

The Linear Programming Polytope of Binary Constraint Problems with Bounded Tree-Width

Meinolf Sellmann, Luc Mercier, and Daniel H. Leventhal

Brown University
Department of Computer Science
115 Waterman Street, P.O. Box 1910
Providence, RI 02912
sello,mercier@cs.brown.edu

Abstract. We show how to efficiently model binary constraint problems (BCP) as integer programs. After considering tree-structured BCPs first, we show that a Sherali-Adams-like procedure results in a polynomial-size linear programming description of the convex hull of all integer feasible solutions when the BCP that is given has bounded tree-width.

Keywords: constraint programming, integer programming, polyhedral combinatorics, cutting planes

1 Introduction

When solving combinatorial problems, all competitive state-of-the-art solvers combine search with inference. Integer programming (IP) solvers like XpressMP or Cplex base inference on tight continuous linear programming (LP) relaxations. Satisfiability (SAT) solvers perform unit propagation and no-good learning. And constraint programming (CP) solvers make excessive use of constraint filtering techniques. The efficiency, i.e. the effectiveness over CPU-time, is the decisive performance measure for inference algorithms that are used within search.

In IP, effectiveness is best characterized by the gap between the value of the optimal solution and the bound that results from the continuous relaxation of the problem. For certain problems, we find that there is no gap at all. This is the case, for example, when the constraint matrix is totally unimodular and the right hand side vector is integer. In this case, the strength of the inference algorithm alone allows us to solve the corresponding problem in polynomial time.

Interestingly, such islands of tractability are not always found by the analysis of inference algorithms. For certain problems, we can also devise ways how to *search* more efficiently. This is for example the case when we solve a problem by means of dynamic programming. Another example are search algorithms

that exploit problem decomposability, such as polynomial-time algorithms for problems on graphs with bounded tree-width [15]. A recent heuristic algorithm that exploits structure to speed-up search is given in [12].

For practical reasons, a perfect inference algorithm is preferable over a specialized search routine. Consider for example the case where we hit an island of tractability as a subproblem within a general search. An inference algorithm automatically takes advantage of it, while a specialized search routine could only be invoked when the necessary conditions for its efficient application are detected. Moreover, consider the case where an IP has just a few additional constraints that compromise problem decomposability, thus preventing a specialized search algorithm from being applicable. Then, a tight description of the polytope that the majority of the constraints define still helps to tighten the LP/IP gap.

Consequently, researchers are interested in describing or at least approximating the convex hull of tractable problems. A recent example for such work is [3]. Given that Knapsack problems can be approximated efficiently, Vyve and Wolsey [17] had raised the question whether, for all $\varepsilon > 0$, the Knapsack polytope over n items can be approximated by at most a polynomial number (in n and $1/\varepsilon$) of cuts so that the LP relaxation value does not over-estimate the optimal IP value by more than a factor of $1+\varepsilon$. This could be viewed as the mathematical programming analogue of a fully polynomial approximation scheme (FPTAS) for the Knapsack polytope. Bienstock provides the analogue of a polynomial approximation scheme (PTAS) by giving a lifted formulation of the Knapsack polytope with $O(n^{1+\lceil 1/\varepsilon \rceil}/\varepsilon)$ variables and $O(n^{2+\lceil 1/\varepsilon \rceil}/\varepsilon)$ constraints.

In this paper, we study the polytope of tree-structured binary constraint networks. It is well known that these problems can be solved in polynomial time by a specialized search based on problem decomposition. We bring this result to the realm of inference by providing a perfect characterization of the corresponding polytope. Particularly, we show that a certain set of linear constraints leads to continuous relaxations with no LP/IP gap. By introducing Sherali-Adams-like variables, we then generalize the result to problems with bounded tree-width.

2 Binary Constraint Satisfaction

We start our study by defining binary constraint satisfaction problems.

Definition 1 (Binary Constraint Satisfaction Problem).

- A binary constraint problem (BCP) is a triplet $\langle V, D, C \rangle$, where $V = \{X_1, \dots, X_n\}$ denotes the finite set of variables, $D = \{D_1, \dots, D_n\}$ denotes a set of n finite sets of possible values for these variables (D_i is called the domain of variables X_i), and $C = \{C_1, \dots, C_m\}$ is the set of constraints, where $C_j : D_{j_1} \times D_{j_2} \rightarrow \text{Bool}$ specifies which simultaneous assignments of values to the variables X_{j_1} and X_{j_2} are allowed. The set $\{X_{j_1}, X_{j_2}\}$ is called the scope of constraint C_j .
- An assignment for a BCP $\mathcal{P} = \langle V, D, C \rangle$ is a function $\sigma : V \rightarrow \bigcup_{i \leq n} D_i$. A solution to a BCP $\mathcal{P} = \langle V, D, C \rangle$ is an assignment σ such that $\sigma(X_i) \in D_i$ for all $1 \leq i \leq n$ and such that $C_j(\sigma(X_{j_1}), \sigma(X_{j_2})) = \mathbf{true}$ for all $1 \leq j \leq m$. The set of all solutions to a BCP \mathcal{P} is denoted by $\text{Sol}(\mathcal{P})$.

