Multiple-choice knapsack constraint in graphical models*

Pierre Montalbano[1][0000−0001−8126−892X], Simon de Givry[1][0000−0002−2242−0458], and George Katsirelos[2][0000−0002−3727−6698]

[1] Université Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France {pierre.montalbano,simon.de-givry}@inrae.fr
[2] Université Fédérale de Toulouse, ANITI, INRAE, MIA Paris, AgroParisTech, 75231 Paris, France gkatsi@gmail.com

Abstract. Graphical models, such as cost function networks (CFNs), can compactly express large decomposable functions, which leads to efficient inference algorithms. Most methods for computing lower bounds in Branch-and-Bound minimization compute feasible dual solutions of a specific linear relaxation. These methods are more effective than solving the linear relaxation exactly, with better worst-case time complexity and better performance in practice. However, these algorithms are specialized to the structure of the linear relaxation of a CFN and cannot, for example, deal with constraints that cannot be expressed in extension, such as linear constraints of large arity.

In this work, we show how to extend soft local consistencies, a set of approximate inference techniques for CFNs, so that they handle linear constraints, as well as combinations of linear constraints with at-most-one constraints. We embedded the resulting algorithm in TOULBAR2, an exact Branch-and-Bound solver for CFNs which has demonstrated superior results in several graphical model competitions and is state-of-the-art for solving large computational protein design (CPD) problems. We significantly improved performance of the solver in CPD with diversity guarantees. It also compared favorably with integer linear programming solvers on knapsack problems with conflict graphs.

Keywords: graphical model · cost function network · knapsack problem.

1 Introduction

A Graphical Model (GM) may express an arbitrary complex function on several variables as a combination of smaller local functions on subsets of the variables. GMs have been used to reason about logic and probabilities. A deterministic GM can represent a Constraint Satisfaction Problem (CSP) where each local function is a constraint evaluating to true (satisfied) or false (unsatisfied) and the combination operator is Boolean conjunction. It can also represent a Cost Function

* This research was funded by the French "Agence Nationale de la Recherche" through grants ANR-18-EURE-0021 and ANR-19-P3IA-0004.

Network (CFN), where each function evaluates to a cost and the combination operator is addition [14]. A probabilistic GM represents a probability distribution on random variables. Local functions may correspond to conditional probability distributions as in Bayesian Networks (BN) or potentials as in Markov Random Fields (MRF) [28]. They are combined by multiplication. In the following, we focus on CFNs. It can be shown that finding the Maximum A Posteriori assignment on MRFs (MAP/MRF) or the Most Probable Explanation on BNs (MPE/BN) can be cast as finding a solution of minimum cost in an appropriate CFN [13]. By allowing infinite costs to represent infeasibility, CFNs can be seen as a strict generalization of CSPs.

Exact methods to solve GMs/CFNs mostly rely on Branch-and-Bound (B&B) algorithms [36,20]. These methods have proved useful in many GM applications, such as resource allocation [10], image analysis [23], or computational biology [4,1]. For those, it has been shown to outperform other approaches, including Integer Linear Programming (ILP), MaxSAT and Constraint Programming (CP) [25].

CFNs have no native way to express linear constraints. This is in large part due to the algorithms used to compute lower bounds in B&B, which require that all constraints be expressed in extension. In many cases, having the ability to add such constraints would significantly improve the usefulness of CFNs. For example, when searching for diverse solutions, the Hamming distance constraint is naturally expressed as a linear constraint. There are ways to work around this [39,40] but, as we show later, they come with a non-trivial performance penalty. The lack of linear constraints is even more severe when the constraints have large coefficients, as in the knapsack problem with a conflict graph (KPCG). In this case, there is no workaround for the lack of linear constraints and CFN technology cannot be applied.

Contributions. Here, we show how to extend soft local consistency algorithms, a set of approximate inference techniques for CFNs, to deal with Pseudo-Boolean linear constraints (PB constraints for short), i.e., linear constraints over 0/1 variables. In the presence of *unary* cost functions (a cost function coupling a cost to each value), these correspond to knapsack constraints. We additionally consider the combination of PB constraints with Exactly-One (EO) or At-Most-One (AMO) constraints, which correspond to multiple-choice knapsack constraints [37], allowing finite-domain variables.

This new ability enables more modeling options for GM/CFN users, which we demonstrate by applying it to generating diverse solutions for Computational Protein Design (CPD). Here, the objective function has quadratic and linear terms that can be decomposed in a sum of *binary* cost functions on pairs of variables (the quadratic terms) and unary cost functions on single variables (linear terms). Searching for diverse solutions introduces linear constraints in the model (see Section 5.1). Our approach also compared favorably with a state-of-the-art ILP solver on knapsack problems with conflict graphs.

2 Related work

Problems defined by PB constraints are a generalization of the SAT problem. Solvers for PB SAT typically use SAT-inspired constraint learning techniques, either by direct translation to Conjunctive Normal Form (CNF) [41] or by generalizing the clause learning mechanism to PB constraints [19]. These solvers typically do not compute lower bounds during search and have to rely on conflict reasoning only to prove bounds. A notable exception is RoundingSAT [26], which uses a Linear Programming (LP) solver to compute bounds during search and learn constraints from bound violations, but limits the number of iterations given to the LP solver in order to keep the runtime overhead of the solver reasonable. This is in contrast to our approach, which uses a suboptimal LP solver, but places no resource bounds on it. Also, PB solvers are usually restricted to a linear objective, whereas our approach can combine PB constraints with non-linear quadratic (or more) cost functions. PB solvers can also exploit the presence of AMO or EO constraints to strengthen propagation of PB constraints [5,7].

ILP solvers are well suited to solve CFNs, given the local polytope. Their LP solving is not limited to a specific form of LP, like soft local consistency algorithms such as Existential Directional Arc Consistency (EDAC) [21] and Virtual AC (VAC) [12] are, therefore they have no issue reasoning with other linear constraints, as well as combinations with AMO/EO constraints. However, previous evaluations [25] showed that the size of the linear program that specifies that local polytope is often too large even for such highly optimized implementations and therefore they perform worse than a dedicated CFN solver in such problems.

On the CFN side, there has been work on clique constraints [22], a special case of PB constraints. Dlask and Werner [16,17] have shown how to handle arbitrary LPs using BCA algorithms, based on a generalization of VAC. However, despite recent advances [49], BCA algorithms remain too costly for use at every node of a B&B. Many (soft) global constraints can be described by a set of (soft) linear constraints, but require an LP solver [34]. In addition, maintaining (weak) EDAC and the coupling with the other local cost functions can be costly in practice. This was also the case for other soft global constraints exploiting flow-based or dynamic programming algorithms [33,35]. In our approach, we propose a simple and effective soft local consistency called Full $\emptyset$-Inverse Consistency for PB constraints. Finally, we can decompose linear constraints using cost functions of arity 3 and intermediate variables [3], similar to CNF encodings used by PB solvers. However, the size of the domains of the intermediate variables increases linearly with the value of the coefficients of the PB constraints.

