to go.

References

1. R.K. Ahuja, T.L. Magnati, J.B. Orlin. *Network Flows*. Prentice Hall, 1993.
2. G. Appa, D. Magos, I. Mourtos, An LP-based proof for the non-existence of a pair of Orthogonal Latin Squares for $n=6$. *OR Letters*, 32(4): 336–344, 2004.
3. C. Bessiere. Random Uniform CSP Generators. <http://www.lirmm.fr/~bessiere/generator.html>.
4. T. Fahle, U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59-81, 2002.
5. T. Fahle, M. Sellmann. Cost-Based Filtering for the Constrained Knapsack Problem. *Annals of Operations Research*, 115:73–93, 2002.
6. F. Focacci, A. Lodi, M. Milano. Cutting Planes in Constraint Programming: An Hybrid Approach. *Proceedings of CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:45–51, 2000.
7. F. Focacci, A. Lodi, M. Milano. Cost-Based Domain Filtering. *Principles and Practice of Constraint Programming (CP)* Springer LNCS 1713:189–203, 1999.
8. J.N. Hooker. A hybrid method for planning and scheduling. *Proceedings of Principles and Practice of Constraint Programming (CP 2004)*, Springer LNCS 3258:305–316, 2004.
9. J.N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
10. ILOG SA. ILOG Concert 2.0. <http://www.ilog.com>.
11. U. Junker, S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. A Framework for Constraint programming based column generation. *Principles and Practice of Constraint Programming (CP)*, Springer LNCS 1713:261–274, 1999.
12. M.O.I. Khemmoudj, H. Bennaceur, A. Nagih. Combining Arc-Consistency and Dual Lagrangean Relaxation for Filtering CSPs. *Proceedings of CPAIOR'05*, LNCS 3524:258–272, 2005.
13. H-J. Kim and J. N. Hooker. Solving fixed-charge network flow problems with a hybrid optimization and constraint programming approach. *Annals of Operations Research* 115:95–124, 2002.
14. M. Milano. *Integration of Mathematical Programming and Constraint Programming for Combinatorial Optimization Problems*, Tutorial at CP2000, 2000.
15. G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.
16. G. Ottosson, E.S. Thorsteinsson. Linear Relaxation and Reduced-Cost Based Propagation of Continuous Variable Subscripts. *CP-AI-OR'00*, Paderborn Center for Parallel Computing, Technical Report tr-001-2000:129–138, 2000.
17. J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7(3-4):387–405, 2002.
18. Meinolf Sellmann. Theoretical Foundations of CP-based Lagrangian Relaxation. *Proceedings of the 10th intern. Conference on the Principles and Practice of Constraint Programming (CP)*, Springer LNCS 3258:634-647, 2004.
19. M. Sellmann. Approximated Consistency for Knapsack Constraints. *CP*, Springer LNCS 2833: 679–693, 2003.

20. M. Sellmann and T. Fahle. Constraint Programming Based Lagrangian Relaxation for the Automatic Recording Problem. *Annals of Operations Research*, 118:17-33, 2003.
21. M. Sellmann and T. Fahle. Coupling Variable Fixing Algorithms for the Automatic Recording Problem. *Annual European Symposium on Algorithms (ESA)*, Springer LNCS 2161: 134–145, 2001.
22. M. Sellmann and W. Harvey. Heuristic Constraint Propagation. *Proceedings of the 8th intern. Conference on the Principles and Practice of Constraint Programming (CP)*, Springer LNCS 2470: 738–743, 2002.