Bi-Objective Discrete Graphical Model Optimization

Samuel Buchet[1], David Allouche[1], Simon de Givry[1], and Thomas Schiex[1]

Universite Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France
`{samuel.buchet,david.allouche,simon.de-givry,thomas.schiex}@inrae.fr`

Abstract. Discrete Graphical Models (GMs) are widely used in Artificial Intelligence to describe complex systems through a joint function of interest. Probabilistic GMs such as Markov Random Fields (MRFs) define a joint non-normalized probability distribution while deterministic GMs such as Cost Function Networks (CFNs) define a joint cost function. A typical query on GMs consists in finding the joint state that optimizes this joint function, a problem denoted as the *Maximum a Posteriori* or Weighted Constraint Satisfaction Problem respectively.
In practice, more than one function of interest may need to be optimized at the same time. In this paper, we develop a two-phase scalarization method for solving bi-objective discrete graphical model optimization, with the aim of computing a set of non-dominated solutions — the Pareto frontier — representing different compromises between two GM-defined objectives. For this purpose, we introduce a dedicated higher-order constraint, which bounds the value of one GM-defined objective while minimizing another GM on the same variables. Discrete GM optimization is NP-hard, and its bi-objective variants are even harder. We show how existing GM global lower and upper bounds can be exploited to provide anytime bounds of the exact Pareto frontier. We benchmark it on various instances from Operations Research and Probabilistic Graphical Models.

Keywords: bi-objective combinatorial optimization · graphical model · constraint optimization · cost function network · exact methods.

1 Introduction

Many real problems require considering more than one objective. In protein design [2], one may look for the most probable (minimum energy) amino acid sequence given a target structure that also minimizes the complexity of the sequence [43], or which also optimizes its probability given another structure [42]. The same interest for multi-objective optimization exists in drug design [31, 27]. In the uncapacitated warehouse location problem, it is desirable to minimize the setup costs of all facilities while minimizing the serving costs from these facilities [25]. In frequency assignment problems, one typically wants to minimize the number of different frequencies used but also their span (the maximum difference in frequencies) [10]. In all these cases, one objective can be compactly represented as one Graphical Model [14]. But the existence of multiple objectives makes discrete optimization challenging [20]. Even classical polytime problems such as the shortest path or assignment problems become NP-hard (to find a non-dominated solution) or even possibly #P-hard, when a set of states having non-dominated costs (the so-called Pareto frontier) is sought [16].

The hardness of these problems means that they often cannot be exhaustively solved. In a single objective minimization case, most algorithms will deliver an incumbent solution and a global lower bound, providing a desirable *a posteriori* optimality gap. In this paper, we are interested in providing similar services when two objectives, each defined by a Graphical Model, need to be simultaneously optimized under hard constraints. To exploit the efficiency of existing GM solvers [3], we reduce the problem of computing non-dominated states to a series of single GM optimization problems.

Our approach lies in the family of two-phase methods [40, 41, 16], where a first phase identifies all the solutions that can be found by solving a linear combination of both objectives (called *supported* non-dominated states). This phase reduces to a dichotomic series of optimization of a single objective combining the two original objectives linearly [4] that can can be directly solved by any discrete GM optimization solver. To identify non-supported solutions that may exist inside the convex envelope of supported solutions, a second phase requires to optimize one objective while the other is bounded, an approach usually denoted as the ε-constraint method [16]. In our algorithm, for a proper anytime behavior, every single objective resolution is bounded in CPU time, as is the maximum number of non-dominated states produced. In its first phase, the algorithm produces both incumbent solutions defining an upper-bounding set and a set of lower-bounding half-spaces. When this first phase finishes, the second phase enumerates non-supported solutions as well as additional lower-bounding rectangular regions. These two sets offer both incumbent solutions (in the upper bounding set) as well as an optimality gap that can be represented graphically and computed as a ratio of surfaces.

2 Background and notations

Definition 1. *A Bi-Objective GM (BO-GM) N is a tuple $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} = \boldsymbol{F_1} \bigcup \boldsymbol{F_2})$ where $\boldsymbol{X}$ is the set of variables, $\boldsymbol{D}$ the finite domains of each variable and $\boldsymbol{F}$ the set of potential functions. Each function $f_{\boldsymbol{S}} \in \boldsymbol{F}$ associates a cost c to every assignment of the variables in its scope $\boldsymbol{S} \subseteq \boldsymbol{X}$. $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ define the two joint functions (objectives) $F_i(\boldsymbol{x}) = \sum_{f_{\boldsymbol{S}} \in \boldsymbol{F_i}} f_{\boldsymbol{S}}(\boldsymbol{x}|_{\boldsymbol{S}})$ where $\boldsymbol{x}$ is a complete assignment of $\boldsymbol{X}$ and $\boldsymbol{x}|_{\boldsymbol{S}}$ denotes the projection of $\boldsymbol{x}$ to variables $\boldsymbol{S}$.*

We assume that the functions $f_{\boldsymbol{S}} \in \boldsymbol{F}$ may take infinite values, representing forbidden states. In probabilistic GMs such as Markov Random Fields, a joint function $F_i(\boldsymbol{x})$ is used to define a joint probability distribution $p_i(\boldsymbol{X} = \boldsymbol{x}) \propto \exp(-F_i(\boldsymbol{x}))$ and the cost $F_i(\boldsymbol{x})$ is called an *energy*. Minimizing $F_i(\boldsymbol{x})$ is equivalent to maximizing $p_i(\boldsymbol{x})$ (infinite energies define zero probabilities).

For any state $\boldsymbol{x}$, we denote by $\ddot{F}(\boldsymbol{x})$ the pair of costs $(F_1(\boldsymbol{x}), F_2(\boldsymbol{x}))$. Given two pairs of costs or bi-costs $c = (c_1, c_2)$ and $c' = (c'_1, c'_2)$, we say that c weakly dominates c', denoted as $c \preccurlyeq c'$, iff $\forall i, c_i \leq c'_i$. c dominates c', denoted as $c \prec c'$ iff $c \preccurlyeq c'$ and $\exists i, c_i < c'_i$. This can be extended to states where $\boldsymbol{x} \prec \boldsymbol{x}'$ iff $\ddot{F}(\boldsymbol{x}) \prec \ddot{F}(\boldsymbol{x}')$. We denote by $\boldsymbol{P}$ the set of all states with non-dominated bi-costs or efficient assignments. The image $\ddot{F}(\boldsymbol{P})$ is a set of non-dominated bi-costs of states called the Pareto frontier. An efficient state $\boldsymbol{x}$ such that $\ddot{F}(\boldsymbol{x}) = (c_1, c_2)$ is said to be supported if $\exists \lambda_1, \lambda_2 \in {\mathbb{R}^+}^2$ such that $\forall c' \in \ddot{F}(\boldsymbol{P}), \lambda_1 c_1 + \lambda_2 c_2 \leq \lambda_1 c'_1 + \lambda_2 c'_2$. The bi-costs of supported efficient assignments are known to lie on the frontier of the convex envelope of $F(\boldsymbol{P})$. They are said to be extreme when located at extreme points of the convex envelope. On the other hand, non-supported efficient assignments have costs in the interior of this convex envelope.