Note how, in contrast to the custom in integer programming, in CP the term “binary” is used to express that all *constraints* affect just two variables, while the size of the *domain* of each variable is not limited! The fact that the arity of the constraints is limited to two allows us to state constraints simply as sets of allowed pairs $\mathcal{R}_{j_1, j_2} = \{(k, l) \mid X_{j_1} = k, X_{j_2} = l \text{ ok}\}$, or, alternatively, as sets of forbidden pairs $\overline{\mathcal{R}}_{j_1, j_2} = \{(k, l) \mid X_{j_1} = k, X_{j_2} = l \text{ forbidden}\}$.

It is easy to see that the general BCP is NP-hard. One simple way is to reduce from graph coloring where each node is modeled as a variable that must be assigned a color such that adjacent nodes are not colored identically (i.e., the corresponding constraint on each edge $\{i, j\}$ is a not-equal constraint $\overline{\mathcal{R}}_{i, j} = \{(k, k) \mid \forall k\}$). Conversely, every binary constraint problem can be visualized as a *constraint network* where each node corresponds to a variable and an edge connects two nodes iff there exists a constraint over the corresponding variables. Of course, the exact semantic of the constraints is lost in that visualization. However, it is a well-known fact that any BCP whose corresponding constraint network is a tree can be solved in polynomial time. Even more generally, a BCP is already tractable when its constraint network has bounded tree-width [6, 7].

In the following, we consider ways to express BCPs by means of linear constraints. We will review a recently introduced IP model for BCPs and show that, for tree-structured BCPs, the model gives a perfect representation of the convex hull of all integer feasible solutions.

3 The Support Formulation

When designing constraints for a model, humans tend to think in terms of “what is forbidden.” For BCPs, this leads to the common IP formulation in which we consider each constraint truth table over variables X_i and X_j and add an IP

constraint $y_{ik} + y_{jl} \leq 1$ for each inconsistent pair $(k, l) \in \overline{\mathcal{R}_{i,j}}$, where $y_{pq} \in \{0, 1\}$ is one iff $X_p = q$ in the solution to the BCP. Since each variable must take exactly one value, we also add constraints $\sum_k y_{ik} = 1$ for all $1 \leq i \leq n$. When the task is constraint optimization rather than constraint satisfaction as in CP, we are also given a linear objective function. The complete IP then reads:

Traditional IP model (T_{IP})

$$\begin{aligned} \max \quad & \sum p_{ik} y_{ik} \\ \text{s.t.} \quad & y_{j_1 k} + y_{j_2 l} \leq 1 \quad \forall 1 \leq j \leq m, (k, l) \in \overline{\mathcal{R}_{j_1, j_2}} \quad (1) \end{aligned}$$

$$\sum_{k \in D_i} y_{ik} = 1 \quad \forall i \in \{1, \dots, n\} \quad (2)$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, k \in D_i \quad (3)$$

In [2], we provided a different way of formulating a BCP as an IP by focussing on the allowed pairs in each constraint truth table. In essence, we use the linear constraints to specify that, when a variable X_{j_1} takes value k , variable X_{j_2} must take a value that is consistent with $X_{j_1} = k$. In that way, we enforce that each variable assignment is *supported* by a correct assignment to adjacent variables (by which we mean variables that share a constraint). The IP then reads:¹

Support IP model (S_{IP})

$$\begin{aligned} \max \quad & \sum p_{ik} y_{ik} \\ \text{s.t.} \quad & y_{j_1 k} - \sum_{l: (k, l) \in \mathcal{R}_{j_1, j_2}} y_{j_2 l} \leq 0 \quad \forall 1 \leq j \leq m, k \in D_{j_1} \quad (4) \end{aligned}$$

$$\sum_{k \in D_i} y_{ik} = 1 \quad \forall i \in \{1, \dots, n\} \quad (5)$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, k \in D_i \quad (6)$$

The fact that support encodings can lead to strong inference algorithms is not new. In [10, 8], for example, it was shown that formulating a BCP as a SAT formula in this way has preferable propagation properties. In [2], we showed that the support IP formulation leads to stronger linear continuous relaxations than the Lagrangian relaxation proposed in [11]. Specifically, we showed that the support encoding of the Lagrangian subproblem studied in [2] is totally unimodular. Unfortunately, this property does not hold for the entire IP, even when the corresponding BCP has just two variables (consider for example the case of a two node/one edge graph coloring problem with three colors.) Despite this problem, we will now show that the linear continuous relaxation S_{LP} of S_{IP}

¹ Whereby, for simplicity in constraints (4), we assume that each constraint over variables i, j induces two truth tables $\mathcal{R}_{i,j}$ and $\mathcal{R}_{j,i}$.

(which is given by S_{IP} when replacing integrality constraints (6) with simple constraints on the bounds of the variables; in our case, the upper bounds are redundant and are therefore left out) provides a perfect characterization of the convex hull of all integer feasible solutions.

