Clique Cuts in Weighted Constraint Satisfaction

Simon de Givry, George Katsirelos

Clique Cuts in Weighted Constraint Satisfaction

Simon de Givry and George Katsirelos

MIAT, UR-875, INRA, F-31320 Castanet Tolosan, France
simon.de-givry@inra.fr, gkatsi@gmail.com

Abstract. In integer programming, cut generation is crucial for improving the tightness of the linear relaxation of the problem. This is relevant for weighted constraint satisfaction problems (WCSPs) in which we use approximate dual feasible solutions to produce lower bounds during search. Here, we investigate using one class of cuts in WCSP: clique cuts. We show that clique cuts are likely to trigger suboptimal behavior in the specialized algorithms that are used in WCSP for generating dual bounds and show how these problems can be corrected. At the same time, the additional structure present in WCSP allows us to slightly generalize these cuts. Finally, we show that cliques exist in instances from several benchmark families and that exploiting them can lead to substantial performance improvement.

1 Introduction

The performance of branch and bound algorithms depends crucially on the quality of the dual bound produced during search. One of the techniques used in Integer Linear Programming (ILP) to improve dual bounds is cut generation. These work by adding to the LP relaxation constraints that are entailed by the integer program but not by the linear program. These eliminate optimal non-integral solutions and hence improve the dual bounds. One such class of cuts is clique cuts [4]. These are quite powerful, as they strengthen the inference possible in the LP from $\sum_{i=1}^{n} x_i \geq n/2$ to $\sum_{i=1}^{n} x_i \geq n-1$, so when they can be found, they can increase performance significantly.

In weighted constraint satisfaction, adding cuts remains under explored so far. Instead, research has focused on other local techniques such as on-the-fly variable elimination [24], soft arc consistency [25,16,8,9,28,27,31], dominance detection [26,14], or global techniques like mini-bucket elimination [21] and tree-decomposition based search [20,29,15,12,1].

These techniques are useful, but they are orthogonal to cut generation and can be further improved by it. However, adding cuts to dual bound reasoning in WCSP presents several challenges. The bounds used in WCSP are based on producing feasible — but often suboptimal — dual solutions of the LP relaxation. The algorithms that do so are weaker than LP solvers, but are significantly faster. In some extreme cases, a WCSP solver can solve an instance to optimality by search on several thousand nodes in shorter time than it would take to solve the linear relaxation of that instance. This speed comes at a price, as it is not

easy to extend these algorithms to handle arbitrary linear constraints and to do so efficiently. In fact, we show that clique cuts reveal the worst cases for these bounds and, if used without care, may lead to no improvement whatsoever in the dual bound. We develop, however, a set of techniques and heuristics that prove successful in applying clique cuts to some families of instances. These families were cases where WCSP solving was much worse than applying ILP. With our contributions, the performance gap is closed significantly.

2 Background

Integer Linear Programming. An ILP has the general form

$$\begin{aligned} &\min c^T x \\ s.t. &Ax \geq b \\ &x \geq 0 \\ and & \\ &x \in Z^n \end{aligned}$$

where c, b are constant vectors, x is a vector of variables and A is a constant matrix, and all entries in c, b and A are integral. If we ignore the integrality constraint, the resulting problem is the linear relaxation of the original ILP. As a relaxation, any dual bound of the LP is a dual bound of the ILP.

The dual of an LP as given above is

$$\begin{aligned} &\max b^T y \\ s.t. &A^T y \leq c \\ &y \geq 0 \end{aligned}$$

Any feasible solution of the dual gives a lower bound of the primal and the optima meet.

Solving an ILP exactly is typically done using branch and bound. At each node, the linear relaxation is solved and if the lower bound is greater than the cost of the incumbent, the node is closed. The linear relaxation can be strengthened using cuts, i.e., linear inequalities that are not entailed by the LP relaxation, but are entailed by the ILP. All linear combinations of inequalities are entailed by the LP. A particular method of deriving inequalities that are not entailed by the LP is strengthening, which derives $\sum a_i x_i \geq \lceil b \rceil$ from $\sum a_i x_i \geq b$ when b is not integral. A special case that we use here is deriving $\sum a_i x_i \geq \lceil b/c \rceil$ from $\sum c a_i x_i \geq b$.

Weighted CSP . A WCSP is a tuple $\langle \mathcal{X}, \mathcal{C}, k \rangle$ where $\mathcal{X}$ is a set of discrete variables with associated domain $D(X)$ for each variable $X \in \mathcal{X}$, $\mathcal{C}$ is a set of cost functions

with associated scope $scope(c) \subseteq \mathcal{X}$ for each $c \in \mathcal{C}$ and k is a distinguished "top" cost. For simplicity, we assume a single cost function per scope and write c_S for the unique function with scope S. We also write c_i for the unary cost function with scope $\{X_i\}$ and c_{ij} for the binary cost function with scope $\{X_i, X_j\}$. A partial assignment τ is a mapping from each of a subset of $\mathcal{X}$ (denoted $scope(\tau)$) to one of the values in its domain. A complete assignment is a partial assignment such that $scope(\tau) = \mathcal{X}$. We write $\tau(S)$ for all partial assignments such that $scope(\tau) = scope(S)$. Given a cost function c_S and a partial assignment τ such that $S \subseteq scope(\tau)$, $c(\tau|_S) \geq 0$ gives the cost of τ for c. If $c(\tau|_S) = k$, we say that the constraint is violated. Given a complete assignment τ, its cost is $c(\tau) = \sum_{c_S \in \mathcal{C}} c(\tau|_S)$. The objective of solving a WCSP is to find an assignment that minimizes $c(\tau)$. It defines an NP-hard problem.