The Bi-Objective GM Optimization Problem (BO-GMO) is to find a set $\boldsymbol{E}$ of efficient states such that $\ddot{F}(\boldsymbol{E}) = \ddot{F}(\boldsymbol{P})$. In the worst case, the set $\ddot{F}(\boldsymbol{P})$ (and therefore $\boldsymbol{E}$) can have a size that grows exponentially with the number of variables. Finding a single efficient state is NP-hard while counting the number of elements in $\ddot{F}(\boldsymbol{P})$ is $\#P$-hard [14, 16]. Exploring the complete Pareto front is therefore challenging and quickly infeasible. However, as in single GM optimization, the Pareto front can be approximated by a lower and upper bound. The global ideal cost, defined as the pair $\iota = (\min_{\boldsymbol{x} \in \boldsymbol{P}} F_1(\boldsymbol{x}), \min_{\boldsymbol{x} \in \boldsymbol{P}} F_2(\boldsymbol{x}))$ defines a bi-cost that weakly dominates the bi-cost of any state.

More generally, we define a lower-bound set $\boldsymbol{L}$ as any closed subset of $\mathbb{R}^2$ such that all states have a cost that is weakly dominated by some element of $\boldsymbol{L}$ and is dominated by some element of the interior of $\boldsymbol{L}$. By definition, the union of two lower-bound sets is also a lower-bound set. In this paper, we consider lower-bound sets defined as the union of simple polygonal regions (half-spaces or rectangles). An upper bound set is a set $\boldsymbol{U}$ of bi-costs such that any efficient state weakly dominates some element of $\boldsymbol{U}$.

3 A Two-phase method for Bi-Objective GM Optimization

Multi-objective optimization problems can be reduced to a series of single-objective optimization problems, using so-called scalarization techniques. As our method follows the two-phase scheme, it relies on two types of scalarizations. The overall search algorithm is described as Algorithm 1 and its different phases are outlined in Figure 1 and presented in detail below.

3.1 First phase

In the first phase, we focus on linear scalarization, which consists of aggregating objectives into a weighted sum. This technique ensures that the optimal solution of the resulting problem is a supported efficient solution for the original problem [16]. Moreover, as the linearly scalarized problem corresponds precisely to a GM, it can be solved by ready-to-use solvers. We assume we have a discrete

GM optimizer which is called by Solve $(N, \top, t)$ where N is a GM, $\top$ is a global upper bound (the solver will only report solutions of cost lower than $\top$) and t is a time-limit. In all cases, the solver returns a pair $(\boldsymbol{x}, lb)$ where $\boldsymbol{x}$ is the best state identified in the time budget and lb is a global lower bound on the joint function defined by N. The difference in cost between lb and the cost of $\boldsymbol{x}$ defines a possibly non-zero optimality gap. When there is provably no solutions, the solver returns $\boldsymbol{x} = \varnothing$ and $lb = \top$.

Definition 2. *Given a BO-GM* $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} = \boldsymbol{F_1} \bigcup \boldsymbol{F_2})$ *and two multipliers* $\lambda_1, \lambda_2 \in \mathbb{R}^2$*, its* (λ_1, λ_2)*-scalarized GM is a GM* $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} = \lambda_1 \boldsymbol{F_1} + \lambda_2 \boldsymbol{F_2})$ *with a single objective* $F = \lambda_1 \sum_{f_S \in \boldsymbol{F_1}} f_{\boldsymbol{S}} + \lambda_2 \sum_{f_S \in \boldsymbol{F_2}} f_{\boldsymbol{S}}$.

When a linear scalarization of a given GM is solved, the single-objective lower bound returned by the solver defines a half-space lower bound set over feasible bi-costs, guaranteed to contain no feasible bi-cost. Let $\lambda_1, \lambda_2 \in \mathbb{R}^2$ and lb be a lower bound on the optimum of the (λ_1, λ_2)-scalarized GM, we know that $lb \le \lambda_1 F_1(\boldsymbol{x}) + \lambda_2 F_2(\boldsymbol{x})$ and the half-space $\lambda_1 c_1 + \lambda_2 c_2 \le lb$ defines a lower bound set.

To identify supported efficient states, we use the dichotomic enumeration approach [4]. We maintain a heap Q_1 of lexicographically sorted pairs of supported states. Q_1 is initialized with a pair $(\boldsymbol{x_1^*}, \boldsymbol{x_2^*})$ containing extreme states that respectively minimize F_1 and F_2. At each iteration, we extract a pair of supported states $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ and check if a new improved (extreme) supported state between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ exists (see Figure 1(a)) by solving the (λ_1, λ_2)-scalarization of the BO-GM for $\lambda_1 = F_2(\boldsymbol{x_1}) - F_2(\boldsymbol{x_2})$ and $\lambda_2 = F_1(\boldsymbol{x_2}) - F_1(\boldsymbol{x_1})$ with an upper bound of $\lambda_1 F_1(\boldsymbol{x_1}) + \lambda_2 F_2(\boldsymbol{x_1})$. To focus the exploration on the sparsest regions of the Pareto frontier, we choose $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ such that $||F(\boldsymbol{x}_1) - F(\boldsymbol{x}_2)||$ is maximum. An optimal solution $\boldsymbol{x}$ of the problem, if any, results in new supported pairs $(\boldsymbol{x}_1, \boldsymbol{x})$ and $(\boldsymbol{x}, \boldsymbol{x}_2)$ added in Q_1 for further exploration (Figure 1(b)). Whether the new state is efficient or not, it is added in $\boldsymbol{U}$ as part of the upper bound set. However, when there is provably no solution, $(\boldsymbol{x_1}, \boldsymbol{x_2})$ is stored for phase 2. In all cases, the lower bound l found together with the multipliers λ_1, λ_2 represent a lower bounding half-space which is stored as a triple $\langle \lambda_1, \lambda_2, l \rangle$ in a growing list $\boldsymbol{L_1}$ (Figure 1(c)). The algorithm proceeds until Q_1 is emptied. If all sub-problems have been solved to optimality, the convex envelope of $\boldsymbol{P}$ is determined.