4 Tree-Structured Binary Constraint Programs

We state our result as follows:

Theorem 1. *If the BCP that is given has a tree-structured constraint network, then the programs S_{IP} and S_{LP} have the same value.*

Proof. Assume we are given a (potentially fractional) solution to S_{LP} (if there is no solution then the corresponding BCP is obviously infeasible, too). Denote this solution by y_{ik}^0 with $1 \leq i \leq n$ and $k \in D_i$. Now consider the following sets: $D_i^0 := \{k \mid y_{ik}^0 > 0\} \subseteq D_i$. We make two important observations:

- (a) First, none of the sets D_i^0 is empty.
- (b) And second, for each $1 \leq j \leq m$ and each value $k \in D_{j_1}^0$, there exists a value $l \in D_{j_2}^0$ such that setting $X_{j_1} = k$ and $X_{j_2} = l$ is allowed. Analogously, for each $1 \leq j \leq m$ and each value $l \in D_{j_2}^0$, there exists a value $k \in D_{j_1}^0$ such that setting $X_{j_1} = k$ and $X_{j_2} = l$ is allowed.

The first is a simple consequence of constraints (5), the latter follows from constraints (4) which enforce that at least one non-conflicting value must still be present in the adjacent variable's domain:

$$0 < y_{j_1 k} \leq \sum_{l: (k,l) \in \mathcal{R}_{j_1, j_2}} y_{j_2 l}.$$

The second property is known in CP as *arc-consistency*. Basically, arc-consistency just states that each value in a variable's domain has supporting values in each of the domains of adjacent variables. It is a well-known fact that a tree-structured BCP has a solution if properties (a) and (b) hold for the domains of the variables. This is easy to verify: Assume that the BCP that is given is arc-consistent. Take any variable and assign to it any value in its domain. Shrink the domains in the remaining BCP by removing all values without support until it is arc-consistent again. Since, for each value in each domain, there exists at least one supporting value in the domains of adjacent variables, no domain can be empty now. So we have properties (a) and (b) again, and we repeat. After at most n such steps all domains have become singletons and we can read out the integer feasible solution y^1 .

Therefore, when we artificially shrink the domains of the variables in the given tree-structured BCP by replacing D_i with D_i^0 , then the remaining constraint problem is still solvable.

It remains to show that any solution to the reduced BCP where each variable X_i takes a value in D_i^0 has the same objective value as our fractional solution. For this purpose, consider the dual optimal solution with variables $\pi_{jk} \geq 0$ for constraints (4) (and unsigned variables μ_i for constraints (5) that we will not use). Consider the relaxed problem where we soften constraints (4) and penalize their violation in the objective with the help of Lagrange multipliers π :

$$\begin{aligned} \max \quad & \sum p_{ik} y_{ik} - \sum_{j,k \in D_j} \pi_{jk} y_{jk} + \sum_{j_1} \sum_{(k,l) \in \mathcal{R}_{j_1, j_2}} \pi_{jk} y_{j_2 l} \\ \text{s.t.} \quad & \sum_{k \in D_i} y_{ik} = 1 && \forall i \in \{1, \dots, n\} \\ & y_{ik} \in \{0, 1\} && \forall i \in \{1, \dots, n\}, k \in D_i \end{aligned}$$

which can be simplified to

Relaxed IP model ($R_{IP}(\pi)$)

$$\begin{aligned} \max \quad & \sum_{s \leq n, t \in D_s} \overline{p_{st}} y_{st} \\ \text{s.t.} \quad & \sum_{k \in D_i} y_{ik} = 1 && \forall i \in \{1, \dots, n\} \end{aligned} \quad (7)$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, k \in D_i \quad (8)$$

when setting $\overline{p_{st}} := p_{st} - \sum_{j_1=s} \pi_{j_1 t} + \sum_{j_2=s, \exists u: (u,t) \in \mathcal{R}_{j_1, j_2}} \pi_{j_1 t}$. From the theory of Lagrangian relaxation we then know the following facts [1, 14]:

- The Lagrangian subproblem exhibits the integrality property. Therefore, with optimal dual multipliers π , the optimal value of $R_{IP}(\pi)$, $R_{LP}(\pi)$, and S_{LP} are all the same.
- Then, the optimal fractional solution y^0 to S_{LP} is also optimal for $R_{LP}(\pi)$. Consequently, because of constraints (7), for all $1 \leq i \leq n$, it holds that $\overline{p_{ik}} = \overline{p_{il}}$ for all $k, l \in \{t \mid y_{it}^0 > 0\}$. That is, all domain values of variable i that received positive weight have the same reduced costs $\overline{p_{ik}}$.