3 Preliminaries

Definition 1. *A Cost Function Network (CFN) P is a tuple $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{C}, \top)$ where $\boldsymbol{X}$ is a set of variables, with finite domain $\boldsymbol{D}(x)$ for $x \in \boldsymbol{X}$. $\boldsymbol{C}$ is a set of constraints. Each constraint $c \in \boldsymbol{C}$ is defined over a subset of variables called its scope ($scope(c) \subseteq \boldsymbol{X}$). $\top$ is a maximum cost indicating a forbidden assignment.*

The size of the scope of a constraint is its arity. Unary (resp. binary) cost functions have arity 1 (resp. 2). A partial assignment τ is an assignment of all the variables x_i in its scope ($\mathit{scope}(\tau)$) to a value of its domain $\boldsymbol{D}(x_i)$. The set of all the partial assignments on a scope $\boldsymbol{S}$ is denoted $\tau(\boldsymbol{S})$. A constraint over a scope $\boldsymbol{S}$ is denoted $c_{\boldsymbol{S}}$. The cost of a partial assignment τ for a constraint $c_{\boldsymbol{S}}$ is denoted $c_{\boldsymbol{S}}(\tau)$ with $\boldsymbol{S} \subseteq \mathit{scope}(\tau)$. Without loss of generality, we assume all costs are positive integers, bounded by $\top$, a special constant signifying infeasibility. Hence if $c_{\boldsymbol{S}}(\tau) = \top$ then the assignment τ is not a feasible solution. A constraint $c_{\boldsymbol{S}}$ is hard if for all $\tau \in \tau(\boldsymbol{S})$, $c_{\boldsymbol{S}}(\tau) \in \{0, \top\}$, otherwise it is soft. A CFN P that contains only hard constraints is a constraint network (CN). In the following, we use the term *cost function* interchangeably with the term constraint. An assignment τ with $\mathit{scope}(\tau) = \boldsymbol{X}$ is a complete assignment. The cost of a complete assignment τ is given by $c_P(\tau) = \sum_{c_{\boldsymbol{S}} \in \boldsymbol{C}} c_{\boldsymbol{S}}(\tau)$. The Weighted Constraint Satisfaction Problem (WCSP) asks, given a CFN P, to find a complete assignment minimizing $c_P(\tau)$. This task is NP-hard [14]. When the underlying CFN is a CN, the problem is the CSP, which we call crisp CSP here. In the following, we use WCSP to refer both to the optimization task and the underlying CFN.

In this paper, we assume there exists exactly one unary constraint for each variable and we say that the unary cost of $x_i = v$ for some $v \in \boldsymbol{D}(x_i)$ is $c_i(v)$. We also assume the existence of a constraint $c_\varnothing$ with empty scope, which represents a constant in the objective function and, since there exist no negative costs, it is a lower bound on the cost of all possible assignments.

Exact methods to solve GMs/CFNs mostly rely on Branch-and-Bound (B&B) algorithms [36,20]. At every node of the B&B tree, the solver computes a bound and closes the node if that bound is higher than the cost of the incumbent solution or if it represents infeasibility. Typical bounding algorithms compute either static memory-intensive bounds [15] or memory-light ones [12] better suited to dynamic variable orderings. The latter, on which we focus here, are called Soft Arc Consistencies (SAC) because they reason on each non-unary cost function one by one, in a generalization of propagation in CSP.

Soft arc consistencies use $c_\varnothing$ as the lower bound and compute a reparameterization of the instance with a higher $c_\varnothing$. A reparameterization P' of a WCSP P is a WCSP with an identical structure, i.e., one where there exist constraints over the same scopes, the costs assigned by each individual cost function may differ, but $c_P(\tau) = c_{P'}(\tau)$ for all complete assignments τ.

Procedure MoveCost($c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2}, \tau_1, \alpha$): Move α units of cost between the tuple τ_1 of scope $\boldsymbol{S}_1$ and tuples τ_2 that extend τ_1 in scope $\boldsymbol{S}_2$

```
  Data: Scopes $\boldsymbol{S}_1 \subset \boldsymbol{S}_2$
  Data: $\tau_1 \in \tau(\boldsymbol{S}_1)$
  Data: cost $\alpha$ to move
1 $c_{\boldsymbol{S}_1}(\tau_1) \leftarrow c_{\boldsymbol{S}_1}(\tau_1) + \alpha$
2 foreach $\tau_2 \in \tau(\boldsymbol{S}_2) \mid \tau_2[\boldsymbol{S}_1] = \tau_1$ do
3     $c_{\boldsymbol{S}_2}(\tau_2) \leftarrow c_{\boldsymbol{S}_2}(\tau_2) - \alpha$
```

All reparameterizations that we study here are computed as a sequence of local *Equivalence Preserving Transformations* (EPTs). Let $\boldsymbol{S}_1 \subset \boldsymbol{S}_2$ be two scopes with corresponding cost functions $c_{\boldsymbol{S}_1}$ and $c_{\boldsymbol{S}_2}$. Procedure MoveCost describes how a cost α moves between the corresponding cost functions. To see its correctness, observe if τ_1 is used in a complete assignment, then exactly one extension of τ_1 to $\boldsymbol{S}_2$ will be used. Therefore, the sum of $c_{\boldsymbol{S}_1}$ and $c_{\boldsymbol{S}_2}$ remains unaffected whether the cost α is attributed to τ_1 in $c_{\boldsymbol{S}_1}$ or to all of its extensions τ_2 in $c_{\boldsymbol{S}_2}$. As an example, it is clear that adding a cost α on $c_x(a)$ and subtracting a cost α on $c_{\{x,y\}}(\{x = a, y = b\})$ for all $b \in D(y)$ preserves problem equivalence. Indeed, paying α when we assign $x = a$ (cost function $c_x(a) = \alpha$) or when we assign $x = a$ and $y = b$ ($\forall b \in D(y)$) (cost function $c_{\{x,y\}}(\{x = a, y = b\}) = \alpha$, $\forall b \in D(y)$) is equivalent. As a matter of terminology, when $\alpha > 0$, cost moves from the larger arity cost function $c_{\boldsymbol{S}_2}$ to the smaller arity $c_{\boldsymbol{S}_1}$ and the move is called a projection, denoted $project(c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2}, \tau_1, \alpha)$. When $\alpha < 0$, cost moves to the larger arity cost function $c_{\boldsymbol{S}_2}$ and the move is called an extension, denoted $extend(c_{\boldsymbol{S}_1}, \tau_1, c_{\boldsymbol{S}_2}, -\alpha)$, equivalent to MoveCost$(c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2}, \tau_1, \alpha)$. When $\boldsymbol{S}_1 = \emptyset$ and $|\boldsymbol{S}_2| = 1$, with $\boldsymbol{S}_2 = \{x_i\}$, the move is called a unary projection, denoted $unaryProject(c_i, \alpha)$, equivalent to MoveCost$(c_\varnothing, c_i, \emptyset, \alpha)$. We never perform extensions from $c_\varnothing$, so it monotonically increases during the run of an algorithm and as we descend a branch of the search tree.