3.2 Second phase

The linear scalarization method, however, cannot identify non-supported efficient states and may potentially miss supported non-extreme efficient solutions. In two-phase algorithms, starting from the pairs of supported efficient states pushed in $\boldsymbol{Q_2}$ during the first phase, the second phase explores all pairs to identify possibly non-supported states that may exist in between. This requires restricting the search to a polygonal region where we can optimize one objective (say F_1) and bound the other objective F_2 to reside between $F_2(\boldsymbol{x_2})$ and $F_2(\boldsymbol{x_1})$. Additionally, an upper bound requires F_1 not to be worse than $F_1(\boldsymbol{x_2})$. Contrarily to the first phase, a lower bound l on the optimum of this constrained problem

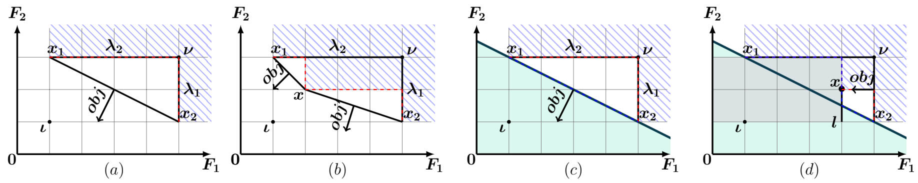

Fig. 1. (a, left) In phase 1, given two efficient solutions (x_1, x_2), $obj = \lambda_1 F_1 + \lambda_2 F_2$ is minimized. (b, center-left) If a supported efficient solution x is found, one recursively considers two new subproblems given by (x_1, x) and (x, x_2). (c, center-right) if no solution is found, the half-space region depicted in light cyan defines a lower-bounding space. (d, right) In phase 2, after (c), objective F_1 is minimized with a bounding constraint on F_2 (*i.e.,* efficient solutions are searched inside the rectangle given by x_1, x_2 and their nadir point ν). Any lower bound l of this problem defines a forbidden rectangle region (light gray). The union of half-space and rectangle regions defines a lower bounding set. If an efficient solution x is found, then the method explores the subproblem given by (x, x_2). Each solution found defines a region of dominated solutions.

does not define a lower bounding half-space. It instead excludes all bi-costs in the interior of the rectangle defined by $0 \leq c_1 \leq l$ and $F_2(\boldsymbol{x_2}) \leq c_2 \leq F_2(\boldsymbol{x_1})$. This region is added as a triple $(l, F_2(\boldsymbol{x_2}), F_2(\boldsymbol{x_1}))$ to the set $\boldsymbol{L_2}$ of second-phase lower-bounding regions. The new solution $\boldsymbol{x}$, if any, is added to $\boldsymbol{U}$. Moreover, similarly to phase 1, when optimality has been proven, the remaining interval pair $(\boldsymbol{x}, \boldsymbol{x}_2)$ is inserted in the list Q_2 of pairs of efficient states for exhaustive search (Figure 1(d)).

In the end, the union of the lower-bounding regions in $\boldsymbol{L}$ defines a lower-bounding polygon. Similarly, each upper-bound set defined by a bi-cost $\ddot{F}(u)$ with $u \in \boldsymbol{U}$ excludes a top-right quadrant with a corner in $\ddot{F}(u)$ and their union defines an upper-bounding polygon. All our polygons are quadrangles, and their union can be computed in time $O(n \log n)$ where n is the number of regions [33, Chapter 7]. The surfaces $s_{\boldsymbol{L}}$ and $s_{\boldsymbol{U}}$ of the lower and upper bounding regions can be computed using the Shoelace formula [9]. Together with the overall surface s_o of the rectangle defined by the global ideal point with coordinates $(F_1(\boldsymbol{x_1^*}), F_2(\boldsymbol{x_2^*}))$ and the opposite $(F_1(\boldsymbol{x_2^*}), F_2(\boldsymbol{x_1^*}))$ they define a Pareto optimality gap as $1 - \frac{s_{\boldsymbol{L}} + s_{\boldsymbol{U}}}{s_o}$. Eventually, the states in U with a bi-cost that lies on the frontier defined by s_L are known to be efficient.

Theorem 1. *Algorithm 1 is correct and has a bounded time complexity.*

Proof. Correctness follows from the exactly identified lower bounding regions, found by an exact solving method (Solve) in phases 1 and 2. The algorithm outputs a guaranteed Pareto optimality gap in time bounded by $m \times t$ (input parameters m and t).

```
Function TwoPhase($N, t, m$)
       $L_1 := \varnothing$; $L_2 := \varnothing$; $U := \varnothing$; $Q_1 := \varnothing$; $Q_2 := \varnothing$;
       /* Solve $F_1$ within time $t$, store upper & lower bound */
1      $\boldsymbol{x_1^*}, l_1 :=$ Solve($(X, D, F_1), \infty, t$);
2      $L_1$.push($\langle 1, 0, l_1 \rangle$);
3      $U$.push($\boldsymbol{x_1^*}$);
       /* Solve $F_2$ within time $t$, store upper & lower bound */
4      $\boldsymbol{x_2^*}, l_2 :=$ Solve($(X, D, F_2), \infty, t$);
5      $L_1$.push($\langle 0, 1, l_2 \rangle$);
6      $U$.push($\boldsymbol{x_2^*}$);
       /* Proceed if $F_1$ and $F_2$ have been solved to optimality */
       if $\boldsymbol{x_1^*} \neq \varnothing \wedge \boldsymbol{x_2^*} \neq \varnothing \wedge l_1 = F_1(\boldsymbol{x_1^*}) \wedge l_2 = F_2(\boldsymbol{x_2^*})$ then
           $Q_1 := \{(\boldsymbol{x_1^*}, \boldsymbol{x_2^*})\}$
7      while $Q_1 \neq \varnothing$ and $|L_1| \leq m$ do
           /* Bisect by $(\lambda_1, \lambda_2)$-scalarization and store bounds */
           $(x_1, x_2) :=$ pop-best($Q_1$);
           $\lambda_1 := F_2(x_1) - F_2(x_2)$; $\lambda_2 := F_1(x_2) - F_1(x_1)$;
           $\top := \lambda_1 F_1(x_1) + \lambda_2 F_2(x_1)$ ;
8          $x, l :=$ Solve($(X, D, (\lambda_1 F_1 + \lambda_2 F_2)), \top, t$);
9          $L_1$.push($\langle \lambda_1, \lambda_2, l \rangle$) ;
10         $U$.push($x$) ;
           /* Push in $Q_1$ only if solved to optimality */
           if $x \neq \varnothing$ and $l = \lambda_1 F_1(x) + \lambda_2 F_2(x)$ then
11             $Q_1$.push($(x_1, x)$) ; $Q_1$.push($(x, x_2)$) ;
           /* Push in $Q_2$ when there is provably no solution */
           if $x = \varnothing$ and $l = \top$ then
12             $Q_2$.push($(x_1, x_2)$)
       /* Phase 2 if all phase 1 problems solved to optimality */
13     if $|Q_2| \neq |U| - 1$ then return $L_1, L_2, U$;
14     while $Q_2 \neq \varnothing$ and $|L_1| + |L_2| \leq m$ do
           $(x_1, x_2) :=$ pop-best($Q_2$);
           /* Optimize $F_1$ with a bounding constraint on $F_2$ */
15         $x, l :=$ Solve($(X, D, F_1 \bigcup (F_2(x_2) + 1 \leq F_2 < F_2(x_1))), F_1(x_2), t$) ;
16         $L_2$.push($\langle l, F_2(x_2), F_2(x_1) \rangle$) ;
17         $U$.push($x$) ;
18         if $x \neq \varnothing$ and $l = F_1(x)$ then $Q_2$.push($(x, x_2)$);
       return $L_1, L_2, U$
```

Algorithm 1: Two-phase method. N is the BO-GM to optimize, t is a maximum CPU-time for any execution of **Solve** and m the maximum number of calls to **Solve**.