Thus, our integer solution y^1 to S_{IP} for which $y_{ik}^1 = 1$ implies $k \in D_i^0$ is feasible and optimal for $R_{IP}(\pi)$. And so, y^1 has the same objective value as y^0 . Thus, for each fractional optimal solution there exists an integer optimal solution with the same objective value. \square

Since Theorem 1 applies regardless of the objective function, what we have really shown is that the polytope described by S_{LP} is exactly the convex hull of the feasible solutions to S_{IP} . Note that there are certainly other ways to prove this result. However, this is the only proof we know which reveals a connection between the CP variable domains and the support (i.e. the set of those LP variables that have positive weight) of the LP relaxation.

So we can efficiently describe the convex hull of IPs modeling tree-structured BCPs. Since tree-structured problems are known to be polynomial-time solvable, this is not really surprising. However, the characterization of the convex hull has several advantages over specialized search procedures:

- In many cases, binary constraints constitute only a part of the constraint structure. Specialized search procedures that work for the BCP part do not generalize to the problem at hand. However, the tight description of the polytope of feasible solutions can still be exploited in a branch-and-bound procedure.
- Assume that the overall problem is not tree-structured, but becomes tree-structured after a couple of branching decisions. In practice, we do not want to put large overhead into the recognition of special cases that can be solved by a specialized search routine. When using the support formulation, special recognition is not necessary - the relaxation will automatically come out integer when using the simplex algorithm to solve the linear relaxation.

5 Exploiting Bounded Tree-Width

Obviously, we would like to generalize our results to BCPs with bounded tree-width which are also known to be polynomial-time tractable. We do this in two steps: First, we propose a set of constraints that define the convex hull of a BCP with bounded tree-width k when a tree-decomposition is known. In the second step, we then show how to achieve a perfect bound when the decomposition is not known.

Definition 2. *An undirected graph $G = (N, A)$ has tree-width k , iff there is a pair (S, T) , where $S = \{S_1, \dots, S_p\}$ is a family of subsets of N , and T is a tree whose nodes are the subsets S_i , such that:*

- $\bigcup_i S_i = N$,
- for every edge $\{v, w\} \in A$, there is a node S_i that contains both v and w ,
- if S_i and S_j both contain a vertex v , then all nodes S_z of the tree in the (unique) path between S_i and S_j contain v , and
- the size of all sets S_i is limited by $k + 1$, i.e., $|S_i| \leq k + 1$ for all i .

A BCP has bounded tree-width k when its constraint network has tree-width k .

It is known that a BCP with bounded tree-width k can be transformed into a tree-structured BCP that contains the original variables as well as some additional auxiliary variables such that the size of the new BCP is polynomial in the original size (and exponential in the constant k). We can compute this new BCP by exploiting the tree-decomposition of the given problem. Note that, while computing the minimal tree-width of a graph is NP-hard, for fixed k we can efficiently check whether a graph has tree-width k and compute a corresponding tree-decomposition in linear time [5].

The construction of the new BCP is simple: the variables are the old variables X_1, \dots, X_n plus new variables Y_1, \dots, Y_p , one for each set $S_i = \{X_{i_1}, \dots, X_{i_{q(i)}}\}$ in the tree-decomposition (whereby $1 \leq q(i) \leq k+1$). The domain of variable Y_i are all tuples $(t_1, \dots, t_{q(i)}) \in D_{i_1} \times \dots \times D_{i_{q(i)}}$ that are *consistent with all binary constraints* over any two variables in S_i . The domains of the old variables are the same as before. Now, we add two sets of binary constraints to the problem:

- For each variable X_j there exists a set S_i such that $X_j \in S_i$. For one such set (ties can be broken arbitrarily), we add a constraint over X_j and Y_i that enforces that the value of X_j and the corresponding entry in the tuple assigned to Y_i are the same.
- For each pair of variables Y_h, Y_i such that $S_h \cap S_i \neq \emptyset$, we add a constraint that enforces that entries in the corresponding tuples that are associated with the same variable(s) X_j are identical.

Note how the restrictions that apply to a valid tree-decomposition ensure that the new BCP is connected, cycle-free, and polynomial in the size of the original problem when k is viewed as a constant. Most importantly, a valid tree-decomposition ensures that a solution to the original problem has a corresponding solution in the new BCP and vice versa: Given a solution σ to the original problem, we achieve a solution to the new one simply by assigning the same values to the old variables as before and by assigning the corresponding consistent tuples to the variables Y_i . Since σ obeys all original constraints, each such tuple is a member in the domain of Y_i . Conversely, assume we are given a solution τ to the new problem. Note that, for all X_j , the variables Y_i with $X_j \in S_i$ and X_j are all connected. Consequently, all tuples assigned to variables Y_i and the assignment to X_j agree on the value of X_j . Also, for each original constraint C_j on variables X_{j_1}, X_{j_2} there is at least one Y_i such that $X_{j_1}, X_{j_2} \in S_i$. Therefore, τ assigns a consistent pair of values to X_{j_1} and X_{j_2} . Consequently, when projecting τ on variables X_j , we achieve a feasible solution to the original problem.