Finding which cost moves lead to an optimal reparameterization, which means one that derives the optimal increase in the lower bound, is not obvious. It has been shown that any reparameterization can be derived by a set of local cost moves [29] and that the optimal reparameterization (with α rational) – and, equivalently, the optimal set of cost moves – can be found from the optimal dual solution of the following linear relaxation of the WCSP [12], whose feasible region is called the *local polytope*:

$$
\begin{aligned}
\min \quad & \sum_{c_{\boldsymbol{S}} \in \boldsymbol{C}, \tau \in \tau(\boldsymbol{S})} c_{\boldsymbol{S}}(\tau) \times y_\tau \\
s.t. \quad & \\
& y_{\tau_1} = \sum_{\tau_2 \in \tau(\boldsymbol{S}_2), \tau_2[\boldsymbol{S}_1] = \tau_1} y_{\tau_2} && \forall c_{\boldsymbol{S}_1}, c_{\boldsymbol{S}_2} \in \boldsymbol{C}, \boldsymbol{S}_1 \subset \boldsymbol{S}_2, \\
& && \tau_1 \in \tau(\boldsymbol{S}_1), |\boldsymbol{S}_1| \geq 1 \\
& \sum_{\tau \in \tau(\boldsymbol{S})} y_\tau = 1 && \forall c_{\boldsymbol{S}} \in \boldsymbol{C}, |\boldsymbol{S}| \geq 1
\end{aligned}
$$

However, solving this LP to optimality is often prohibitively expensive because the worst-case complexity of an exact LP algorithm is $O(N^{2.5})$ [50], with $N \in O(ed + nd)$ for binary WCSPs, where e is the number of distinct binary cost functions, n is the number of WCSP variables and d is the maximum domain size. The poor asymptotic complexity matches empirical observation [25]. Moreover, the particular structure of this LP does not allow for a more efficient solving algorithm, as it has been shown that solving LPs of this form is as hard as solving any LPs [38]. Instead, work has focused on producing good but potentially suboptimal feasible dual solutions. Various algorithms have been proposed for this, going all the way back to Schlesinger [44], who first expressed the problem as linear optimization and gave a specific algorithm for optimizing the dual. Since Schlesinger, a long line of algorithms has been pursued both in areas like image analysis [29,51,46,30,45,47], where Block-Coordinate Ascent (BCA) algorithms were developed, and constraint programming [43,31,21,53,12], where they are called *soft local consistencies*. Notably, the strongest algorithms from both lines of research, such as TRWS [29] and VAC [12] converge on fixpoints with the same properties.

We do not describe all the existing local consistency algorithms but we need the following consistency properties:

Definition 2. *A WCSP P is Node Consistent (NC) [31] if for every variable* $x_i \in \boldsymbol{X}$ *there exists a value* $v \in \boldsymbol{D}(x_i)$ *such that* $c_i(v) = 0$ *and for every value* $v' \in \boldsymbol{D}(x_i)$, $c_\varnothing + c_i(v') < \top$.

In the following, we assume that a WCSP is NC before our propagator runs.

Definition 3. *A WCSP P is* $\emptyset$*-Inverse Consistent (*$\emptyset$*IC) [53] if for every cost function* $c_{\boldsymbol{S}} \in \boldsymbol{C}$ *there exists a tuple* $\tau \in \tau(\boldsymbol{S})$ *such that* $c_{\boldsymbol{S}}(\tau) = 0$.

Definition 4. *A WCSP P is Existential Arc Consistent (EAC) [21] if it is NC and for every* $x_i \in \boldsymbol{X}$ *there exists a value* $v \in \boldsymbol{D}(x_i)$ *such that* $c_i(v) = 0$ *and for every cost function* $c_{\boldsymbol{S}} \in \boldsymbol{C}$, $x_i \in \boldsymbol{S}$, $|\boldsymbol{S}| > 1$, *there exists a tuple* $\tau \in \tau(\boldsymbol{S})$ *verifying* $\tau[x_i] = \{v\}$ *(i.e.,* $x_i = v$ *in* τ*) and* $c_{\boldsymbol{S}}(\tau) + \sum_{x_j \in \boldsymbol{S}} c_j(\tau[x_j]) = 0$. *Value* v *is called an EAC support.*

This last definition applies only to binary cost function networks.[3] A weaker notion of EAC has been defined on global cost functions in order to avoid cost oscillation [33]. Given a variable x_i, it relies on a partition of the unary cost functions $c_j(\tau[x_j]), x_j \in \boldsymbol{X}$ such that each part is associated to some non-unary cost function $c_{\boldsymbol{S}}$ related to x_i ($x_i \in \boldsymbol{S}$).

We follow another weakening approach related to $\emptyset$IC. We strengthen the previous definition to take into account unary costs as in EAC.

Definition 5. *A WCSP is Full* $\emptyset$*-Inverse Consistent (F*$\emptyset$*IC) if for every cost function* $c_{\boldsymbol{S}} \in \boldsymbol{C}$ *there exists* $\tau \in \tau(\boldsymbol{S})$ *such that* $c_{\boldsymbol{S}}(\tau) + \sum_{x_j \in \boldsymbol{S}} c_j(\tau[x_j]) = 0$.

Compared to existing notions of consistency, F$\emptyset$IC is weaker than T-DAC [2]. It is also weaker than EAC on binary cost function networks, but it is incomparable with weak EAC [33] on non-binary networks.

Example 1. Consider two variables x, y with $\boldsymbol{D}(x) = \boldsymbol{D}(y) = \{a, b, c\}$ and three cost functions $c_x, c_y, c_{\{x,y\}}$ such that the only non-zero costs are $c_x(a) = c_y(a) = 1$, $c_{\{x,y\}}(\{x = b, y = b\}) = c_{\{x,y\}}(\{x = b, y = c\}) = c_{\{x,y\}}(\{x = c, y = b\}) = 1$, and $c_{\{x,y\}}(\{x = c, y = c\}) = 2$.

[3] An extension to ternary cost functions has been proposed [42] but it requires managing all scope intersections and not only unary cost functions.

Let $\forall u \in \boldsymbol{D}(x), \alpha_u = \min_{v \in \boldsymbol{D}(y)}(c_{\{x,y\}}(\{x = u, y = v\}) + c_y(v))$ and $\forall v \in \boldsymbol{D}(y), \beta_v = \max_{u \in \boldsymbol{D}(x)}(\alpha_u - c_{\{x,y\}}(\{x = u, y = v\}))$. We apply $project(c_x, c_{\{x,y\}}, \{x = u\}, \alpha_u)$ for each value $u \in \boldsymbol{D}(x)$ and $extend(c_y, \{y = v\}, c_{\{x,y\}}, \beta_v)$ for each value $v \in \boldsymbol{D}(y)$. These cost moves will result in adding a cost $\beta_v - \alpha_u$ to every tuple in $c_{\{x,y\}}$. We have $\alpha_a = 0, \alpha_b = \alpha_c = 1$ and $\beta_a = 1, \beta_b = \beta_c = 0$. All the costs remain positive (proof in [32]). The reparameterized cost functions are $c_x(a) = c_x(b) = c_x(c) = 1$, $c_{\{x,y\}}(\{x = a, y = a\}) = c_{\{x,y\}}(\{x = c, y = c\}) = 1$, the rest being equal to 0. We can now increase $c_\varnothing$ by 1 using $unaryProject(c_x, 1)$. The resulting WCSP is EAC.

For each of the consistencies we defined above, there exist corresponding algorithms that compute parametrization that satisfy them in polynomial-time[4]. Given the connection to linear programming, these reparameterizations map to feasible dual solutions of the local polytope. However, these algorithms rely on all constraints being expressed in extension, meaning that for all constraints the cost of every partial assignment must be explicitly written. This is not the case for many constraints that are typically used in modeling in CP, namely global constraints, i.e., those whose definition does not imply a fixed arity. In order to enforce these soft local consistencies in instances that contain global constraints, we need to define bespoke algorithms. In contrast with crisp CSPs, these algorithms must do more than prune values that appear in no feasible solution. They must compute a reparameterization such that the constraint satisfies the appropriate consistency, F∅IC here.