4 The higher-order GM bounding constraint

To optimize $\boldsymbol{F}_1$ while constraining $\boldsymbol{F}_2$, a dedicated bounding constraint needs to be added to the discrete GM solver. The corresponding higher-order GM bounding *hard* constraint takes as input a GM $N = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F})$ and two costs lb and ub such that $lb < ub$. It enforces $lb \leq F(\boldsymbol{x}) < ub$ for all assignments $\boldsymbol{x}$ of $\boldsymbol{X}$.

To enforce this constraint during branch and bound search, since solving N would be NP-hard, we rely on polynomial time convergent message passing algorithms, also known as equivalence preserving soft local consistencies [13, 14]. We use soft local consistencies such as EDAC [19, 13]. When enforced on a GM N, soft arc consistencies produce a lower-bound $lb_N(x_i, a)$ on the joint cost F if variable x_i is assigned state a, for every state a of every variable x_i. To enforce $F(\boldsymbol{x}) < ub$, one can simply prune any state a for any variable x_i such that $lb_N(x_i, a) \geq ub$. Since these lower bounds eventually become exact on fully assigned GMs, this guarantees that no solution violating $F(\boldsymbol{x}) < ub$ will ever be produced while pruning branches as soon as lower bounds allow for.

To enforce $lb \leq F(\boldsymbol{x})$, we build the GM denoted as $-N = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{-F})$ simply by taking the opposite *finite* costs in every cost function, *i.e.,* $\forall f_S \in \boldsymbol{F}$ of N, we have $f'_S \in \boldsymbol{-F}$ in $-N$ such that $\forall \boldsymbol{x} : f'_S(\boldsymbol{x}) = -f_S(\boldsymbol{x})$ if $f_S(\boldsymbol{x}) < \infty$ else $f'_S(\boldsymbol{x}) = \infty$. To enforce $lb \leq F(\boldsymbol{x})$, we prune a state a for a variable x_i as soon as the lower bound $lb_{-N}(x_i, a) \geq -lb$. For the same reason as above, this guarantees that no solution violating $lb \leq F(\boldsymbol{x})$ will ever be produced while pruning branches.

Interestingly, if N does not contain any infinite cost ($\forall f_S \in \boldsymbol{F}, f_S(\boldsymbol{x}) < \infty$), if the current lower bound of N is greater than lb and the current lower bound of $-N$ is strictly greater than $-ub$ then the higher-order GM bounding constraint is always satisfied and can be ignored. This is checked after every variable assignment during branch and bound search. Notice that any state removal in the subproblem N (resp. $-N$) is informed/channeled in $-N$ (resp. N) and in the main problem optimizing F_1 and *vice versa.* By doing so, we synchronize domains between the different GMs sharing the same variables.

When adding the higher-order GM bounding constraint, after having tried to decompose all non-unary cost functions into a sum of unary cost functions [17], we check if N contains only unary cost functions. In this case, the objective is linear and the higher-order GM bounding constraint can be replaced by two generalized linear constraints, without introducing any extra variables [30].

For variable ordering heuristics, we use the weighted-degree heuristic $\frac{dom}{wdeg}$ [8]. For each higher-order GM bounding constraint, we maintain counters of conflicts for all the non-unary cost functions inside the sub-problems N and $-N$. Their sum gives a conflict weight associated with every variable in the scope of the higher-order GM bounding constraint, used to compute the $\frac{dom}{wdeg}$ heuristic.

It is important to note, as indicated in [23], that mono-objective preprocessing techniques that do not preserve all optimal solutions cannot be used safely in such a multi-objective optimization. This is the case for bounded variable elimination [26] and dominance rules [18] which need to be deactivated when the higher-order GM bounding constraint is processed.

5 Related works

While systematic multi-objective optimization has been well explored in (integer) Linear Programming [16], there is far less algorithmic work for GMs. The C-semiring framework with a partial order [7] can in principle represent such problems, but algorithms are restricted to min-max instead of min-sum problems in MRFs and CFNs and no implementation is available. More closely to what we propose, MO-MBE [35] is a multi-objective mini-bucket-elimination based higher-order constraint that computes a set of non-dominated solutions. However, the algorithm requires initial upper bounds on the optimum and has space complexity in $O(ed^{z-1}\prod_{j=1}^{p-1}(ub_j))$ and time complexity in $O(ed^{z}\prod_{j=1}^{p-1}(ub_j^2))$ with e cost functions, p objectives, maximum bucket size z and initial upper bounds ub_i. This limits the approach to problems with small known initial upper bounds ub_j. The algorithm is not incremental and is used as a constraint in a constraint programming context. In the same context, dedicated Pareto constraints have been developed using either a support-based algorithm or a multi-valued decision diagram to filter dominated solutions in a multi-objective branch and bound [21, 28]. Instead, our two-phase approach is based on single-objective B&B and the bounding constraint helps to reason about costs.

The introduction of a bounding constraint on a function defined by a Graphical Model, similar to our higher-order GM bounding constraint, has been explored for stochastic GMs [34] but the proposed bounds are not used for exact bi-objective search.

Compared to the original two-phase method [40, 41], instead of depth-first search, we used (hybrid) best-first branch and bound methods that offer anytime global lower bounds [1], allowing to produce an anytime Pareto front gap on difficult instances.[1] The main novelty of our approach is to provide an anytime Pareto front bounding within a generic two-phase method for (stochastic) graphical models for the first time. The higher-order bounding constraint for discrete GM optimization is another contribution.

6 Computational experiments

We implemented the two-phase method and the higher-order GM bounding constraint in C++ using the GM optimization solver `toulbar2`, winner of several medals in constraint programming (XCSP3 2022 and 2023) and probabilistic graphical model (UAI'2022) competitions. [2][3][4][5] We used the Hybrid-Best First

[1] In phase 2, we did not use a combination of objectives as in Test 3 with u3 bound [41] but used only one objective to optimize and prune search nodes (see Fig 1.d *obj* arrow). A comparison of these two bounding approaches remains to be done.