Dual (lower) bounds in WCSP are derived by applying equivalence preserving transformations (EPTs). Given two cost functions c_{S_1}, c_{S_2} with $S_1 \subset S_2$, a partial assignment $\tau \in \tau(S_1)$ and a value α, we can move cost α between c_{S_1} and c_{S_2} by updating $c_{S_1}(\tau) \leftarrow c_{S_1}(\tau) + \alpha$ and $c_{S_2}(\tau') \leftarrow c_{S_2}(\tau') - \alpha$ for all $\tau' \in \tau(S_2)$ such that $\tau'|_{S_1} = \tau$ if this operation leaves no negative costs. This operation preserves the global cost of all assignments, so it is called an EPT. If α is positive, this operation is called a projection, otherwise it is an extension. They are written, respectively $project(c_{S_2}, c_{S_1}, \tau, \alpha)$ and $extend(c_{S_1}, \tau, c_{S_2}, -\alpha)$. For convenience, it is usual to assume the presence of a cost function with nullary scope, $c_\varnothing$. Since this function has nullary scope, its cost is used in every assignment of the WCSP and its value is a lower bound for the cost of any assignment of the WCSP (remember that all costs are non-negative). For any cost function c with non-empty scope, if $\alpha = \min_{scope(\tau)=scope(c)} c(\tau)$, we can apply $project(c, c_\varnothing, \emptyset, \alpha)$.

The *extend* and *project* operations preserve equivalence even if α is chosen so that the operation creates negative costs. But it would violate the requirement that all costs are non-negative. This implies that a given EPT may be inadmissible because it creates negative costs, but is admissible as part of a set of EPTs, such that if they are all applied no negative costs remain.

Soft consistencies and linear programming. There has been a long sequence of algorithms for computing a sequence of EPTs. For the results from the WCSP literature, an overview is given by Cooper et al [9]. A parallel development of algorithms has happened under the name maximum a posteriori (MAP) inference for Markov random fields (MRF) [39], starting with [36]. In most of the literature, the extension and projection operations are limited so that one of the cost functions involved has arity 1. We will also mostly limit our attention to this case here. In any case, it has been shown that all these algorithms find a feasible dual solution of the following linear relaxation of a WCSP/MRF:

$$\min \sum_{c_S \in \mathcal{C}, \tau \in \tau(S)} c_S(\tau) * y_\tau$$

s.t.

$$y_\tau = \sum_{\tau' \in \tau(S_2), \tau'|_{S_1} = \tau} y_{\tau'} \qquad \forall c_{S_1}, c_{S_2} \in \mathcal{C} \mid S_1 \subset S_2, \tau \in \tau(S_1), |S_1| \geq 1$$

$$\sum_{\tau \in \tau(S)} y_\tau = 1 \qquad \forall c_S \in \mathcal{C}, |S| \geq 1$$

However, specialized algorithms aim to be faster than using a linear solver on the above problem. But solving this special form is as hard as solving an arbitrary LP [32], hence these specialized algorithms typically converge on suboptimal dual solutions. One particular condition to which some algorithms converge is virtual arc consistency:

Definition 1. *Given a WCSP C, let $hard(C)$ be the CSP which has the same set of variables and domains as C and for each $c_S \in \mathcal{C}$ has a hard constraint $hard(c_S)$, which is satisfied by an assignment τ if and only if $c_S(\tau) = 0$. C is virtually arc consistent if and only if $hard(C)$ is arc consistent.*

Among those algorithms that converge on virtual arc consistency is, not so surprisingly, VAC [9], to which we refer further in this paper.

Finally, we define EDAC, which is the default level of soft consistency enforced in the ToulBar2 solver.

Definition 2. *A WCSP C is node consistent (NC) if for every cost function $c \in \mathcal{C}$ with $|scope(c)| = 1$, there exists $\tau \in \tau(c)$ with $c(\tau) = 0$ and for all $\tau \in \tau(c)$, $c_\varnothing + c(\tau) < k$*

Definition 3. *A WCSP C is generalized arc consistent (GAC) if for all $c_S \in \mathcal{C}, |S| > 1$, $\forall \tau \in \tau(S), c_S(\tau) = k$ if $c_\varnothing + c_S(\tau) + \sum_{X_i \in S} c_i(\tau|_{\{i\}}) = k$ and for all $X_i \in S, \forall v \in D(X_i)$, $\exists \tau \in \tau(S)$ such that $\tau_{\{i\}} = v$ and $c_S(\tau) = 0$.*

Definition 4. *A binary WCSP C is existential arc consistent (EAC) if there exists a value $u \in D(X_i)$ for each $X_i \in \mathcal{X}$ such that $c_i(u) = 0$ and $\forall c_{ij} \in \mathcal{C}, \exists v \in D(X_j)$ s.t. $c_{ij}(u, v) + c_j(v) = 0$. It is existential directional arc consistent (EDAC) if it is NC, GAC, EAC, and directional arc consistent (DAC),* i.e., *$\forall c_{ij} \in \mathcal{C}, i < j, \forall u \in D(X_i), \exists v \in D(X_j)$ s.t. $c_{ij}(u, v) + c_j(v) = 0$.*

EDAC has been extended to ternary cost functions in [35]. Different generalizations for arbitrary arities have been proposed in [28,27,2].

3 Clique Cuts

An important class of cuts used by MIP solvers are *clique* cuts [4]. Given a set S of 0/1 variables and the constraints

$$x_i + x_j \geq 1 \qquad\qquad \forall x_i, x_j \in S, i \neq j$$

we can derive

$$\sum_{x_i \in S} x_i \geq |S| - 1 \tag{1}$$

This is done as follows. Given any triplet of distinct $x_i, x_j, x_k \in S$, we sum the binary constraints involving these three variables to get $2x_i + 2x_j + 2x_k \geq 3 \equiv x_i + x_j + x_k \geq 3/2$, which we strengthen to $x_i + x_j + x_k \geq 2$. We repeat this with $m-$tuples of variables and the $(m-1)$-ary constraints from the previous step to generate $m-$ary constraints until $m = |S|$, which gives the above constraint. This is in contrast to the much weaker constraint which can be derived by linear combinations only (that is, the constraint that is entailed by the LP), which is $\sum_{x_i \in S} x_i \geq |S|/2$.

The reason for the name of these cuts comes from the fact that the set S corresponds to a clique in a graphical representation of binary constraints of the IP. Specifically, we construct a graph with a vertex for each 0/1 variable of the IP and have edges between two vertices if their variables appear together in a binary constraint of the form $x + y \geq 1$. From every clique in this graph we can derive a clique cut, i.e., a constraint of the form (1).