Here, we deal with pseudo-Boolean (PB) linear constraints and their generalizations. These are constraints of the form $\sum_{x_i \in \boldsymbol{S}} w_i x_i \triangle C$, where $\boldsymbol{S}$ is a scope, all $x_i \in \boldsymbol{S}$ are Boolean variables, w_i and C are constants and $\triangle \in \{<, \leq, \neq, \geq, >\}$. A PB constraint is normalized if w_i, $C \geq 0$ and $\triangle$ is $\geq$. Any PB constraint can be written as a combination of normalized PB constraints. It is possible to detect in linear time whether this constraint is satisfiable in a crisp CSP, by testing if $\sum_{x_i \in \boldsymbol{S}} \max(0, w_i) \geq C$. It is also possible to detect values that appear in no solutions by computing all partial sums of $|\boldsymbol{S}| - 1$ variables, in linear time.

A PB constraint is an at-most-one (AMO) constraint if it has the form $\sum_{x_i \in \boldsymbol{S}} x_i \leq 1$, normalized as $\sum_{x_i \in \boldsymbol{S}} -x_i \geq -1$. It is an exactly-one (EO) constraint if it has the form $\sum_{x_i \in \boldsymbol{S}} x_i = 1$.

4 Pseudo-Boolean constraints in CFNs

The specific constraint we consider here is a pseudo-Boolean constraint $\sum_{x_i \in \boldsymbol{S}} w_i x_i \geq C$ along with a partition of its variables into sets $A_1, \ldots, A_k$ such that there exists an EO constraint among the variables of each partition A_i.

Reformulations. This formulation allows us to express PB constraints over multi-valued variables. Let $\boldsymbol{S}$ be a scope over a set of WCSP variables with arbitrary domains, and w_{iv} weights for each value. The constraint $\sum_{x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i)} w_{iv} x_{iv}$

[4] E.g., EDAC [21], an extension of EAC property, is maintained in $O(ed^2 \max(nd, \top))$ for a WCSP with n variables, maximum domain size d, and e binary cost functions.

$\geq C$, where x_{iv} is the 0/1 variable which takes the value 1 if $x_i = v$, matches the pattern described above, with partitions $A_i = \{x_{iv} \mid v \in \boldsymbol{D}(X_i)\}$.

Finally, this formulation admits the case where there exists an AMO constraint over some partitions: we add another 0/1 variable in each such partition and give it weight 0, so that this partition now has an EO constraint.

Constraint representation. We will focus here on $F\emptyset IC$ as the soft consistency we aim to enforce. But first, we need an appropriate encoding that can represent the state of the constraint after a series of cost moves to and from unary cost functions, without storing a cost for each of the exponentially (in the arity of the constraint) many tuples. Observe first that the cost of any given tuple starts out at 0 for allowed tuples and $\top$ for tuples that violate the constraint. After some cost moves, the cost of each tuple is the sum of costs that have been moved to or from the values it contains. Therefore, it can be expressed as a linear function. Let δ_{iv} be the total cost that has been moved between the constraint and the corresponding unary cost and $\delta_\emptyset$ the cost we have moved from this constraint to $c_\varnothing$. Therefore, initially $\delta_\emptyset = 0$ and $\delta_{iv} = 0$ for all i, v. We use the following integer program as the representation of the constraint.

$$\begin{aligned} \min \ & \textstyle\sum_{x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i)} \delta_{iv} x_{iv} - \delta_\emptyset && (1)\\ s.t. \ & \\ & \textstyle\sum_{x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i)} w_{iv} x_{iv} \geq C && (2)\\ & \textstyle\sum_{v \in \boldsymbol{D}(x_i)} x_{iv} = 1, \qquad \forall x_i \in \boldsymbol{S} && (3)\\ & x_{iv} \in \{0,1\}, \qquad \forall x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i) && (4) \end{aligned}$$

We call this $ILP_\emptyset$. The main property of $ILP_\emptyset$ is that the cost of any feasible complete assignment is equal to the cost of the corresponding tuple in $c_{\boldsymbol{S}}$ after any sequence of cost moves. Hence, $opt(ILP_\emptyset) > 0$, if and only if $c_{\boldsymbol{S}}$ is not $\emptyset IC$, and we can move some cost to $c_\varnothing$: $project(c_\varnothing, c_{\boldsymbol{S}}, \emptyset, opt(ILP_\emptyset))$.

However, for the purposes of detecting violations of $F\emptyset IC$, it is not enough to look at the cost of tuples of the constraint, as we must also take unary costs into account. Therefore, while $ILP_\emptyset$ remains the representation of the constraint, the propagator considers the problem with the modified objective

$$\min \sum_{x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i)} (\delta_{iv} + c_i(v)) x_{iv} - \delta_\emptyset \tag{5}$$

Let this problem be $ILP_{F\emptyset}$. $c_{\boldsymbol{S}}$ is $F\emptyset IC$ if and only if $opt(ILP_{F\emptyset}) = 0$. In the following, we write $p_{iv} = \delta_{iv} + c_i(v)$ for compactness, when it does not matter how much of the coefficient came from δ_{iv} and how much came from $c_i(v)$. In contrast with $ILP_\emptyset$, if $opt(ILP_{F\emptyset}) > opt(ILP_\emptyset)$, we cannot move $opt(ILP_{F\emptyset})$ units of cost to $c_\varnothing$. Instead, we first have to move some cost from unary cost functions into the constraint before we can project it to $c_\varnothing$. In this case, the composition of p_{iv} from δ_{iv} and $c_i(v)$ is significant.

Unfortunately, $ILP_\emptyset$ and $ILP_{F\emptyset}$ have the knapsack problem as a special case, hence it is NP-hard to determine whether a PB constraint is $\emptyset IC$ or $F\emptyset IC$. Therefore, we detect only a subset of cases where the constraint is not $F\emptyset IC$ by relaxing the integrality constraint (4) into $0 \leq x_{iv} \leq 1$ and solving the resulting linear programs, called $LP_\emptyset$ and $LP_{F\emptyset}$, respectively. This forgoes the guarantee that $opt(LP_{F\emptyset}) = 0$ if and only if the constraint is $F\emptyset IC$, and satisfies only the 'only if' part. More simply, if $opt(LP_{F\emptyset}) > 0$ then the constraint is not $F\emptyset IC$, and similarly for $LP_\emptyset$ and $\emptyset IC$.

$LP_{F\emptyset}$ has a special structure. It is a Multiple-Choice Knapsack Problem (MCKP) [37], or a *knapsack problem with special ordered sets* [27]. These can be solved more efficiently than arbitrary LPs, a fact that we use in our propagator.

4.1 Solving the Knapsack LP

We obtain an optimal solution $\boldsymbol{x}^*$ of the primal $LP_{F\emptyset}$ by applying Pisinger's greedy algorithm [37]. This gives a $\boldsymbol{x}^*$ in time $O(N \log N)$ [5], with $N = |\boldsymbol{x}^*|$, such that either $\boldsymbol{x}^*$ has no fractional value or it has exactly two fractional values. In the latter case, the WCSP variable $x_k \in \boldsymbol{S}$, verifying $\exists s, s' \in \boldsymbol{D}(x_k)$ such that $0 < x^*_{ks}, x^*_{ks'} < 1$, is called a *split class* and $x_{ks}, x_{ks'}$ are the *split variables.* We denote by $o = \sum_{x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i)} p_{iv} x^*_{iv} - \delta_\emptyset$, the optimal solution cost of $LP_{F\emptyset}$. Consider now the dual of $LP_{F\emptyset}$:

$$\begin{aligned} &\max C \times y_{cc} + \textstyle\sum_{x_i \in \boldsymbol{S}} y_i \\ &\ s.t. \\ &\qquad y_{cc} \times w_{iv} + y_i \leq p_{iv} \ \forall x_i \in \boldsymbol{S}, v \in \boldsymbol{D}(x_i) \\ &\qquad\quad y_{cc} \geq 0 \end{aligned} \tag{6}$$

Where y_{cc} is the dual variable corresponding to the capacity constraint and y_i corresponds to the EO constraint of x_i. From the optimal primal solution, it is easy to compute the optimal dual solution. Let x_k be the split class, $x_{ks}, x_{ks'}$ the split variables and for $i \neq k$, define the variable x_{is} as the variable used in the optimal solution, *i.e.,* $x^*_{is} = 1$.

$$\begin{aligned} y_{cc} &= \frac{p_{ks} - p_{ks'}}{w_{ks} - w_{ks'}} \\ y_k &= p_{ks} - y_{cc} \times w_{ks} = p_{ks'} - y_{cc} \times w_{ks'} \\ y_i &= p_{is} - y_{cc} \times w_{is} \ \forall x_i \in \boldsymbol{S} \setminus \{x_k\} \end{aligned}$$

From the dual solution $\mathbf{y}$, we compute the reduced cost $rc^{\mathbf{y}}(x_{iv})$ of every variable x_{iv}, i.e., the slack of the dual constraint that corresponds to x. When context makes it clear, we omit $\mathbf{y}$ and write $rc(x_{iv})$.