[2] https://forgemia.inra.fr/samuel.buchet/tb2_twophase in *release* branch.

[3] https://github.com/toulbar2/toulbar2 in *master* branch from version 1.2.1 including a dedicated linear constraint propagation method [30].

[4] https://xcsp.org/competitions

[5] https://uaicompetition.github.io/uci-2022

Search branch and bound method [1] with default parameters (function Solve in Algorithm 1). Additional preprocessing techniques (Virtual Arc Consistency [13] and Virtual Pairwise Consistency [29]) were also considered.

As a baseline competitive approach, we also implemented the two-phase method replacing `toulbar2` by the state-of-the-art commercial integer linear programming solver `cplex` (version 22.1.1.0). We used default parameters except for tolerance MIP gaps set to zero to ensure a complete search. The two objectives were linearized using the *local polytope* [13] formulation (*aka* the tuple encoding in [22], providing tighter bounds than a direct linear encoding):

$$\min \sum_{\substack{f_{\{x_i\}} \in \mathbf{F} \\ a \in \mathbf{D}_i}} f_{\{x_i\}}(a) x_{i:a} + \sum_{\substack{f_\mathbf{S} \in \mathbf{F}, |\mathbf{S}|>1 \\ \tau \in \ell(\mathbf{S})}} f_\mathbf{S}(\tau) y_{\mathbf{S}:\tau} \tag{1}$$

$$\text{s.t. } \forall x_i \in \mathbf{X}, \quad \sum_{a \in \mathbf{D}_i} x_{i:a} = 1 \tag{2}$$

$$\forall f_\mathbf{S} \in \mathbf{F}, |\mathbf{S}| > 1, x_i \in \mathbf{S}, a \in \mathbf{D}_i, \sum_{\substack{\tau \in \ell(\mathbf{S}) \\ \tau|\{x_i\}=a}} y_{\mathbf{S}:\tau} = x_{i:a} \tag{3}$$

where $x_{i:a}$ is a Boolean 0/1 variable taking value 1 if $x_i = a$, similarly, $y_{\mathbf{S}:\tau}$ is a non-negative continuous variable taking value 1 if tuple $\tau \in \ell(\mathbf{S})$ is chosen ($\ell(\mathbf{S})$ representing the Cartesian product of domains in $\mathbf{S}$). The objective function (1) minimizes the sum of the linear and nonlinear cost functions in one criterion ($\mathbf{F_1}$ or $\mathbf{F_2}$), while constraints (2) enforce that each variable must be assigned to exactly one value and constraints (3) enforce that the assignments of variables and tuples are compatible.[6] After linearization, expressing the bounding constraint on the second criterion becomes trivial. We just have to replace the objective function (1) with two linear constraints using the lower and upper bounds defined in TwoPhase (line 15).

Experiments were run on a single core of an Intel Xeon E5-2680 v3 2.5GHz processor with 128GB of RAM, using Linux Debian 6.1.52-1 operating system. The CPU time limit of Solve is $t = 30$ seconds per call (except for Protein where $t = 300$ and SetCover where $t = 3,600$) with a maximum number of Solve calls $m = 1,000$ (except for Knapsack where $m = 2,000$). The total CPU time limit is 1 hour.

6.1 Benchmarks

We experimented on six benchmarks. We took four existing benchmarks from the multi-objective literature in Operations Research. We added two benchmarks

[6] Further simplifications are made on the model, as done in [22]. First, original GM variables with a domain size of 2 are translated into a single Boolean 0/1 variable and no constraint (2). Secondly, a more-compact direct encoding of hard constraints is used. For every forbidden tuple τ in $f_\mathbf{S}$, we have $\sum_{x_i \in \mathbf{S}} (1 - x_{i:\tau|\{x_i\}}) \geq 1$.

coming from Graphical Models. The first five have at least one linear objective function ($\boldsymbol{F_1}$ or $\boldsymbol{F_2}$), allowing to replace the GM bounding constraint with linear constraints. The last benchmark has nonlinear objective functions and constraints. All benchmarks, sources, and results are made publicly available. [7]

Bi-objective vertex cover problem. In the weighted vertex cover problem (VertexCover), the aim is to select a subset of vertices with a minimum total weight in order to cover at least one extremity of every edge in a given graph. We followed the same protocol as in [36], generating random graphs with N vertices having two associated random costs $c^i_j \in [0, C], i \in \{1, 2\}$ for every vertex $j \in \{1, \ldots, N\}$, and E edges (randomly selected among $\frac{N(N-1)}{2}$ possible ones). In the GM, there is a 0/1 variable x_j for every vertex j and two cost functions $f^i_j, i \in \{1, 2\}$ with $f^i_j(0) = 0, f^i_j(1) = c^i_j$, representing the 2 objectives. For every edge (k, l), a hard constraint enforces that x_k or x_l is equal to one. We tested on 25 samples for every parameter combination of $N \in \{60, 70, 80, 90\}$, $E \in \{95, 250, 500, 950\}$, and $C = 4$, resulting in 400 bi-objective instances.

Bi-objective set cover problem. A related problem is the weighted set cover problem (SetCover). Here, the aim is to select a subset of elements with minimum total weight such that all the sets of a given list are covered by at least one of the selected elements. We took 120 randomly-generated instances from [24] composed of 5 instances for each combination of the following parameters: number of elements $N \in \{100, 150, 200\}$, number of sets $M \in \{20, 40, 60, 80\}$, fixed set cardinality $C \in \{5, 10\}$.[8][9] The integer cost values were chosen uniformly at random in the range $[1, 100]$. In the GM, there is a 0/1 variable x_j for every element j and two cost functions $f^i_j, i \in \{1, 2\}$ with $f^i_j(0) = 0, f^i_j(1) = c^i_j$, representing the 2 objectives. A hard constraint/clause is used to represent the fact that at least one element is selected in every set.

Bi-objective knapsack problem. The Knapsack problem is to select a subset of items such that the total weight of the selected items is less than a given capacity and the total profit associated to the items is maximized. Following [41], we generated 20 bi-objective instances with $N = 300$ items ; weights w_j and profits p^i_j (uncorrelated) being randomly chosen in $[1, 100]$, and the capacity $W = \frac{\sum_{j=1}^{N} w_j}{2}$. In the GM, there is a 0/1 variable x_j for every item and two cost functions $f^i_j, i \in \{1, 2\}$ with $f^i_j(0) = p^i_j, f^i_j(1) = 0$, representing the 2 objectives in minimization. A hard linear constraint is added to enforce the capacity constraint.

[7] https://forgemia.inra.fr/samuel.buchet/tb2_twophase (release branch).

[8] https://bitbucket.org/coreo-group/bioptsat

[9] The current version of `toulbar2` could not tackle large cardinality sets occurring in the fixed element probability set benchmark (SetCovering-EP) of [24].