We can generalize this construction to have a vertex also for $1 - x$ (equivalently $\overline{x}$) for every 0/1 variable x of the IP, connected to the vertex x. Then, there exists an edge between x and $\overline{y}$ for every constraint $x + (1-y) \geq 1 \equiv x - y \geq 0$ and an edge between $\overline{x}$ and $\overline{y}$ for every constraint $(1-x) + (1-y) \geq 1 \equiv -x - y \geq -1$. In this case if we find a clique that contains both x and $\overline{x}$, the clique cut requires that we must set all other variables in the clique to 1:

$$\begin{aligned} x + (1-x) + \sum_{y_i \in S} y_i &\geq |S| + 1 \Rightarrow \\ 1 + \sum_{y_i \in S} y_i &\geq |S| + 1 \Rightarrow \\ \sum_{y_i \in S} y_i &\geq |S| \Rightarrow \\ y_i &= 1 \qquad\qquad \forall y_i \in S \end{aligned}$$

If we find a clique that contains $x, \overline{x}, y, \overline{y}$, the problem is unsatisfiable.

3.1 Cliques in WCSPs

We apply this reasoning to get clique cuts in WCSPs. From a WCSP P with top k, we construct a graph $G(P)$ as follows: we have a vertex for v_{xi} every variable

x and every value $i \in D(x)$. There exists an edge between two vertices $v_{pv(p)}$, $v_{qv(q)}$ if there exists a cost function c_{pq} such that $c_{pq}(v(p), v(q)) = k$ [1]. This corresponds to the constraint $(1 - x_{pv(p)}) + (1 - x_{qv(q)}) \geq 1$ in the LP relaxation.

Now, given a clique S in $G(P)$, we can add the clique constraint $\sum_{v_{pv(p)} \in S}(1 - x_{pv(p)}) \geq |S| - 1$. In other words, the constraint requires at least $n - 1$ variables in the clique must get a value other than the one included in the clique.

Overlapping cliques. In the ILP case, we mentioned that the graph construction can be generalized to have vertices for both x and $(1 - x)$ and an edge between them. In the WCSP case we already have vertices for different values of the same variable. We can ensure that each set of vertices corresponding to values of the same variable forms a clique. Then, the general case for a clique is that it may contain several values from each variable involved. We assume in the rest of this paper that this is the case. Then, given such a clique S, $varsof(S) = \{X_p \mid \exists i.v_{pi} \in S\}$. For every $X_p \in varsof(S)$ we write $S(p) = \{i \mid v_{pi} \in S\}$. The constraint then requires that at least $|varsof(S)| - 1$ of the variables get a value outside their respective set $S(p)$.

Despite this generalization, in order to simplify presentation, we will assume that every variable in the clique is a binary variable and $S(p) = \{1\}$. All results and algorithms are valid for the general case, with the caveat that when we use $c_i(0)$ $(c_i(1))$ on the right hand side of an expression, it means $\min_{i \notin S(i)} c_i(v)$ $(\min_{i \in S(i)} c_i(v))$ and when it appears on the left hand side, it means for all $v \notin S(i), c_i(v)$ $(v \in S(i), c_i(v))$.

3.2 Propagating clique constraints

We can encode a clique constraint with a non-uniform layered automaton (meaning an automaton where the transition function may differ in each layer) with two states q_0, q_1. The initial state is q_0 and both are accepting. Suppose $|S| = n$ and the variables involved are $x_1, \ldots, x_n$. Then the transition function at layer i is

Q_{i-1}	X_i	Q_i
q_0	$j \notin S(i)$	q_0
q_0	$j \in S(i)$	q_1
q_1	$j \notin S(i)$	q_1
q_1	$j \in S(i)$	*not allowed*

We can encode the automaton using the usual ternary construction [33,34]. This ensures that EDAC deduces the optimal lower bound for each clique constraint, at least when viewed in isolation. As the constraint is invariant under reordering of the variables, we can use an ordering that agrees with the EDAC ordering and place each state variable Q_i in between the variables x_i and x_{i+1} in

[1] This is the micro-structure of the WCSP, restricted only to binary tuples with infinite cost

the EDAC ordering. This is sufficient to guarantee that EDAC propagates each clique constraint optimally [2].

However, this is not an attractive option in practice. The reformulation to automata introduces many variables for each clique constraint, something to which EDAC is quite sensitive. That is, depending on the vagaries of the algorithm, such as the order in which constraints of a variable are processed, it may derive a stronger or weaker lower bound, even though the formulation is locally optimal.

Combined with the issues that we describe later, this leads us to implement a specialized propagator for these constraints, which includes the reasoning performed by the softregular constraint [18] on this automaton. For completeness, we describe the propagation independent of the softregular constraint. This also sets the stage for the discussion of section 3.3.

Propagating clique constraints. For each clique constraint clq, we store a single integer a_0 which summarizes the effect on the constraint of all extensions and projections we have performed. This quantity is the cost of assigning all variables to 0.

We define the following transformation, $u \to clq$ which performs a set of extensions and projections through the clique constraint, getting the maximum increase in $c_\varnothing$ and extending the minimum amount from unary costs in order to achieve this increase.

Let clq be the clique constraint, $l_0 = \sum c_i(0) - \max\{c_i(0)\}$, $r_0 = \max(\{c_i(0)\} \setminus \max\{c_i(0)\})$ (the second largest $c_i(0)$, possibly equal to the largest), $t = \sum c_i(0)$, $t_r = l_0 + r_0$ and $l_1 = \min\{t_r - l_0, \min c_i(1), a_0\}$. Then $u \to clq$ comprises the following operations:

- If the arity is 2 (resp. 1), project a_0 to the $(0, 0)$ tuple (resp. 0 value) of the corresponding binary (resp. unary) cost function. Otherwise:
- Add $l_0 + l_1$ to $c_\varnothing$
- Set each unary cost $c_i(0)$ to $\max(0, c_i(0) - r_0)$
- Add $t_r - l_0 - l_1$ to a_0 (possibly reducing a_0)
- Add $t_r - l_0 - l_1 - \min(c_i(0), r_0)$ to $c_i(1)$ for all i

Proposition 1. *$u \to clq$ is an EPT*

Proof. We ignore here any binary costs. Since these are left untouched, they would contribute the same cost to any assignment before and after the EPT.