The reduced cost of a variable x can be interpreted as the amount by which we must decrease the coefficient of x in the objective function in order to have

[5] The Dyer-Zemel algorithm [18,52] can compute a solution in O(N) time, but we have not yet implemented it.

$x > 0$ in the optimal solution. We explain later that this implies that we can project some cost to unary cost functions.

In the specific case of $LP_{F\emptyset}$, we have:

$$rc(x_{ks}) = rc(x_{ks'}) = 0$$
$$rc(x_{is}) = 0 \ \forall x_i \in \boldsymbol{S} \setminus \{x_k\}$$
$$rc(x_{iv}) = p_{iv} - y_{cc} \times w_{iv} - y_i \ \forall x_i \in \boldsymbol{S}, v \neq s$$

Observation 1 Consider the linear program $LP'_{F\emptyset}$ which is identical to $LP_{F\emptyset}$ but has $p'_{iv} = p_{iv} - rc(x_{iv})$. Then $opt(LP'_{F\emptyset}) = opt(LP_{F\emptyset})$.

Proof. The optimal solution $\boldsymbol{x}^*$ of $LP_{F\emptyset}$ has the same cost o in $LP_{F\emptyset}$ and $LP'_{F\emptyset}$, as the coefficients of the variables that are greater than 0 are unchanged. The optimal dual solution $\boldsymbol{x}^*$ remains feasible in $LP'_{F\emptyset}$, as the slack in the dual of $LP_{F\emptyset}$ matches exactly the reduction in the right-hand side. Moreover, as the dual objective did not change, it has the same cost and matches the primal cost, so $opt(LP'_{F\emptyset}) = o = opt(LP_{F\emptyset})$. □

Example 2. Consider the following problem:

$$\min 40x_{11} + 55x_{12} + 85x_{13} + 47x_{21} + 95x_{22}$$
$$s.t.$$
$$4x_{11} + 14x_{12} + 24x_{13} + 16x_{21} + 40x_{22} \geq 40$$
$$\sum_{v \in \boldsymbol{D}(x_i)} x_{iv} = 1 \quad \forall x_i \in \{x_1, x_2\}$$
$$0 \leq x_{iv} \leq 1 \quad \forall x_i \in \{x_1, x_2\}, v \in \boldsymbol{D}(x_i)$$

Pisinger's algorithm gives the optimal primal solution $\boldsymbol{x}^* = \{0, 1, 0, \frac{7}{12}, \frac{5}{12}\}$ with cost $o = 55 + \frac{7}{12} \times 47 + \frac{5}{12} \times 95 = 122$.

We deduce the following dual optimal solution : $y_{cc} = 2, y_1 = 55 - 2 \times 14 = 27, y_2 = 47 - 2 \times 16 = 15$.
The following reduced costs are obtained : $rc(x_{12}) = rc(x_{21}) = rc(x_{22}) = 0$ and $rc(x_{11}) = 5$, $rc(x_{13}) = 10$, we deduce that replacing the previous objective function by the following one does not change the cost of the optimal solution:

$$\min 35x_{11} + 55x_{12} + 75x_{13} + 47x_{21} + 95x_{22}$$

We observe that the solution $\boldsymbol{x}^* = \{0, 1, 0, \frac{7}{12}, \frac{5}{12}\}$ is still optimal.

4.2 Propagation

Given a PB constraint and the associated unary costs, it is possible to increase the lower bound by at least $opt(LP_{F\emptyset})$. Our goal is to extend as little cost as possible from the unary cost functions in order to make $opt(LP_{\emptyset}) = o = opt(LP_{F\emptyset})$ and then project o to $c_{\varnothing}$.

Procedure TransformPB($c_{\boldsymbol{S}}, y_{cc}, y_i, o$)

```
Data: $c_{\boldsymbol{S}}$: PB constraint
Data: $y_{cc}, y_i, o$: optimal dual solution of $LP_{F\emptyset}$
1 for all the variables $x_{iv}$ do
2     $c_i(v) \leftarrow c_i(v) - y_{cc} \times w_{iv} - y_i + \delta_{iv}$
3     $\delta_{iv} \leftarrow y_{cc} \times w_{iv} + y_i$
4 $c_\varnothing \leftarrow c_\varnothing + o$
5 $\delta_\varnothing \leftarrow \delta_\varnothing + o$
```

If we move $|c_i(v) - rc(x_{iv})|$ between the constraint and each unary cost function and value, then $opt(LP_\emptyset) = o$ and we can project o to $c_\varnothing$. Indeed we have $|c_i(v) - rc(x_{iv})| = |(y_{cc} \times w_{iv} + y_i) - \delta_{iv}|$, we thus obtain the EPTs performed by Procedure TransformPB.

Theorem 1. *Algorithm TransformPB preserves equivalence.*

Proof. Recall that $p_{iv} = c_i(v) + \delta_{iv}$ and that $rc(x_{iv}) \geq 0$. If $c_i(v) - rc(x_{iv}) \geq 0$ then the cost move is an extension of less than $c_i(v)$, it is valid. If $c_i(v) - rc(x_{iv}) < 0$ then the cost move is a projection, while the cost of any solution $\boldsymbol{x'}$ with $x'_{iv} = 1$ is at least $o - c_i(v) + rc(x_{iv})$. This operation is also valid.

Finally, to check that our sequence of EPTs justifies the increase of $c_\varnothing$ by o, we compute the optimum of $LP_\emptyset$. From Observation 1, $opt(LP_\emptyset) = o$, which means we can project o to $c_\varnothing$ and increase $\delta_\varnothing$ to bring $opt(LP_\emptyset) = opt(LP_{F\emptyset}) = 0$. □

We can improve on this by observing that the integer optimum must be integral. Therefore, we can increase $c_\varnothing$ by $\lceil o \rceil$. In this case, it is also necessary to round up all cost moves. By rounding up, we can no longer rely on Observation 1, but it still holds that $opt(LP_\emptyset) = 0$. We also approach $\emptyset IC$ by verifying that for any value x_{ab} we have $\delta_{ab} + \min \sum_{x_i \in \boldsymbol{S} \setminus x_a, v \in \boldsymbol{D}(x_i)} (\delta_{iv} + c_i(v)) x_{iv} - \delta_\varnothing = 0$. If this is not the case, we can project a positive cost to $c_a(b)$.