Bi-objective warehouse location problem. The goal of the uncapacitated warehouse or facility location problem (Warehouse) is to open a subset of N warehouses to fulfill the demands of M stores. Each store must be served by one warehouse. Following [5], in every objective $i \in \{1, 2\}$, we add a random cost $o_j^i \in [C, 2C]$ for opening warehouse j. In the first objective, the cost c_{jk}^1 for serving store k by warehouse j is the Manhattan distance between k and l, stores and warehouses being randomly placed in a square of side C. In the 2nd objective, the corresponding cost c_{jk}^2 is randomly chosen in $[1, C]$. In the GM, there are N 0/1 variables for the warehouses and M variables with domain size N indicating which warehouse serves each store. Hard constraints ensure consistency between the two sets of variables [19]. Bi-objective linear cost functions associated with warehouses (resp. stores) are f_j^i with $f_j^i(0) = 0, f_j^i(1) = o_j^i$ (resp. f_k^i with $f_k^i(j) = c_{jk}^i$). We tested on 20 samples for $N = 6, M = 30$.

Bi-objective computational protein design problem. Our main motivation is to solve the bi-criteria problem (Protein) of designing a protein with minimum energy and a minimum number of different amino acid types, as done heuristically in [43]. We used a data set of 109 protein backbones from [43, 32, 38] for benchmarking. The first objective is defined by a Deep Learned decomposable protein design scoring function [15] and the second one is the number of different values used (a straightforward decomposition of NValue [6] as a GM using one extra Boolean variable B_a per value a, constrained to take value 1 — with associated cost 1 — if one variable uses a). This benchmark contains 109 instances with $n \in [42, 100]$ amino acid positions, $d = 20$ domain values, and $e \in [643, 2275]$ (non-linear) cost functions.

Bi-objective UAI'2022 benchmark. Last, we experimented with UAI'2022 Final Evaluation MMAP benchmark (UAI2022).[10] From the initial 100 instances, we kept those that were solved by `toulbar2` in less than 600 seconds. As in [34], the second objective of each instance is obtained by adding Gaussian noise $\mathcal{N}(\mu = 0, \sigma = 0.1)$ to the original functions (in the exponential/probability domain), truncating to non-negative numbers.[11] After the removal of instances with no solution, we obtained 26 instances with $n \in [225, 1997]$ variables, $d \in [2, 21]$ domain size, $e \in [578, 3334]$ cost functions per objective, and max. arity 5.

6.2 Experimental Results

We present a summary of our comparative results in Table 1.[12] The comparison measure is the number of instances completely solved per class of benchmark.

[10] https://www.ics.uci.edu/ dechter/uaicompetition/2022/FinalBenchmarks/MMAP.zip
[11] Zero probabilities lead to forbidden tuples after the required $-\log(\cdot)$ transform.
[12] Detailed results are available in Supplementary Materials. See https://forgemia.inra.fr/samuel.buchet/tb2_twophase/-/tree/release/results/supplementals.pdf.

Table 1. For each of the 6 benchmark classes, number of solved instances per method (in parenthesis, average CPU-time in seconds when all instances have been solved). Tested methods are `toulbar2`: two-phase method using `toulbar2` ; `tb2` no kp: two-phase method using `toulbar2` without transforming the GM bounding constraint into linear constraints ; `tb2` no pre: two-phase method using `toulbar2` without extra preprocessing techniques ; `cplex`: two-phase method using `cplex` ; RL: results from [36] ; J: from [24] ; C: results from [11]; BS: from [5]. N/A: method not applicable or result not available.

Benchmark	#	`toulbar2`	`tb2` no kp	`tb2` no pre	`cplex`	RL/J/C/BS
VertexCover	400	**400** (3.1s)	**400**(**0.5s**)	N/A	**400** (1.6s)	RL: **400**(10.2s)
SetCover	120	44	N/A	N/A	**120(47.3s)**	J: 40
Knapsack	20	0	N/A	N/A	**20** (**40.2s**)	C: **20** (41.7s)
Warehouse	20	**20** (**2.5s**)	**20** (33.6s)	N/A	**20** (4.8s)	BS: **20** (186.0s)
Protein	109	66	**69**	46	0	N/A
UAI2022	26	**26**(**23.8s**)	N/A	18	21	N/A

The solving task is to find a single representative state for each point of the exact Pareto front. To break ties when all the instances are solved, we use the average CPU-time.

We compare our two-phase method (`TwoPhase` using `toulbar2`) with the same method using `cplex` and with other approaches when directly available:

– for VertexCover, a multi-objective depth-first branch and bound (MO-BB) using multi-objective minibucket elimination (MO-MB$_{MOMBE}$) (experiments were made on a Pentium IV at 3GHz with 2GB) [36] ;
– for SetCover, a Max-SAT based hybrid method between core-guided and SAT-UNSAT search methods (MSHybrid) [24] (experiments made on Intel Xeon E5-2670 at 2.6GHz with 64GB and 1.5-hour CPU-time limit) ;
– for Knapsack, a two-phase method including dedicated instance preprocessing and where the second phase is a branch and bound with an adaptive branching heuristic (UCB) [11, 12] (experiments made on Intel Xeon E5620 at 2.40GHz with 6GB) ;
– for Warehouse, a multi-objective hybrid branch and bound combined with scalarization and using `Bensolve/GLPK` for solving the linear relaxations (M2.1.1.2) (experiments made on an Intel i7-8700 at 3.2GHz with 32GB) [5].

Notice that some reported experimental results (VertexCover, Warehouse) were performed on older machines, so the comparison should be taken with caution.

For two of the Operations Research benchmarks, `TwoPhase` using `toulbar2` has similar performance as `TwoPhase` using `cplex`, `toulbar2` being twice faster (resp. slower) on Warehouse (resp. VertexCover). However, it is clearly dominated by `cplex` on SetCover and Knapsack. Still, it remains faster than dedicated multi-objective branch and bound approaches, being 3.3 (resp. 74) times faster than MO-MB$_{MOMBE}$ (resp. M2.1.1.2) on VertexCover (resp. Warehouse). It also solves more SetCover instances than Max-SAT MSHybrid in less CPU-time (we set a 1-hour limit instead of 1.5 hours for MSHybrid to compensate for the difference in CPU frequencies). On Knapsack, TwoPhase using `cplex` is as efficient as a dedicated multi-objective branch and bound (UCB B&B [11]).