Suppose $n - 1$ variables are assigned 0 and that $x_i = 1$. The cost of this assignment is $c_i(1) + \sum_{j \neq i} c_j(0)$. In the reformulated problem, the cost of the assignment is $l_0 + l_1$ from $c_\varnothing$, $c_i(1) + t_r - l_0 - l_1 - \min(c_i(0), r_0)$ from the unary cost of $x_i = 1$, and $\sum_{j \neq i} \max(0, c_j(0) - r_0)$ from the unary costs of $x_j = 0$ for $j \neq i$. This sums to $c_i(1) + t_r - \min(c_i(0), r_0) + \sum_{j \neq i} \max(0, c_j(0) - r_0)$.

If $\max c_i(0) = r_0$, $\min(c_i(0), r_0)$ simplifies to $c_i(0)$, $\max(0, c_j(0) - r_0)$ and $t_r = t$. Then the above sum is $c_i(1) + t - c_i(0)$ which is $c_i(1) + \sum_{j \neq i} c_i(0)$, as in the original problem.

If $\max c_i(0) > r_0$, there exists a unique x_k for which $c_k(0) = \max c_i(0)$. Then $\min(c_i(0), r_0)$ simplifies to $c_i(0)$ for all $i \neq k$ and to r_0 for $i = k$, while $\sum_{j \neq i} \max(0, c_j(0) - r_0)$ is 0 if $i = k$ and $c_k(0) - r_0$ if $i \neq k$. So if $i = k$ the sum is $c_i(1) + t_r - r_0 = c_i(1) + l_0 = c_i(1) + \sum_{j \neq i} c_j(0)$, as in the original problem. If $i \neq k$, the sum is $c_1(1) - c_i(0) + t_r + c_k(0) - r_0 = c_1(1) + l_0 + c_k(0) - c_i(0)$. As l_0 is the sum of all $c_j(0)$ except for $c_k(0)$, that works out to $c_1(1) + \sum_{j \neq i} c_i(0)$, equivalent to the cost in the original problem.

Finally, assume all n variables are assigned 0, the cost of the assignment is $t = \sum_i c_i(0) = l_0 + l_1 + t - l_0 - l_1 = c_\varnothing + a_0 + (t - t_r)$. We have $t - t_r = \max c_i(0) - r_0$, which is exactly the cost left in $c_k(0)$ if $\max c_i(0) > r_0$ and 0 otherwise. In either case the cost remains the same before and after the transformation, as long as a_0 is later projected to $c_\varnothing$. As all variables are assigned 0, the cost a_0 is indeed used by projecting to a lower arity cost function, so the cost in the reformulated problem is the same as in the original problem.

In general, we apply $u \rightarrow clq$ after node consistency has been enforced. Therefore, for each variable, either $c_i(0) = 0$ or $c_i(1) = 0$ and so if $\min c_i(1) > 0$, it has to be that $t - l_0 = \max c_i(0) = 0$. This means that either l_0 or l_1 will be non-zero. But unary costs change non-monotonically as other constraints move costs through the variable of the clique, so it is possible for both to occur at different times in the lifetime of the same constraint.

3.3 Issues with Virtual Arc Consistency.

Using clique constraints to improve lower bounds computed by VAC or any algorithm that converges to an arc consistent state is problematic. This includes algorithms such as MPLP [17,38,37], TRW-S [23] and other algorithms from the MRF community. We will show that the problem is that all these algorithms can only improve the lower bound by *sequences* of EPTs, but it is required to use *sets* of EPTs to fully exploit clique constraints.

As a hard constraint, the clique constraint is redundant, not only logically, but also in terms of propagation. Indeed, suppose a clique S contains values from three variables x, y and z, that the pairwise constraints are arc consistent, and let $d \in D(y)$ such that $v_{yd} \in S$ and t_{xy}, t_{yz} be the supports of $y = d$ in the corresponding binary constraints. Since $t_{xy}[x]$ and $t_{yz}[z]$ are values that may not be part of the clique, they are consistent with each other in the clique constraint and hence $t_{xyz} = t_{xy} \cup t_{yz}$ is a support for all three variables simultaneously. We can extend this reasoning to an arbitrary number of variables, hence we can get global supports from pairwise supports.

The fact that clique constraints add no propagation strength to $hard(C)$ means that adding clique constraints after VAC propagation will have no effect. Indeed, VAC can only improve the lower bound as long as $hard(C)$ is arc inconsistent. Since the clique constraint is propagation redundant, if $hard(C)$ is arc consistent, it will remain so after adding the clique constraint.

Empirically, we have observed the same is often true after EDAC, even though it does not necessarily converge to a virtually arc consistent state. Moreover, even

if the clique constraint exists before we enforce VAC, the fact that VAC does not have a unique fixpoint means that we cannot predict whether the fixpoint that it does reach will use the clique constraint optimally. Hence, we need to devise a method to exploit clique constraints in a virtually arc consistent problem.

Example 1. To begin, consider what happens in a problem with 3 Boolean variables in a clique, where all costs are 0 except $c_x(0) = c_y(0) = c_z(0) = 1$ and $c_{xy}(1,1) = c_{yz}(1,1) = c_{xz}(1,1) = k$. The following series of EPTs makes the problem VAC with $c_\varnothing = 3/2$:

1. $extend(x, 0, xy, 1/2)$
2. $extend(x, 0, xz, 1/2)$
3. $extend(z, 0, yz, 1/2)$
4. $project(xy, y, 1, 1/2)$
5. $project(xz, z, 1, 1/2)$
6. $project(yz, y, 1, 1/2)$
7. $project0(y, 1)$
8. $project0(z, 1/2)$

The reformulated problem has $c_{xy}(1,1) = c_{yz}(1,1) = c_{xz}(1,1) = k$ and $c_{xy}(0,0) = c_{yz}(0,0) = c_{xz}(0,0) = 1/2$. Note that actually running VAC would perform a more convoluted series of moves: in the first iteration, a conflict involves only two variables, say x and y. That allows it to move 1 unit of cost from $x = 0$ to $y = 1$, and to project cost 1 to $c_\varnothing$. In the next iteration, the conflict involves all 3 variables and it moves $1/2$ a unit of cost from $z = 0$ to $x = 1$ and from there to $y = 0$ and another $1/2$ a unit directly from $z = 0$ to $y = 1$, projecting another $1/2$ to $c_\varnothing$.