Procedure Propagate is the entry point to the propagator. It enforces domain consistency on the PB constraint, then solves $LP_{F\emptyset}$. If there is more than one optimal solution we prefer the one minimizing the reduced cost of the EAC support of each variable. Finally, it uses Procedure TransformPB, to perform cost moves.

Theorem 2. *Procedure Propagate runs in $O(nd \log nd)$ time where n is the number of WCSP variables involved and d the maximum domain size.*

Proof. Pisinger's algorithm dominates the complexity, as it runs in $O(N \log N)$, where N is the number of LP variables. In our case, $N = nd$, so it takes $O(nd \log nd)$ time. Domain consistency on the linear inequality can be performed in linear time. Finally, Procedure TransformPB iterates once over all variables and values and performs constant time operations on each. Hence, the total complexity is $O(nd \log nd)$. □

```
Procedure Propagate($c_{\boldsymbol{S}}$)
  Data: $c_{\boldsymbol{S}}$: PB constraint with EO partitions
1 DomainConsistency($c_{\boldsymbol{S}}$)
2 $(y_{cc}, y_i, o)$ = DualSolve($LP_{F\emptyset}$)
3 TransformPB($c_{\boldsymbol{S}}, y_{cc}, y_i, o$)
```

Example 3. Returning to Example 2, where $c_{\boldsymbol{S}}$ is the PB constraint with EO partitions over two WCSP variables x_1 and x_2, we had the following reduced costs: $rc(x_{12}) = rc(x_{21}) = rc(x_{22}) = 0$, $rc(x_{11}) = 5$, $rc(x_{13}) = 10$, the optimal cost was 122. We deduce the following cost moves:

- $extend(c_1, \{x_1 = 1\}, c_{\boldsymbol{S}}, 35)$
- $extend(c_1, \{x_1 = 2\}, c_{\boldsymbol{S}}, 55)$
- $extend(c_1, \{x_1 = 3\}, c_{\boldsymbol{S}}, 75)$
- $extend(c_2, \{x_2 = 1\}, c_{\boldsymbol{S}}, 47)$
- $extend(c_2, \{x_2 = 2\}, c_{\boldsymbol{S}}, 95)$
- $project(c_{\varnothing}, c_{\boldsymbol{S}}, \emptyset, 122)$

It implies the resulting costs: $\delta_{11} = 35$, $\delta_{12} = 55$, $\delta_{13} = 75$, $\delta_{21} = 47$, $\delta_{22} = 95$, $\delta_{\varnothing} = 122$. The unary costs after these operations are $c_1(2) = c_2(1) = c_2(2) = 0$, $c_1(1) = 5$, $c_1(3) = 10$. If we construct the table of possible assignments of $LP_{\emptyset}$ obtained after the extensions, we can see that $c_{\boldsymbol{S}}(\{x_1 = 1, x_2 = 2\}) = 8$, $c_{\boldsymbol{S}}(\{x_1 = 2, x_2 = 2\}) = 28$, $c_{\boldsymbol{S}}(\{x_1 = 3, x_2 = 1\}) = 0$, $c_{\boldsymbol{S}}(\{x_1 = 3, x_2 = 2\}) = 48$, and all the other assignments don't satisfy the constraint. We observe that the optimal solution is 0, hence our extensions justify the increase of $c_{\varnothing}$.
Now assume that other EPTs outside the PB constraint have modified the unary costs: $c_1(1) \to c_1(1) + 16 = 21$, $c_1(2) \to c_1(2) + 30 = 30$, $c_1(3) \to c_1(3) - 9 = 1$. We want to compute a new lower bound for the PB constraint by solving $LP_{F\emptyset}$:

$$\begin{aligned}\min 56x_{11} + 85x_{12} + 76x_{13} + 47x_{21} + 95x_{22} - 122\\ s.t.\\ 4x_{11} + 14x_{12} + 24x_{13} + 16x_{21} + 40x_{22} \geq 40\\ \sum_{v \in \boldsymbol{D}(x_i)} x_{iv} = 1 \quad \forall x_i \in \{x_1, x_2\}\\ 0 \leq x_{ij} \leq 1 \quad \forall x_i \in \{x_1, x_2\}, v \in \boldsymbol{D}(x_i)\end{aligned}$$

The optimal solution is $\boldsymbol{x}^* = \{0, 0, 1, 1, 0\}$ and its cost is $o = 76 + 47 - 122 = 1$. We deduce the dual optimal solution $y_{cc} = 1$, $y_1 = 52$, $y_2 = 31$ with reduced costs $rc(x_{11}) = rc(x_{13}) = rc(x_{21}) = 0$ and $rc(x_{12}) = 19$, $rc(x_{22}) = 24$. We carry out the following cost moves: $extend(c_1, \{x_1 = 1\}, c_{\boldsymbol{S}}, 21)$, $extend(c_1, \{x_1 = 2\}, c_{\boldsymbol{S}}, 11)$, $extend(c_1, \{x_1 = 3\}, c_{\boldsymbol{S}}, 1)$, $project(c_2, c_{\boldsymbol{S}}, \{x_2 = 2\}, 24)$, $project(c_{\varnothing}, c_{\boldsymbol{S}}, \emptyset, 1)$, with $\delta_{11} = 56$, $\delta_{12} = 66$, $\delta_{13} = 76$, $\delta_{21} = 47$, $\delta_{22} = 71$, $\delta_{\varnothing} = 123$.

5 Experimental results

We implemented our approach in TOULBAR2, an exact WCSP solver in C++,[6] winner of past UAI-2008, 2014 competitions. TOULBAR2 default variable ordering heuristic is the weighted degree heuristic [8], in order to gain information from PB constraints, we adapted an explanation-based weighted degree for linear inequality presented by Hebrard and Siala [24]. For all the tests we imposed a time limit of 30 minutes (except for CPD with 1 hour) on a single core of an Intel Xeon E5-2680 v3 at 2.50 GHz and 256 GB of RAM. We compared our PB propagator with other modeling approaches in protein design. We compared TOULBAR2 to state-of-the-art ILP solver CPLEX 20.1 on knapsack problems with conflict graphs. We also compared TOULBAR2 on pseudo-Boolean Competition 2016 (previously out of reach by TOULBAR2) but the results were not competitive with recent PB solvers (not reported here for the lack of space).

5.1 Sequence of diverse solutions for CPD

A protein is a chain of simple molecules called amino acids. This sequence determines how the protein will *fold* into a specific 3D shape. The Computational Protein Design (CPD) [4] problem consists of identifying the sequence of amino acids that should fold into a given 3D shape. This problem can be modeled as a CFN[7] with unary and binary cost functions representing the energy of the protein but the criteria only approximate the reality, thus producing a sequence of diverse solutions increases the chance of finding the correct real sequence of amino acids. Each time a solution is found, a Hamming distance constraint is added to the model to enforce the next solution to be different from the previous ones. This Hamming distance can be directly encoded as a PB linear constraint, in the form of Eq. (2), with EO partitions associated to domains (Eq. (3)). For each variable, a negative weight of -1 is associated to the value found in the last solution (other values having a zero-weight) and the weighted sum in Eq. (2) must be greater than or equal to $-(|X| - \zeta)$, where ζ corresponds to the required minimum Hamming distance.

This has been implemented in TOULBAR2 and compared to previous automata-based encoding approaches (ternary, hidden, and dual encodings from [40]) on 30 instances [48].[8] Selected instances have from 23 to 97 *residues*/variables with

[6] `https://github.com/toulbar2/toulbar2` version 1.2.