Now, all that we need to do is formulate the new, tree-structured BCP by exploiting the support formulation as outlined in Section 3. Then, all extreme points of the new polytope give integer feasible solutions that are consistent with the original problem. It follows:

Theorem 2. *Given a constant k , any BCP with bounded tree-width k can be expressed as an integer program that exhibits the integrality property and which size is polynomial in the size of the given problem.*

Note that our formulation still contains all original variables which makes it easy to add the objective to the problem and which also allows us to augment the problem by adding additional constraints on those original variables. Obviously, any additional constraints will, in general, compromise the integrality property of the feasible polytope. However, with the tight characterization of the convex hull of all solutions feasible for the BCP we can hope for a small LP/IP gap.

Now, in some cases we would prefer if we did not need to know the exact tree-decomposition of our problem. For example, it would be desirable if, during a branch-and-bound search, inference on a sub-problem encountered during search would simply turn out perfect whenever the sub-problem has bounded tree-width k , without the need of applying a special recognition algorithm for all sub-problems. That is, rather than analyzing a problem, determining its tree-width, and then choosing the formulation, it would be nicer if we could simply fix the parameter k (as an algorithm design choice) and then be sure that all subproblems of width lower than k that we may ever encounter will be solved perfectly by our inference algorithm. Fortunately, this can be achieved!

First, the following simple claim allows us to restrict our attention to a certain class of tree decompositions, that we call *saturated*.

Lemma 1. *Let $G = (N, A)$ be an undirected graph of tree-width less than or equal to k , with $|N| > k$. There exists a tree decomposition (S, T) of G such that $|s| = k + 1$ for all $s \in S$.*

Proof. First, we may assume there is no edge $\{S_i, S_j\} \in T$ such that $S_i \subseteq S_j$ since we can get a new tree decomposition (S', T') by contracting this edge and keeping only S_j . Now, if $|S| = 1$, then the tree consists in one node containing all $k + 1$ elements of N . So assume $|S| > 1$, and there is $S_i \in S$ such that $|S_i| \leq k$. Let S_j be adjacent to S_i . Since there exists $p \in S_j \setminus S_i$, we can augment the cardinality of S_i by adding p which gives a new valid tree-decomposition. By repetition, we achieve a decomposition where all nodes have cardinality $k + 1$. \square

Now, we formulate a new BCP whose solutions closely correspond to solutions in the original problem. The construction is very easy: We add all original variables X_j to our problem, keeping their domains as before. Additionally, for each subset of exactly $k + 1$ variables $\{X_{s_1}, \dots, X_{s_{k+1}}\}$, we introduce a variable $Y_{s_1, \dots, s_{k+1}}$. The domains of these variables are limited to tuples $(t_1, \dots, t_{k+1}) \in D_{s_1} \times \dots \times D_{s_{k+1}}$ which are *consistent with all binary constraints* over variables X_{j_1}, X_{j_2} with $j_1, j_2 \in \{s_1, \dots, s_{k+1}\}$. We add the same binary constraints on the auxiliary variables as we did when the tree-decomposition was known. The original variables X_j are tied to our problem by adding a constraint between X_j and each variable $Y_{s_1, \dots, s_{k+1}}$ with $j = s_r$ for some $1 \leq r \leq k + 1$, enforcing that the value assigned to X_j is the same as the entry with index s_r of the tuple assigned to $Y_{s_1, \dots, s_{k+1}}$.

Let us denote the BCP that emerges in this way from a BCP \mathcal{P} by $\mathcal{T}_k(\mathcal{P})$. Clearly, $\mathcal{T}_k(\mathcal{P})$ is not tree structured at all. However, it holds:

Theorem 3. *Given a constant k and a BCP \mathcal{P} with tree-width lower or equal k , the support encoding of $\mathcal{T}_k(\mathcal{P})$ as an integer program is polynomial in the size of \mathcal{P} and it has the same value as its linear continuous relaxation.*

Proof. We consider the following integer programs and their linear continuous relaxations: IP_T , the integer programming formulation based on the saturated tree-decomposition from Theorem 2; LP_T , the linear continuous relaxation of IP_T ; IP_F , the support encoding of $\mathcal{T}_k(\mathcal{P})$ as an integer program; and LP_F , the linear continuous relaxation of the latter. By abuse of language, we identify the optimal value of the objective with the name of a problem. According to Theorem 1, it holds that $\text{IP}_T = \text{LP}_T$. Furthermore, we observe that LP_F is, in some sense, a lifted version of LP_T : LP_F operates on a super-set of variables of LP_T , but extra variables present in LP_F yield no additional profit in the objective function, and all original constraints are still present. Consequently, LP_F just contains some extra constraints, and it holds $\text{LP}_T \geq \text{LP}_F$. The same relation also holds for IP_F and IP_T . However, since all extra constraints present in IP_F are redundant for any integer solution in IP_T , we even have that $\text{IP}_F = \text{IP}_T$. But then: $\text{LP}_T \geq \text{LP}_F \geq \text{IP}_F = \text{IP}_T = \text{LP}_T$. And thus, $\text{LP}_F = \text{IP}_F$. \square

It is interesting to observe the analogies of our method to the work from Bienstock and Ozbay in [4]. They show that the polytope of packing problems whose matrices have a clique-graph with bounded tree-width can be described perfectly by adding a polynomial number (exponential in the tree-width) of Sherali-Adams variables and cuts [16]. This work does not apply directly to the matrices that we encounter (even when we ignore negative matrix entries, note that the size of BCP domains is usually in the same order as the number

of variables which causes the clique-graphs to have large tree-width). However, it is interesting to see that in both their work and in our approach the introduction of variables that model subsets of the original variables leads to the desired result.