Since this reformulated problem is VAC, adding a clique constraint will do nothing, but solving the linear relaxation detects that we can improve the lower bound by the following set of operations[2]:

1. $extend(y, 1, xy, 1/2)$. After this, $c_y(1) = -1/2$
2. $extend(z, 1, xz, 1/2)$, $extend(z, 1, yz, 1/2)$. After these, $c_z(1) = -1$
3. $project(xy, x, 0, 1/2)$
4. $project(xz, x, 0, 1/2)$
5. $project(yz, y, 0, 1/2)$
6. $extend(x, 0, clq, 1)$
7. $extend(y, 0, clq, 1/2)$
8. $project0(clq, 1/2)$
9. $project(clq, y, 1, 1/2)$
10. $project(clq, z, 1, 1)$

This leaves the problem with lower bound increased to 2 and the only non-zero costs are the hard tuples, which are unchanged throughout, and the tuple $c_{clq}(0,0,0) = 1$. □

[2] In this case, triangle-based consistencies [31] achieve the same effect, but not in arbitrary arity cliques.

More generally, we can derive a specific method for propagating cliques in virtually arc consistent instances by extending our reasoning to binary costs. In the following, let $n = |S|$. Since $(1-x_{i1}) = x_{i0}$ for binary domains, the constraint can be written either as $\sum_{x_{i1} \in S} x_{i0} \geq n-1$ or equivalently $\sum_{x_{i1} \in S} x_{i1} \leq 1$. Thus, either $n-1$ or n variables must be assigned 0, meaning either $\binom{n}{2}$ or $\binom{n-1}{2}$ binary $\langle 0,0 \rangle$ tuples will be used. This means

$$\sum_{x_{i1}, x_{j1} \in S, i<j} y_{ij00} \geq \binom{n-1}{2} \tag{2}$$

These constraints can be further strengthened by observing that if we assign $x_i = 1$, then none of the binary tuples $c_{ij}(0,0)$ for $j \neq i$ are used. Thus we can treat these binary tuples as blocks, one for each variable, at least $n-1$ of which have to be used. This is captured by the following system:

$$\begin{aligned} \sum_{x_{i1}, x_{j1} \in S, i<j} y_{ij00} &\geq \binom{n-1}{2} \\ \sum_{x_{i1} \in S} x_{i0} &\geq n-1 \\ x_{i0} &\geq y_{ij00} \qquad \forall i,j \end{aligned} \tag{3}$$

We can incorporate (3) into propagation of the clique constraint. Let $base_i = \sum_{j<k \in [1,n] \setminus \{i\}} c_{jk}(0,0)$ for all $i \in [1,n]$, $base = \min\{base_i \mid i \in [1,n]\}$ and $total = \sum_{j<k \in [1,n]} c_{jk}(0,0)$. We define the *binary-to-clique* transform, denoted as $b \rightarrow clq$ as the transformation which performs the following operations:

1. Add $base$ to $c_\varnothing$
2. Set $c_{ij}(0,0) = 0$ for every binary constraint ij in the clique.
3. Add cost $total - base$ to a_0
4. For every variable i, add cost $comp_i = base_i - base$ to $c_i(1)$.

Proposition 2. *The $b \rightarrow clq$ transform is an EPT.*

Proof. Consider an arbitrary feasible solution of the subproblem. Since it is feasible, at least $n-1$ variables are assigned 0, so we only need to consider the cases where n or $n-1$ variables are 0.

Assume all n variables are assigned 0. The cost of this assignment in the original problem is $total = \sum_{j<k \in [1,n]} c_{jk}(0,0)$. In the reformulated problem, the clique constraint entails cost $a_0 = total - base$, because we assign all zeroes. Together with the cost $base$ projected to $c_\varnothing$, it gives $total$ in the reformulated problem as well.

Assume exactly $n-1$ variables are assigned 0 and that the variable assigned to 1 is X_i. The cost of this assignment is $base_i = \sum_{j<k \in [1,n] \setminus \{i\}} c_{jk}(0,0)$. In the reformulated problem, the cost of the assignment is $base$ from $c_\varnothing$ and $base_i - base$ from $c_i(1)$, giving $base_i$, identical to the original problem.

This transform involves a higher-order transformation, so it is tempting to think that it is stronger than the linear relaxation. Unfortunately, this is not the case.

Proposition 3. *The $b \rightarrow clq$ transform can be expanded to a set of EPTs.*

Proof (Sketch). There exist already positive costs on the (0,0) tuple of binary cost functions. If the 1 value of one unary cost function has positive cost, we can extend it to the binary cost function and project it to the 0 value on the other variable involved. We can do this even if no such positive cost exists and create a temporary negative cost. However, we can now apply the $u \rightarrow clq$ transform, because we have positive costs on the 0 value of the unary cost functions. This projects costs back to the 1 value of the unary cost functions involved, hence covering the deficit created in the first step.

Convergence. As was observed in [28], EDAC may not converge when cost functions of higher arity exist. VAC is better behaved, but it may converge only as the number of iteration grows to infinity. It is straightforward to see that the presence of clique constraints does not raise any such issues: every time either the $u \rightarrow clq$ or $b \rightarrow clq$ transform is applied, the cost $c_\varnothing$ is increased. As $c_\varnothing$ cannot be increased past the global optimum, this means that application of these rules converges. Moreover, $c_\varnothing$ is increased by an integer, so the number of iterations is also bounded by the cost of the global optimum.

3.4 Clique selection and ordering heuristic

Finding Cliques We implement detection of cliques as a preprocessing step. We construct a graph as described previously: there exists a vertex for each variable-value pair and an edge between two vertices if $c_\varnothing + c_p(i) + c_q(j) + c_{pq}(i,j) \geq k$ or if they represent two values of the same variable. We then use the Bron-Kerbosch algorithm [6] with degeneracy ordering to generate a set of cliques. In some cases, the number of cliques can be overwhelming, so we place a limit on the maximum number of cliques generated per top-level branch (which should roughly correspond to the number of cliques that contain a single vertex), as well as a global limit on the total number of cliques. We discard cliques S for which $|varsof(S)| < 3$, as cliques over 2 variables are simply subsets of binary cost functions and can propagate no better.