For Graphical Model benchmarks, TwoPhase using `toulbar2` got much superior performance than with `cplex`. It solves 63% of the Protein benchmark whereas none were solved by `cplex` in less than 300 seconds per internal Solve call. The largest solved instance by our approach contains 99 amino-acids. [13] The GM bounding constraint allows us to solve all the 26 UAI'2022 selected instances within the local 30-second CPU-time limit. In comparison, TwoPhase using `cplex` could solve only 21 instances. Using a longer 300-second time limit (detailed results in supplementary materials), TwoPhase using `cplex` was able to solve all the 26 instances requiring x7 more time (166.1 seconds) than with `toulbar2` (23.8 seconds) on average. We noticed that `cplex` can still be faster (up to 2.7 times) than `toulbar2` for 6 instances (Grids_26, or_chain_8/22/41/60, pedigree1) showing that both approaches can be worthwhile for this benchmark.

Impact of the GM bounding constraint. We tested the impact of removing the transformation of the GM bounding constraint into linear constraints when one of the original objectives is linear.[14] The fourth column in Table 1 shows the effect. Surprisingly, it improves the results on VertexCover (being x6.2 faster with x11.75 fewer search nodes in Phase 2) and Protein (solving optimally 3 more instances) whereas it significantly deteriorates on Warehouse. This behavior shows the complex interactions between the cost functions on the performance of the solver.

Impact of the preprocessing. We also tested the impact of preprocessing techniques. In the Protein benchmark, applying virtual arc consistency (VAC) [13] in preprocessing was shown to be effective [39]. It solves 66 instances instead of 46 without enforcing VAC. [15] Another stronger preprocessing already used in the UAI'2022 competition is to apply virtual pairwise consistency (VPWC) with additional zero-cost ternary cost functions [29] (options -A -pwc=-1 -t=1). Because this preprocessing can be quite time-consuming (2×0.48 seconds on average for UAI2022, up to 5.3 seconds on or_chain_41), we applied it on every objective only once before the two-phase method starts.[16] On UAI2022, using VPWC allows to solve 8 more instances than without it. We applied these preprocessing techniques on the GM benchmarks in the following part.

Two-phase method analysis and anytime Pareto fronts. In Table 2 we report a more detailed analysis of TwoPhase using `toulbar2`. We compared the

[13] 2pko_0009_multi having 15 states in the Pareto front found in 1,723 seconds with 37 Solve calls

[14] We could not test it for SetCover and Knapack due to a limitation in `toulbar2`.

[15] We also tested running VAC (option -A) in preprocessing at every Solve, instead of only once on the quadratic objective before running the two-phase method, but it solved optimally one less instance (65).

[16] CPU-times reported in Tables 1 and 2 do not include this preprocessing time.

Table 2. For each of the 6 benchmarks, number of tested instances, average CPU-time (seconds) and fraction spent in phase 1, average number of efficient states and fraction found in phase 1, average number of calls to **Solve** and fraction in phase 1, number of problems solved to optimality (full Pareto front) and after phase 1 (supported Pareto front), and final optimality gap.

Benchmark	#	Time (s)	p1%	Sols	p1%	Solve #	p1%	opt	opt1	gap%
VertexCover	400	3.1	5.6	8.0	57.2	16.0	59.8	400	400	0.000
SetCover	120	2411.1	5.6	28.9	38.5	46.3	46.3	44	119	1.462
Knapsack	20	2040.7	0.1	486.1	10.6	669.4	15.4	0	20	0.276
Warehouse	20	2.5	3.0	97.8	17.8	135.5	25.1	20	20	0.000
Protein	109	1334.5	82.5	12.7	84.6	32.1	67.9	66	73	14.766
UAI2022	26	23.8	6.8	14.0	45.5	24.9	47.8	26	26	0.000

first and second phases in terms of CPU-time spent, number of Pareto assignments found, and number of Solve calls. The ratio of supported efficient states was around 50% except for Warehouse (17.8%), Knapsack (10.6%), and Protein (85%). Knapsack had also the largest total number of efficient states (up to 851 for $n = 300$ items, average optimality gap of 0.276).

Protein is the only benchmark class where phase 1 takes more time and Solve than phase 2 (which finds two non-supported efficient states on average). The Pareto front of real protein dpbbss_multi (solved in 244.3 seconds by our approach) is given in Fig. 2 (Left). Two solutions of dpbbss_multi with $F_2 =$ 6 or 5 look very attractive compared to the $F_2 = 7$ identified in [43]. Fig. 2 also shows the pareto front boundings of an UAI Promedas instance solved to optimality (Center) and partially (Right), highlighting the anytime behaviour of the method.

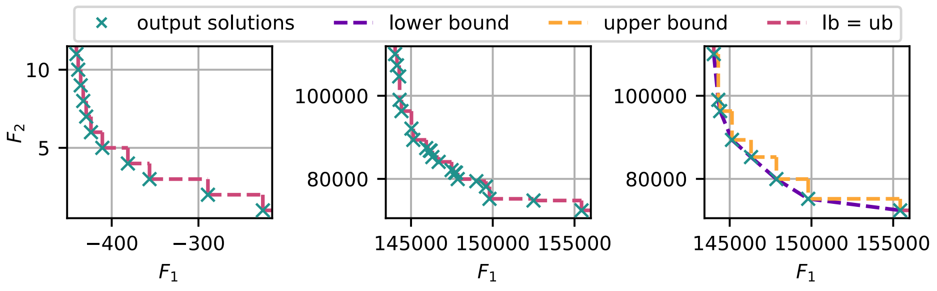

Fig. 2. Three examples of Pareto front bounding: (Left) Protein dpbbss_multi (optimal front), (Center) UAI2022 Promedas or_chain_60.fg.Q0.5.I3 (optimal), and (Right) UAI2022 Promedas or_chain_60.fg.Q0.5.I3 (optimality gap of 5.164%).

7 Conclusion

In this paper, motivated by the resolution of bi-objective protein design problems [43], we introduce an original anytime multi-objective two-phase method for solving discrete probabilistic and deterministic Graphical Models optimization problems. Our algorithm relies on a new higher-order GM bounding constraint exploiting Soft Arc Consistencies, a family of convergent message passing algorithms initially introduced for solving the Weighted Constraint Satisfaction Problem [37, 13]. Our experimental study, using a variety of real protein design problems as well as Operations Research and UAI'2022 evaluation benchmarks shows that our approach can outperform the state-of-the-art ILP solver CPLEX. At the crossroad of Probabilistic Reasoning and Operations Research, our algorithm offers a first effective answer to bi-objective discrete Graphical Model optimization, a family of problems that could possibly further benefit from additional multi-objective OR technology [20].

Acknowledgments. This study was funded by the French ANR projects ANR-19-CE09-0032 SpaceHex and ANR-19-PIA3-0004 ANITI. The authors have no competing interests to declare that are relevant to the content of this article.