6 Numerical Results for Augmented BCPs with a Linear Objective

In [2], we showed that LP-inference is not worthwhile for pure BCPs. Here, we will experiment with BCPs that are augmented by some linear constraints and a linear objective function. For this purpose, we consider the following multi-knapsack problem: Given m knapsacks with capacities C_1, \dots, C_m and n items with knapsack-dependent weights $w_{1,1}, \dots, w_{m,n} \geq 0$, we have to place each item in exactly one knapsack such that the sum of the weights of the items placed in each knapsack does not exceed its capacity. Items achieve knapsack-dependent profits $p_{1,1}, \dots, p_{m,n} \geq 0$, and we try to maximize the total profit $\sum_{k=1}^m \sum_{i \text{ placed in } k} p_{k,i}$. We augment this problem by adding binary constraints that each forbid some specific simultaneous placements of two items. For instance, a constraint over items i and j could require that i and j cannot be placed in the same bin. Or that they must be placed in the same bin. Or generally, that placing item i in knapsack k_1 and placing item j in knapsack k_2 is not allowed for a set of tuples (k_1, k_2) . For our experiments, we compare the following models.

$$\begin{aligned}
 \text{(CP)} : \max \sum_{i=1}^n p_{x_i,i} \\
 (1) \quad & \sum_{i \leq n, x_i=k} w_{k,i} \leq C_k & \forall 1 \leq k \leq m \\
 (2) \quad & F_p(x_{p_1}, x_{p_2}) = \text{true} & \forall 1 \leq p \leq q \\
 (3) \quad & x_i \in \{1, \dots, m\} & \forall 1 \leq i \leq n
 \end{aligned}$$

where F_1, \dots, F_q denote the binary constraints and constraint F_p limits the simultaneous placement of items p_1 and p_2 . The next model is based on the traditional LP formulation:

$$\begin{aligned}
 \text{(LP}_T\text{)} : \sum_{k=1}^m \sum_{i=1}^n p_{k,i} y_{k,i} \\
 (1) \quad & \sum_{i \leq n} w_{k,i} y_{k,i} \leq C_k & \forall 1 \leq k \leq m \\
 (2) \quad & y_{k,p_1} + y_{l,p_2} \leq 1 & \forall 1 \leq p \leq q, F_p(k,l) = \text{false} \\
 (3) \quad & \sum_k y_{k,i} = 1 & \forall 1 \leq i \leq n \\
 (4) \quad & y_{k,i} \in \{0, 1\} & \forall 1 \leq k \leq m, 1 \leq i \leq n
 \end{aligned}$$

Our last model is based on the support LP formulation:

$$\begin{aligned}
 \text{(LP}_S\text{)} : \sum_{k=1}^m \sum_{i=1}^n p_{k,i} y_{k,i} \\
 (1) \quad & \sum_{i \leq n} w_{k,i} y_{k,i} \leq C_k & \forall 1 \leq k \leq m \\
 (2a) \quad & y_{k,p_1} \leq \sum_{l \mid F_p(k,l)=\text{true}} y_{l,p_2} & \forall 1 \leq p \leq q, 1 \leq k \leq m \\
 (2b) \quad & y_{l,p_2} \leq \sum_{k \mid F_p(k,l)=\text{true}} y_{k,p_1} & \forall 1 \leq p \leq q, 1 \leq l \leq m \\
 (3) \quad & \sum_k y_{k,i} = 1 & \forall 1 \leq i \leq n \\
 (4) \quad & y_{k,i} \in \{0, 1\} & \forall 1 \leq k \leq m, 1 \leq i \leq n
 \end{aligned}$$

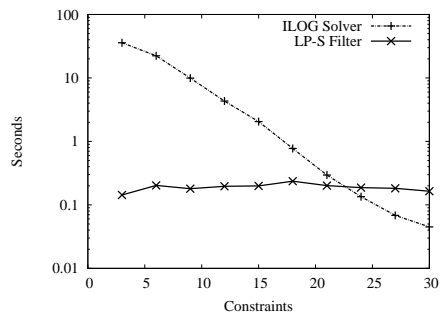


Fig. 1. Comparison of the average time [sec, log-scale] over 100 random instances to solve 5-knapsack instances with 12 items and varying number of binary constraints with a pure CP approach (Ilog Solver) and our LP-filtering approach with support formulation.