Selection and Ordering. The order of EPTs may have a large impact on the quality of the resulting lower bounds.

Example 2. Consider two cliques C_1, C_2 with scope $\{X_1, X_2, X_3\}$ and $\{X_2, X_3, X_4\}$, respectively, such that all variables have binary domains and the 1 value from each variable participates in the cliques and $c_i(0) = i$. If we propagate C_1 first, we project cost 3 to $c_\varnothing$ and update the unary costs $c_1(0) = c_2(0) = 0$, $c_3(0) = 1$ and $c_1(1) = 1$ leaving the rest unchanged. We then propagate C_2 and

project 1 unit of cost to $c_\varnothing$ and update $c_3(0) = 0$, $c_4(0) = 3$ and $c_2(1) = 1$. The final cost of $c_\varnothing$ is 4.

On the other hand, if we propagate C_2 first, we project 5 units of cost to $c_\varnothing$, leaving $c_2(0) = c_3(0) = 0, c_4(0) = 1$ and $c_2(1) = 1$. After this, C_1 does not propagate. Hence, by propagating C_2 before C_1, we get a stronger lower bound.

The above problem does not, of course, manifest itself when actually solving the linear relaxation of the problem. It is, however, an inherent limitation of algorithms like EDAC, which have no flexibility to process constraints in a different order in different passes. As we explained above, a more flexible algorithm like VAC would not help either, as clique constraints are redundant with respect to propagation on the hard problem. Therefore, our only recourse is to select an order of propagation before performing the actual propagation. We can choose this order either before search or dynamically at each node of the search. Here, we chose to select the order just once before search begins.

We choose a greedy heuristic to select and order the initial clique constraint propagation. For that, we collect a bounded number of potential clique constraints, as described previously. We then use the classical Chvatal's set covering heuristic [7] to find the best clique which maximizes the product of current arity and current lower bound increase (by simulating the effect of its unary-to-clique transform). We repeat this selection process followed by the corresponding unary-to-clique transform until all the remaining cliques do not increase the lower bound or have all their variables covered by another already-selected clique. Ties are broken by a lexicographic ordering on the scope of the cliques. We keep also all cliques found of arity 3 because they are natively managed by the ToulBar2 solver as ternary/triangle cost functions with dedicated *EDAC* soft arc consistency [35,31].

4 Related Work

The most related work is that of Atamtürk et al [4], who explore adding clique cuts in MIP solvers, not only during preprocessing but also during search. As we descend the search tree, more tuples become effectively hard, creating more cliques. This remains a future direction for us.

Khemmoudj and Bennaceur [22] studied the use of binary cliques to improve lower bounds. These are used to obtain better approximations than EDAC to the optimum of the LP relaxation of a MaxCSP. In contrast to our work here, the strength of the LP relaxation is not improved.

Sontag et al [38] strengthen the LP relaxation by adding higher order cost functions, which may eventually make the relaxation as strong as the polytope of the integer program, at the cost of potentially making the LP explonentially larger. Later, Sontag et al [37] considered adding cuts that correspond to frustrated cycles in the graph, which is more efficient but less powerful.

Cliques can also be handled efficiently in settings outside of linear relaxations. For example, Narodytska and Bacchus [30] proposed a method for MaxSAT which applies max-resolution on cores extracted from the instance. This method

is complete, i.e., will eventually produce the optimum of an instance. This method can perform equivalent reasoning to a clique cut in a polynomial number of steps and with cores that can be discovered in polynomial time.

On the subject of clique detection, Dixon [13], Ansotegui [3] and Biere et al [5] showed several techniques that may uncover cliques that are not explicitly present in the microstructure (called NAND graph in SAT when restricted to binary clauses), using both syntactic and semantic (propagation-based) information.

5 Experimental Results

problem	nb.	d	n	e	vertices	edges	cliques	selected c.	arity	time
Auction/path	86	2	120.2	1,475.7	120.2	1,355.5	143.9	28.8	7.3	0.02
Auction/sched	84	2	159.7	5,759.9	159.7	5,600.2	822.5	3.6	44.9	0.27
MaxClique	62	2	484.3	50,092.8	484.3	49,608.5	8,372.7	325.2	3.5	2.87
SPOT5	20	4	385.1	6,603.3	761.0	9,411.3	5,888.1	127.3	4.1	1.71

Table 1. Clique generation process: number of instances per benchmark category followed by maximum domain size (d), mean number of variables (n), cost functions (e), graph vertices, graph edges, cliques found, and selected cliques, followed by mean arity of selected cliques and CPU time in seconds to find and select cliques. A limit on the maximum number of cliques found was set to $10,000$.

We have implemented the clique generation and clique constraint propagation inside TOULBAR2 (version 0.9.8), an open-source WCSP solver in C++. Among the various benchmarks from [19] (available in LP, WCNF, WCSP, UAI, and MiniZinc formats) where CPLEX reports that clique cuts applied, we chose four problem categories, combinatorial auctions *Auction/path*, *Auction/sched*, maximum clique *MaxClique*, and satellite management *SPOT5*, a total of 252 instances, having binary forbidden tuples and initial unary cost functions such that the unary-to-clique transform increases the lower bound in preprocessing. The first three categories have Boolean domains, whereas SPOT5 has maximum domain size of 4. For each category, we report in Tab.1 the mean value of the size of the problem, the number of cliques found, the number of selected cliques among them and their arity, and the CPU time to find and select the cliques. We limit the maximum number of cliques found to 10,000 in order to control the computation time. The largest CPU time was 11.61 seconds for *MaxClique/c-fat200-5* ($n = 200$ variables, $e = 11,627$ cost functions, and 10,000 selected cliques of arity 3). The arity of selected cliques varies from 3 to 67 (*MaxClique/san1000*).