[7] Other paradigms such as ILP or Max-SAT have been tested but the experimental results using their corresponding state-of-the-art solvers were inferior to the CFN approach using TOULBAR2 [4,1]. E.g., for CPD instance *1BK2.matrix.24p.17aa.usingEref_self_digit2* ($n = 24$, $d = 182$, $e = 300$), CPLEX 20.1 solves it in 42.84 seconds, TOULBAR2 in 0.37 seconds. ROUNDINGSAT [26] timed out after 10 hours.

[8] `http://genoweb.toulouse.inra.fr/~tschiex/CPD-AIJ/Last35-instances`. We removed 5 instances (*1ENH.matrix.36p.17aa, 1STN.matrix.120p.18aa, HHR.matrix.115p.19aa, 1PGB.matrix.31p.17aa, 2CI2.matrix.51p.18aa*) on which TOULBAR2 timed out after 9,000 seconds even without diversity constraints [1].

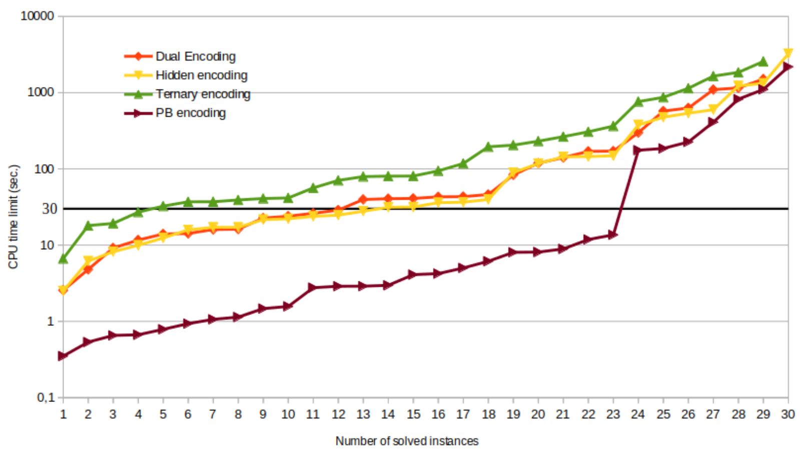

Fig. 1. Cactus plot of CPU solving time (log scale) for different encodings of Hamming distance constraints on CPD.

maximum domain size going from 48 to 194 *rotamers*/values. The number of unary and binary cost functions goes from 276 to 4, 753. For each instance, the time limit was 1 hour and the solver halts after finding a greedy sequence of 10 diverse solutions, the Hamming distance is $\zeta = 10$, and we enforce VAC on unary and binary cost functions in preprocessing (options -A -d: -a=10 -div=10 -divm=(0 for dual, 1 for hidden, 2 for ternary, and 3 for the PB encoding)). Figure 1 reports the solving time of each encoding. The dual and ternary encodings failed to give 10 diverse solutions for one instance, while the PB and hidden encodings didn't. Moreover the PB encoding is faster for 29 instances and it solves 23 of them in less than 30 seconds while dual, hidden, and ternary encodings solve respectively 12, 13, and 4 instances in less than 30 seconds. Note that we are computing a greedy sequence of solutions, the different encodings do not return the exact same sequence (except for 7/30 instances). We also compared for each instance the number of backtracks and time (not reported here) of the previous TOULBAR2 default encoding (dual) with the PB encoding. In all the instances the PB encoding needs fewer backtracks than the dual encoding and except for one instance, the PB encoding is also faster. Automata-based encodings have the flaw of introducing extra variables that can disturb the variable ordering heuristic (by default, *min domain size over weighted degree* [9])[9] and local consistency algorithm (by default, EDAC during search, except partial F∅IC for PB constraints). While the PB encoding directly encodes the Hamming distance, it is heavier to propagate as we can see by comparing the number of backtracks per second (170 for PB encoding and 1060 for dual encoding).

[9] Additionally, the PB constraint provides finer-grain weights using explanations [24] when linear coefficients are not all equal as it is the case in the KPCG benchmark.

	TB2	CPLEX$_t$	CPLEX$_d$	TB2	CPLEX$_t$	CPLEX$_d$
C1	718	689	**720**	**720**	701	**720**
C3	597	487	**614**	**718**	513	639
C10	**490**	318	457	**633**	346	547
R1	**720**	705	**720**	**720**	705	**720**
R3	**720**	573	682	**720**	589	691
R10	**571**	365	519	**665**	384	583

Table 1. Number of solved instances (left) and number of times a solver found the best solution within the time limit (right) for six different classes of KPCG.

5.2 Knapsack problem with a conflict graph

We compare here TOULBAR2 and CPLEX on Knapsack with Conflict Graph (KPCG) [6,11], a knapsack problem combined with binary constraints representing conflicts between pairs of variables. We use 6 different classes $C1$,$C3$,$C10$,$R1$, $R3$,$R10$. In three of them the weight and the profit of each variable are correlated (class C) otherwise the profit is random between $[1, 100]$ (class R). The numbers $1, 3, 10$ correspond to a multiplying coefficient of the capacity, which has the effect of making the instances harder as the multiplier increases. In each class half of the instances have capacity 150, weights are uniformly distributed in $[20, 100]$, and the number of Boolean variables varies between $120, 250, 500$, and 1000. For the other half, the capacity is 1000, weights are uniformly distributed in $[250, 500]$, and the number of Boolean variables varies between $60, 120, 349$, and 501. Additionally, the density of the conflict graph varies from 0.1 to 0.9. In total, each class has 720 instances. We used a direct encoding for TOULBAR2. For CPLEX, we tried with both tuple and direct encodings (tuple encoding corresponds to the local polytope with integer variables) [25]. Table 1 reports the number of instances solved by each solver. TOULBAR2 was more efficient than CPLEX with the tuple encoding and competitive with CPLEX using the direct encoding for four out of six classes. Moreover, TOULBAR2 finds the best solutions for the largest number of instances in every class.

6 Conclusion and future work

It is now possible to model pseudo-Boolean linear constraints in deterministic and probabilistic graphical models. This provides greater modeling flexibility and allows a WCSP solver like TOULBAR2 to solve more problems, such as computational protein design problems with diversity guarantee or knapsack problems with conflict graphs. One of the weaknesses of our approach is that the algorithm fundamentally produces a suboptimal solution to the linear program, because it propagates the pseudo-Boolean linear constraints one by one and does not take into account other constraints (except at-most-one constraints). There are several ways to improve this, including adapting work previously done in this context on Lagrangian relaxation [30] or an approach closer to VAC [16]. It also opens up possibilities for other uses of linear constraints in the WCSP framework, such as the generation of cuts.