In combination with the latter two models, we use two different methods for exploiting the corresponding LP relaxation. The first is called *LP-pruning* which uses it for pruning purposes (i.e. the early termination of search) only. The second is inspired by [11] and called *LP-filtering*: after computing the LP-solution to the problem, it chooses those assignments $X_i = k$ for which the continuous value of y_{ik} is lower than some threshold value $\varepsilon > 0$. Then, for each of the selected assignments, it sets up a new LP with the objective to maximize y_{ik} . If the relaxation gives a value lower than 1, k can be removed from D_i .

We generate random instances for given parameters m , n , and q by drawing weights $w_{k,i}$ uniformly at random between 1 and 100. Knapsack capacities are then set to $C_k := 2 \frac{\sum_i w_{k,i}}{m}$. The profits $p_{k,i}$ were weakly correlated with the weights by drawing them uniformly at random in the intervals $[w_{k,i} - 5, w_{k,i} + 5]$. Binary constraints are generated randomly, whereby the number of allowed pairs of each constraint is set to $m^2/2$.

Figure 1 shows a comparison of a pure CP approach and our LP-filtering approach for random 5-knapsack instances with 12 items. We can see clearly how essential the use of a global bound is: even on this toy-example, in the low-constrained region solver needs, on average, more than 35 seconds while the LP-filtering approach takes less than half a second. We also started runs on instances with 16 items, but the pure CP-approach took so much time that we had to cancel the experiment.

Regarding the effect of the support model and the traditional formulation of the binary constraints, in Figure 2 we compare the number of choice points visited by LP-pruning and filtering when based on the support or traditional formulation. We observe what was to be expected: LP filtering visits fewer choice

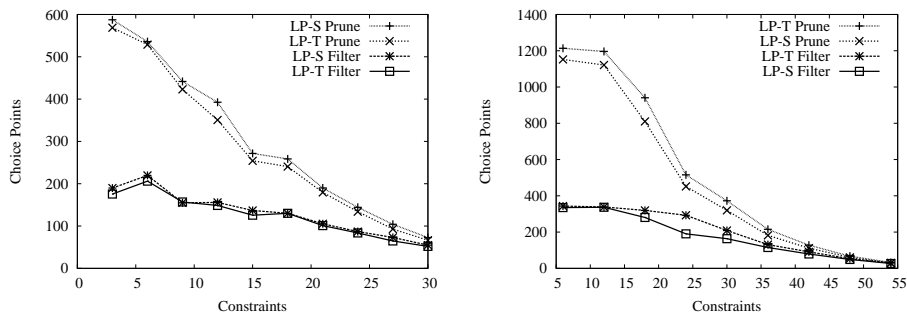


Fig. 2. Comparison of the average number of choice points over 100 random instances to solve 5-knapsack instances with 12 and 16 items and varying number of binary constraints.

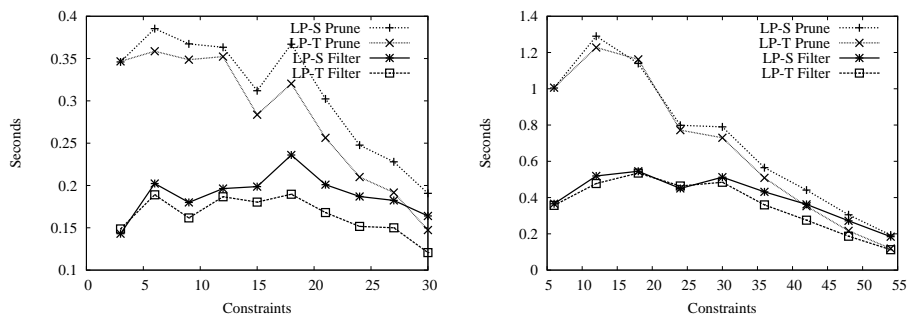


Fig. 3. Comparison of the average time over 100 random instances needed to solve 5-knapsack instances with 12 and 16 items and varying number of binary constraints.

points than LP-pruning, and for both pruning and filtering, the support formulation is stronger and results in smaller search trees than the traditional one.

Of course, LP-filtering incurs larger computational costs per choicepoint than LP-pruning. Whether or not the additional time needed to perform stronger inference will in general depend on the application. Figure 3 shows that LP-based filtering beats the approach that uses the LP-bound for pruning only. We observe that, on small multi-knapsack instances, using the support formulation does not pay off. This can be attributed to the fact that the traditional formulation leads to much sparser matrices and can therefore be solved much faster, which makes up for slightly worse relaxation values. Of course, as problem sizes and search spaces grow, the use of a stronger bound becomes more and more attractive as the larger costs per choice point can be paid for by a much larger reduction in search costs. Consequently, in Figure 4, we see that already on moderately larger problem instances filtering based on the support LP-formulation becomes the method of choice.

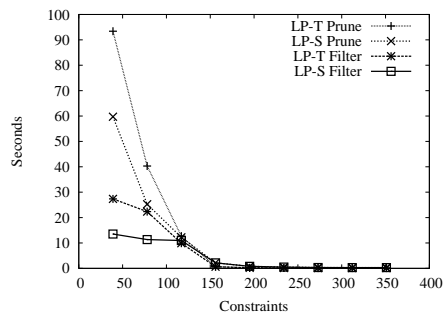


Fig. 4. Comparison of the average time over 100 random instances to solve 5-knapsack instances with 40 items and varying number of binary constraints.