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., André, I., Barbe, S., Davies, J., de Givry, S., Katsirelos, G., O'Sullivan, B., Prestwich, S., Schiex, T., Traoré, S.: Computational protein design as an optimization problem. Artificial Intelligence **212**, 59–79 (2014)
3. Allouche, D., Bessiere, C., Boizumault, P., De Givry, S., Gutierrez, P., Lee, J.H., Leung, K.L., Loudni, S., Métivier, J.P., Schiex, T., et al.: Tractability-preserving transformations of global cost functions. Artificial Intelligence **238**, 166–189 (2016)
4. Aneja, Y.P., Nair, K.P.K.: Bicriteria transportation problem. Management Science **25**(1), 73–78 (1979)
5. Bauß, J., Stiglmayr, M.: Augmenting bi-objective branch and bound by scalarization-based information. arXiv preprint arXiv:2301.11974 (2023)
6. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the nvalue constraint. Constraints **11**, 271–293 (2006)
7. Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. Constraints **4**, 199–240 (1999)
8. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: ECAI. vol. 16, p. 146 (2004)
9. Braden, B.: The surveyor's area formula. The College Mathematics Journal **17**(4), 326–337 (1986)
10. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio link frequency assignment. Constraints Journal **4**, 79–89 (1999)

11. Cerqueus, A.: Bi-objective branch-and-cut algorithms applied to the binary knapsack problem. Ph.D. thesis, université de Nantes (Nov 2015), https://hal.science/tel-01242210
12. Cerqueus, A., Gandibleux, X., Przybylski, A., Saubion, F.: On branching heuristics for the bi-objective 0/1 unidimensional knapsack problem. Journal of Heuristics **23**, 285–319 (2017)
13. 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)
14. Cooper, M.C., de Givry, S., Schiex, T.: Valued Constraint Satisfaction Problems, pp. 185–207. Springer International Publishing (2020)
15. Defresne, M., Barbe, S., Schiex, T.: Scalable coupling of deep learning with logical reasoning. In: Proc. of the 32^{nd} IJCAI. Macau, A.S.R., China (2023)
16. Ehrgott, M., Gandibleux, X., Przybylski, A.: Exact Methods for Multi-Objective Combinatorial Optimisation, pp. 817–850. Springer New York (2016)
17. Favier, A., de Givry, S., Legarra, A., Schiex, T.: Pairwise decomposition for combinatorial optimization in graphical models. In: Proc. of IJCAI-11. Barcelona, Spain (2011)
18. 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)
19. 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)
20. Halffmann, P., Schäfer, L.E., Dächert, K., Klamroth, K., Ruzika, S.: Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. Journal of Multi-Criteria Decision Analysis **29**(5-6), 341–363 (2022)
21. Hartert, R., Schaus, P.: A support-based algorithm for the bi-objective pareto constraint. In: Proceedings of the AAAI conference on artificial intelligence. vol. 28 (2014)
22. 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 (Summary). In: Proc. of CP-AI-OR'2016. p. 1 page. Banff, Canada (2016)
23. Jabs, C., Berg, J., Ihalainen, H., Järvisalo, M.: Preprocessing in sat-based multi-objective combinatorial optimization. In: Proc. of CP-23. Toronto, Canada (2023)
24. Jabs, C., Berg, J., Niskanen, A., Järvisalo, M.: Maxsat-based bi-objective boolean optimization. In: 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022) (2022)
25. Kratica, J., Tošic, D., Filipović, V., Ljubić, I.: Solving the simple plant location problem by genetic algorithm. RAIRO-Operations Research **35**(1), 127–142 (2001)
26. Larrosa, J.: Boosting search with variable elimination. In: Principles and Practice of Constraint Programming - CP 2000. LNCS, vol. 1894, pp. 291–305. Singapore (Sep 2000)
27. Luukkonen, S., van den Maagdenberg, H.W., Emmerich, M.T., van Westen, G.J.: Artificial intelligence in multi-objective drug design. Current Opinion in Structural Biology **79**, 102537 (2023)
28. Malalel, S., Malapert, A., Pelleau, M., Régin, J.C.: Mdd archive for boosting the pareto constraint. In: 29th International Conference on Principles and Practice of Constraint Programming (CP 2023). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2023)

29. Montalbano, P., Allouche, D., de Givry, S., Katsirelos, G., Werner, T.: Virtual pairwise consistency in cost function networks. In: Proc. of CP-AI-OR'2023. Nice, France (2023)
30. Montalbano, P., de Givry, S., Katsirelos, G.: Multiple-choice knapsack constraint in graphical models. In: Proc. of CP-AI-OR'2022. Los Angeles, CA (2022)
31. Nicolaou, C.A., Brown, N.: Multi-objective optimization methods in drug design. Drug Discovery Today: Technologies **10**(3), e427–e435 (2013)
32. Noguchi, H., Addy, C., Simoncini, D., Wouters, S., Mylemans, B., Van Meervelt, L., Schiex, T., Zhang, K.Y., Tame, J.R., Voet, A.R.: Computational design of symmetrical eight-bladed β-propeller proteins. IUCrJ **6**(1), 46–55 (2019)
33. o'Rourke, J.: Computational geometry in C. Cambridge university press (1998)
34. Rahman, T., Rouhani, S., Gogate, V.: Novel upper bounds for the constrained most probable explanation task. Advances in Neural Information Processing Systems **34**, 9613–9624 (2021)
35. Rollon, E., Larrosa, J.: Multi-objective propagation in constraint programming. In: Brewka, G., S.Coradeschi, Perini, A., Traverso, P. (eds.) ECAI 2006, 17th European Conference on Artificial Intelligence Proceedings. Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 128–132. IOS Press (2006)
36. Rollon, E., Larrosa, J.: Constraint optimization techniques for multiobjective branch and bound search. In: International conference on logic programming, ICLP (2008)
37. Schiex, T.: Arc consistency for soft constraints. In: Proc. of CP'00. pp. 411–424. Singapore (2000)
38. Simoncini, D., Allouche, D., De Givry, S., Delmas, C., Barbe, S., Schiex, T.: Guaranteed discrete energy optimization on large protein design problems. Journal of chemical theory and computation **11**(12), 5980–5989 (2015)
39. 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)
40. Ulungu, E.L., Teghem, J.: The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of computing and decision sciences **20**(2), 149–165 (1995)
41. Visée, M., Teghem, J., Pirlot, M., Ulungu, E.L.: Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. Journal of Global Optimization **12**(2), 139–155 (1998)
42. Vucinic, J., Simoncini, D., Ruffini, M., Barbe, S., Schiex, T.: Positive multistate protein design. Bioinformatics **36**(1), 122–130 (2020)
43. Yagi, S., Padhi, A.K., Vucinic, J., Barbe, S., Schiex, T., Nakagawa, R., Simoncini, D., Zhang, K.Y., Tagami, S.: Seven amino acid types suffice to create the core fold of RNA polymerase. Journal of the American Chemical Society **143**(39), 15998–16006 (2021)