We compare solving time to find and prove optimality for TOULBAR2 exploiting cliques (denoted as TOULBAR2^{clq}) against the original code without cliques (both using default options, including hybrid best-first search [1]), and against the CP solver GECODE[3], the MaxSAT solvers MAXHS 2.51 [11,10] and EVA 500a

[3] version 4.4.0, using free search.

problem	TOULBAR2		TOULBAR2^{clq}		CPLEX		MAXHS		EVA		GECODE	
	solv.	time	solv.	time	solv.	time	solv.	time	solv.	time	solv.	time
Auction/path	**86**	59	**86**	0.18	**86**	0.01	**86**	0.01	85	102	29	2614
Auction/sched	**84**	110	**84**	0.23	**84**	0.04	**84**	0.04	**84**	0.28	**84**	76
MaxClique	31	1871	37	1508	38	1533	**40**	1510	26	2268	24	2314
SPOT5	4	2884	6	2603	**16**	738	6	2577	13	1260	0	3600

Table 2. Number of solved instances and mean solving computation time in seconds, for TOULBAR2 solver without using cliques compared to TOULBAR2^{clq} exploiting cliques, CPLEX, MAXHS, EVA and GECODE. TOULBAR2^{clq} solving time does not take into account clique generation time (see Tab. 1). A CPU time limit of 1 hour was used for unsolved instances for reporting mean solving times.

[30], and IBM-ILOG CPLEX 12.6.0.0 (using a direct encoding [19] and parameters EPAGAP, EPGAP, and EPINT set to zero to avoid premature stop). All computations were performed on a single core of AMD Opteron 6176 at 2.3 GHz and 8 GB of RAM with a 1-hour CPU time limit[4]. In Tab. 2, we give the number of solved instances within a 1-hour CPU time limit. Among 252 instances, CPLEX solved 224 instances, MAXHS 216, TOULBAR2^{clq} 213, EVA 208, TOULBAR2 205, and GECODE 137. For *Auction*, TOULBAR2^{clq} is more than two orders of magnitude faster than TOULBAR2, GECODE (which cannot solve 57 *Auction/path* instances in 1 hour) and EVA on *Auction/path* (*cat_paths_60_170_0005* instance unsolved in 1 hour). Still TOULBAR2^{clq} is one order of magnitude slower than CPLEX and MAXHS. On *MaxClique* (resp. *SPOT5*), TOULBAR2^{clq} solved 6 (resp. 2) more instances than without using cliques. For example, TOULBAR2^{clq} solved *MaxClique/MANN_a45* in 57.9 seconds (taking 0.24 supplementary seconds to generate the 330 selected cliques) whereas without using cliques it could not finish in 1 hour (MAXHS took 28 seconds, CPLEX 93 sec., and EVA and GECODE could not solve in 1 hour). *SPOT5/404* was solved in 6 seconds using 32 cliques and 88.7 seconds without using cliques (CPLEX took 0.02 seconds, EVA 0.07 sec., MAXHS 8 sec., and GECODE could not solve in 1 hour).

Finally, we summarize the evolution of lower and upper bounds for each solver over all instances in Figure 1. Specifically, for each instance I we normalize all costs as follows: the initial lower bound produced by TOULBAR2 is 0; the best – but potentially suboptimal – solution found by any solver is 1; the worst solution is 2. This normalization is invariant to translation and scaling. Additionally, we simply normalize time from 0 to 1, corresponding to 1 hour. A point $\langle x, y \rangle$ on the lower bound line for solver S in Figure 1 means that after normalized runtime x, solver S has proved on average over all instances a normalized lower bound of y and similarly for the upper bound. We show both the upper and lower bound curves for all solvers evaluated here, except GECODE which produces no meaningful lower bound before it proves optimality. We observed that using cliques, it mainly improves lower bounds for TOULBAR2. For these benchmarks,

[4] Using parameter *-pe parallel_smp 2* on a SUN Grid Engine to ensure half-load of the cores on the cluster.

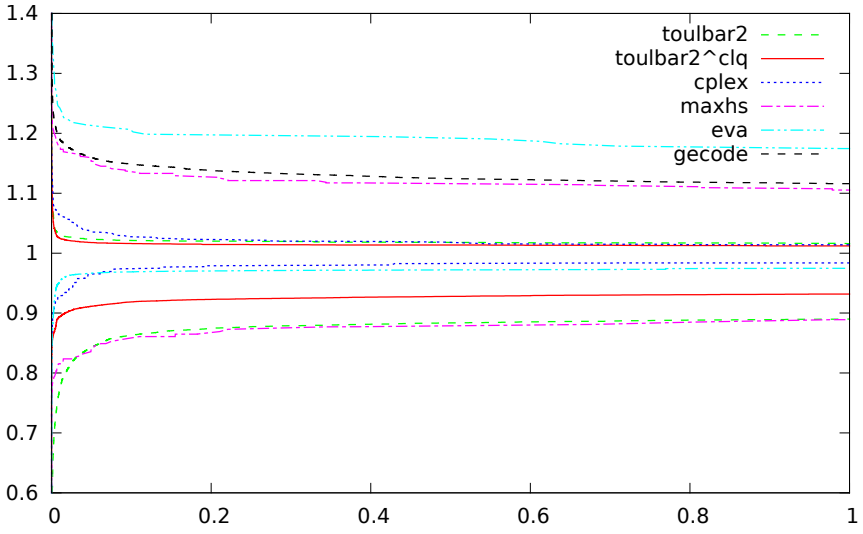

Fig. 1. Normalized lower and upper bounds on 252 instances as time passes.

CPLEX got the best lower bound curve and TOULBAR2^{clq} the best upper bound curve.

6 Conclusions

We have shown how the idea of clique cut originated from MIP can be exploited in the context of WCSPs. Using these cuts in WCSP is significantly more complicated than in integer programming, owing to a large degree to the fact that the fast specialized algorithms that are used in place of solving the linear relaxation have weaknesses which seem to be particularly exposed by clique constraints. To address these shortcomings, we provide two specific EPTs, unary-to-clique and binary-to-clique, which propagate isolated clique constraints optimally, even if costs are hidden in binary cost functions. We then gave an algorithm to greedily select and order a subset of potential cliques in preprocessing and do the propagation on these selected cliques during search. In an experimental evaluation, we have obtained large improvements over the existing complete solver TOULBAR2 on several benchmarks, significantly reducing the gap to state-of-the-art solver CPLEX.