7 Summary and Future Work

We have shown how to efficiently transform binary constraint problems into linear programs whose extreme points are integer whenever the tree-width of the initial BCP is bounded by some constant k .

Some questions remain open. When making the step from tree-structured BCPs to those with tree-width 2, the exponent in the number of CP variables jumps from 1 to 3. Can the exponents be controlled better than we did it here? Are there efficient ways to generate constraints in a lazy fashion, adding them only when the linear relaxation turns out to be fractional? What about linear programming polytopes of other islands of tractability, such as Horn formulas?

Moreover, our approach motivates a procedure for lifting arbitrary binary integer programs $\{x \in \{0, 1\}^n \mid Ax \leq b\}$ that would be interesting to compare with the Sherali-Adams procedure.² To simplify the notation, all sets considered in the following are subsets of $\{1, \dots, n\}$, and we write $X + Y$ for the union of sets X, Y that are disjoint. At level $k \geq 1$, to the original problem we add variables $w(Y, N) \geq 0$ for all $|Y + N| = k$ (with the idea that $x_j = 1$ for all $j \in Y$ and $x_h = 0$ for all $h \in N$). Moreover, we add the following sets of constraints:

- $\sum_{Y+N=S} w(Y, N) = 1$ for all $|S| = k$.
- $w(Y, N) \leq x_j$ and $w(Y, N) \leq x_h$ for all $|Y + N| = k, j \in Y, h \in N$.
- $x_j \leq \sum_{Y+N=S, j \in Y} w(Y, N)$ and $1 - x_j \leq \sum_{Y+N=S, j \in N} w(Y, N)$ for all $|S| = k, j \in S$.

² Whereby it is important to state explicitly that the lifting procedure that we sketch here, due to potentially large CP variable domains, would not have given the desired results when applied to S_{IP} . We really needed to exploit the original structure of the given BCP with bounded tree-width to achieve a good IP model.

- $w(X+Y, M+N) \leq \sum_{Z+O=S} w(X+Z, M+O)$ for all $1 \leq |X+M| < k$, $|S| = |Y+N|$, $S \cap (Y+N) = \emptyset$, and $|X+Y+M+N| = k$.
- $(\sum_{j \in Y} a_{ij} + \sum_{j \notin Y+N, a_{ij} < 0} a_{ij} - b_i)w(Y, N) \leq 0$ for all $|Y+N| = k$.

Note that the variables that we add are semantically the same as the ones that are added by Sherali and Adams. However, the way we post the constraints is quite different so that it suffices to add variables for subsets of size equal to k only. As the results in this paper show, at level n , this lifting method gives a formulation that has integer extreme points only.

References

1. R.K. Ahuja, T.L. Magnati, J.B. Orlin. *Network Flows*. Prentice Hall, 1993.
2. I.D. Aron, D.H. Leventhal, M. Sellmann. A Totally Unimodular Description of the Consistent Value Polytope for Binary Constraint Programming. *CPAIOR*, LNCS 3990:16–28, 2006.
3. D. Bienstock. Approximate formulations for 0-1 knapsack sets. *CORC Report TR-2006-03*, Columbia University, 2006.
4. D. Bienstock and N. Ozby. Tree-width and the Sherali-Adams operator. citeseer.ist.psu.edu/bienstock03treewidth.html, 2003.
5. H.L. Bodlaender. A Linear Time Algorithm for Finding Tree-decompositions of Small Treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
6. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.
7. E.C. Freuder. Complexity of k-tree structured constraint satisfaction problems. *AAAI*, pp. 4–9, 1990.
8. I.P. Gent. Arc Consistency in SAT. *ECAI*, pp. 121–125, 2002.
9. J.N. Hooker. A hybrid method for planning and scheduling. *CP*, LNCS 3258:305–316, 2004.
10. S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.
11. M.O.I. Khemmoudj, H. Bennaceur, A. Nagih. Combining Arc-Consistency and Dual Lagrangean Relaxation for Filtering CSPs. *CPAIOR*, LNCS 3524:258–272, 2005.
12. R. Marinescu and R. Dechter. AND/OR Branch-and-Bound Search for 0-1 Integer Linear Programming. *CPAIOR*, LNCS 3990:152–166, 2006.
13. M. Milano. *Integration of Mathematical Programming and Constraint Programming for Combinatorial Optimization Problems, Tutorial at CP*, 2000.
14. G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.
15. N. Robertson and P.D. Seymour. Graph minors - Algorithmic aspects of treewidth. *Algorithms*, 7:309–322, 1986.
16. H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3:411–430, 1990.
17. M. Van Vyve, L.A. Wolsey. Approximate extended formulations. *Mathematical Programming*, 105(2–3):501–522, 2006.