References

1. Allouche, D., Barbe, S., de Givry, S., Katsirelos, G., Lebbah, Y., Loudni, S., Ouali, A., Schiex, T., Simoncini, D., Zytnicki, M.: Operations Research and Simulation in Healthcare, chap. Cost Function Networks to Solve Large Computational Protein Design Problems. Springer (2021)
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. Allouche, D., Bessière, C., Boizumault, P., de Givry, S., Gutierrez, P., Loudni, S., Metivier, J.P., Schiex, T.: Filtering decomposable global cost functions. In: Proc. of AAAI-12. Toronto, Canada (2012)
4. Allouche, D., Davies, J., de Givry, S., Katsirelos, G., Schiex, T., Traoré, S., André, I., Barbe, S., Prestwich, S., O'Sullivan, B.: Computational protein design as an optimization problem. Artificial Intelligence **212**, 59–79 (2014)
5. Ansótegui, C., Bofill, M., Coll, J., Dang, N., Esteban, J.L., Miguel, I., Nightingale, P., Salamon, A.Z., Suy, J., Villaret, M.: Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints. In: Proc. of CP-19. pp. 20–36. Stamford, CT, USA (2019)
6. Bettinelli, A., Cacchiani, V., Malaguti, E.: A branch-and-bound algorithm for the knapsack problem with conflict graph. INFORMS Journal on Computing **29**(3), 457–473 (2017)
7. Bofill, M., Coll, J., Suy, J., Villaret, M.: An mdd-based SAT encoding for pseudo-boolean constraints with at-most-one relations. Artif. Intell. Rev. **53**(7), 5157–5188 (2020)
8. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: ECAI. vol. 16, p. 146 (2004)
9. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proc. of ECAI-04. vol. 16, p. 146 (2004)
10. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio Link Frequency Assignment. Constraints **4**(1), 79–89 (1999)
11. Coniglio, S., Furini, F., San Segundo, P.: A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. European Journal of Operational Research **289**(2), 435–455 (2021)
12. Cooper, M.C., de Givry, S., Sánchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence **174**(7-8), 449–478 (2010)
13. Cooper, M.C., de Givry, S., Schiex, T.: Graphical models: Queries, complexity, algorithms (tutorial). In: Proc. of 37th International Symposium on Theoretical Aspects of Computer Science (STACS-20). LIPIcs, vol. 154, pp. 4:1–4:22. Montpellier, France (2020)
14. Cooper, M.C., de Givry, S., Schiex, T.: Valued Constraint Satisfaction Problems, pp. 185–207. Springer International Publishing (2020)
15. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM (JACM) **50**(2), 107–153 (2003)
16. Dlask, T., Werner, T.: Bounding linear programs by constraint propagation: Application to Max-SAT. In: Proc. of CP-20. pp. 177–193. Louvain-la-Neuve, Belgium (2020)
17. Dlask, T., Werner, T.: On relation between constraint propagation and block-coordinate descent in linear programs. In: Proc. of CP-20. pp. 194–210. Louvain-la-Neuve, Belgium (2020)

18. Dyer, M.E.: An $o(n)$ algorithm for the multiple-choice knapsack linear program. Mathematical programming **29**(1), 57–63 (1984)
19. Elffers, J., Nordström, J.: Divide and conquer: Towards faster pseudo-boolean solving. In: Proc. of IJCAI. pp. 1291–1299. Stockholm, Sweden (2018)
20. 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)
21. de Givry, S., Heras, F., Zytnicki, M., 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)
22. de Givry, S., Katsirelos, G.: Clique cuts in weighted constraint satisfaction. In: Proc. of CP-17. pp. 97–113. Melbourne, Australia (2017)
23. Haller, S., Swoboda, P., Savchynskyy, B.: Exact map-inference by confining combinatorial search with LP relaxation. In: Proc. of AAAI-18. pp. 6581–6588. New Orleans, Louisiana, USA (2018)
24. Hebrard, E., Siala, M.: Explanation-based weighted degree. In: Proceedings of CPAIOR 2017. pp. 167–175 (2017)
25. 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)
26. Jo Devriendt, A.G., Nordström, J.: Learn to relax: Integrating 0-1 integer linear programming with pseudo-boolean conflict-driven search. In: Proc. of CP-AI-OR'2020. Vienna, Austria (2020)
27. Johnson, E.L., Padberg, M.W.: A note of the knapsack problem with special ordered sets. Operations Research Letters **1**(1), 18–22 (1981)
28. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
29. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. IEEE transactions on pattern analysis and machine intelligence **28**(10), 1568–1583 (2006)
30. Komodakis, N., Paragios, N., Tziritas, G.: MRF energy minimization and beyond via dual decomposition. IEEE transactions on pattern analysis and machine intelligence **33**(3), 531–552 (2010)
31. Larrosa, J.: On arc and node consistency in weighted CSP. In: Proc. AAAI'02. pp. 48–53. Edmondton, (CA) (2002)
32. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for weighted CSP. In: Proc. of IJCAI-03. vol. 3, pp. 239–244 (2003)
33. Lee, J.H.M., Leung, K.L.: Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. Journal of Artificial Intelligence Research **43**, 257–292 (2012)
34. Lee, J.H., Leung, K.L., Shum, Y.W.: Consistency techniques for polytime linear global cost functions in weighted constraint satisfaction. Constraints **19**(3), 270–308 (2014)
35. Lee, J.H., Leung, K.L., Wu, Y.: Polynomially decomposable global cost functions in weighted constraint satisfaction. In: Proc. of AAAI-12. Toronto, Canada (2012)
36. Marinescu, R., Dechter, R.: And/or branch-and-bound for graphical models. In: Proc. of IJCAI-05. pp. 224–229. Edinburgh, Scotland (2005)
37. Pisinger, D., Toth, P.: Knapsack problems. In: Handbook of combinatorial optimization, pp. 299–428. Springer (1998)
38. Prusa, D., Werner, T.: Universality of the local marginal polytope. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1738–1743 (2013)

39. Ruffini, M., Vucinic, J., de Givry, S., Katsirelos, G., Barbe, S., Schiex, T.: Guaranteed diversity & quality for the weighted CSP. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI). pp. 18–25. IEEE (2019)
40. Ruffini, M., Vucinic, J., de Givry, S., Katsirelos, G., Barbe, S., Schiex, T.: Guaranteed diversity and optimality in cost function network based computational protein design methods. Algorithms **4**(6:168) (2021)
41. Sakai, M., Nabeshima, H.: Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. IEICE TRANSACTIONS on Information and Systems **98**(6), 1121–1127 (2015)
42. Sánchez, M., de Givry, S., Schiex, T.: Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. Constraints **13**(1), 130–154 (2008)
43. Schiex, T.: Arc consistency for soft constraints. In: Proc. of CP-00. pp. 411–424. Singapore (2000)
44. Schlesinger, M.: Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (Syntactic analysis of two-dimensional visual signals in noisy conditions). Kibernetika **4**, 113–130 (1976)
45. Sontag, D., Choe, D., Li, Y.: Efficiently searching for frustrated cycles in MAP inference. In: Proc. of UAI. pp. 795–804. Catalina Island, CA, USA (2012)
46. 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. Helsinki, Finland (2008)
47. Tourani, S., Shekhovtsov, A., Rother, C., Savchynskyy, B.: Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In: Proc. of AISTATS-20. pp. 2775–2785. Palermo, Sicily, Italy (2020)
48. Traoré, S., Allouche, D., André, I., de Givry, S., Katsirelos, G., Schiex, T., Barbe, S.: A new framework for computational protein design through cost function network optimization. Bioinformatics **29**(17), 2129–2136 (2013)
49. Trösser, F., de Givry, S., Katsirelos, G.: Relaxation-aware heuristics for exact optimization in graphical models. In: Proc. of CP-AI-OR'2020. pp. 475–491. Vienna, Austria (2020)
50. Vaidya, P.: Speeding-up linear programming using fast matrix multiplication. In: 30th Annual Symposium on Foundations of Computer Science. pp. 332–337 (1989)
51. Werner, T.: A Linear Programming Approach to Max-sum Problem: A Review. IEEE Trans. on Pattern Recognition and Machine Intelligence **29**(7), 1165–1179 (Jul 2007)
52. Zemel, E.: An $o(n)$ algorithm for the linear multiple choice knapsack problem and related problems. Information Processing Letters **18**(3), 123–128 (1984)
53. Zytnicki, M., Gaspin, C., de Givry, S., Schiex, T.: Bounds Arc Consistency for Weighted CSPs. Journal of Artificial Intelligence Research **35**, 593–621 (2009)