Acknowledgements This work has been partially funded by the french "Agence nationale de la Recherche", reference ANR-16-C40-0028. We are grateful to the Bioinfo Genotoul platform Toulouse Midi-Pyrenees for providing computing resources.

References

1. Allouche, D., de Givry, S., Katsirelos, G., Schiex, T., Zytnicki, M.: Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In: Proc. of CP-15. pp. 12–28. Cork, Ireland (2015)
2. Allouche, D., Bessière, C., Boizumault, P., de Givry, S., Gutierrez, P., Lee, J.H., Leung, K.L., Loudni, S., Métivier, J.P., Schiex, T., Wu, Y.: Tractability-preserving transformations of global cost functions. Artificial Intelligence 238, 166–189 (2016)
3. Anstegui Gil, C.: Complete SAT solvers for Many-Valued CNF Formulas. Ph.D. thesis, University of Lleida (2004)
4. Atamtürk, A., Nemhauser, G.L., Savelsbergh, M.W.: Conflict graphs in solving integer programming problems. E. J. of Operational Research 121(1), 40–55 (2000)
5. Biere, A., Le Berre, D., Lonca, E., Manthey, N.: Detecting cardinality constraints in cnf. In: SAT 2014. pp. 285–301 (2014)
6. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM 16(9), 575–577 (1973)
7. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4(3), 233–235 (1979)
8. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M.: Virtual arc consistency for weighted csp. In: Proc. of AAAI-08. Chicago, IL (2008)
9. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174(7–8), 449–478 (2010)
10. Davies, J., Bacchus, F.: Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In: Proc. of CP-11. pp. 225–239. Perugia, Italy (2011)
11. Davies, J., Bacchus, F.: Postponing Optimization to Speed Up MAXSAT Solving. In: Proc. of CP-13. pp. 247–262. Uppsala, Sweden (2013)
12. Dechter, R., Mateescu, R.: And/or search spaces for graphical models. Artificial intelligence 171(2), 73–106 (2007)
13. Dixon, H.E.: Automating Psuedo-Boolean Inference within a DPLL Framework. Ph.D. thesis, University of Oregon (2004)
14. de Givry, S., Prestwich, S., O'Sullivan, B.: Dead-End Elimination for Weighted CSP. In: Proc. of CP-13. pp. 263–272. Uppsala, Sweden (2013)
15. de Givry, S., Schiex, T., Verfaillie, G.: Exploiting Tree Decomposition and Soft Local Consistency in Weighted CSP. In: Proc. of AAAI-06. Boston, MA (2006)
16. de Givry, S., Zytnicki, M., Heras, F., Larrosa, J.: Existential arc consistency: Getting closer to full arc consistency in weighted CSPs. In: Proc. of IJCAI-05. pp. 84–89. Edinburgh, Scotland (2005)
17. Globerson, A., Jaakkola, T.: Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In: Proc. of NIPS. Vancouver, Canada (2007)
18. van Hoeve, W.J., Pesant, G., Rousseau, L.: On global warming: Flow-based soft global constraints. J. Heuristics 12(4-5), 347–373 (2006)
19. Hurley, B., O'Sullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., de Givry, S.: Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. Constraints 21(3), 413–434 (2016)
20. Jégou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artif. Intell. 146(1), 43–75 (2003)
21. Kask, K., Dechter, R.: Branch and bound with mini-bucket heuristics. In: Proc. of IJCAI-99. vol. 99, pp. 426–433 (1999)
22. Khemmoudj, M.O.I., Bennaceur, H.: Clique inference process for solving Max-CSP. European Journal of Operational Research 199(3), 665–673 (2009)

23. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. IEEE Pattern Analysis and Machine Intelligence 28(10), 1568–1583 (2006)
24. Larrosa, J.: Boosting search with variable elimination. In: Proc. of CP-00. pp. 291–305. Singapore (2000)
25. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for weighted CSP. In: Proc. of the 18^{th} IJCAI. pp. 239–244. Acapulco, Mexico (2003)
26. Lecoutre, C., Roussel, O., Dehani, D.: WCSP Integration of Soft Neighborhood Substitutability. In: Proc. of CP-12. pp. 406–421. Quebec, Canada (2012)
27. Lee, J.H.M., Leung, K.L.: Consistency Techniques for Global Cost Functions in Weighted Constraint Satisfaction. J. of Artificial Intelligence R. 43, 257–292 (2012)
28. Lee, J.H., Leung, K.L.: A stronger consistency for soft global constraints in weighted constraint satisfaction. In: Proc. of AAAI-10. Atlanta, USA (2010)
29. Marinescu, R., Dechter, R.: AND/OR branch-and-bound for graphical models. In: Proc. of IJCAI-05, Edinburgh, Scotland, UK. pp. 224–229 (2005)
30. Narodytska, N., Bacchus, F.: Maximum Satisfiability Using Core-Guided MAXSAT Resolution. In: Proc. of AAAI-14. pp. 2717–2723. Quebec City, Canada (2014)
31. Nguyen, H., Bessiere, C., de Givry, S., Schiex, T.: triangle-based consistencies for cost function networks. Constraints 22(2), 230–264 (2016)
32. Prusa, D., Werner, T.: Universality of the local marginal polytope. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. pp. 1738–1743 (2013)
33. Quimper, C., Walsh, T.: Global grammar constraints. In: Proc. of CP-06. pp. 751–755. Nantes, France (2006)
34. Quimper, C., Walsh, T.: Decompositions of grammar constraints. CoRR abs/0903.0470 (2009)
35. Sánchez, M., de Givry, S., Schiex, T.: Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. Constraints 13(1), 130–154 (2008)
36. Schlesinger, M.I.: Syntactic analysis of two-dimensional visual signals in noisy conditions. Kibernetika 4(113-130) (1976), in Russian
37. Sontag, D., Choe, D., Li, Y.: Efficiently searching for frustrated cycles in MAP inference. In: Proc. of UAI. pp. 795–804 (2012)
38. Sontag, D., Meltzer, T., Globerson, A., Weiss, Y., Jaakkola, T.: Tightening LP relaxations for MAP using message-passing. In: Proc. of UAI. pp. 503–510 (2008)
39. Werner, T.: A linear programming approach to max-sum problem: A review. IEEE transactions on pattern analysis and machine intelligence 29(7) (2007)