Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization

Abdelkader Ouali[a,b], **David Allouche**[c], **Simon de Givry**[c], **Samir Loudni**[b], **Yahia Lebbah**[a], **Francisco Eckhardt**[c] and **Lakhdar Loukil**[a]
(a) University of Oran 1 Ahmed Ben Bella, Lab. LITIO, 31000 Oran, Algeria.
(b) University of Caen Normandy, CNRS, UMR 6072 GREYC, 14032 Caen, France.
(c) INRA, MIA Toulouse, UR-875, 31320 Castanet-Tolosan, France.

Abstract

Graphical models factorize a global probability distribution/energy function as the product/sum of local functions. A major inference task, known as MAP in Markov Random Fields and MPE in Bayesian Networks, is to find a global assignment of all the variables with maximum a posteriori probability/minimum energy. A usual distinction on MAP solving methods is complete/incomplete, i.e. the ability to prove optimality or not. Most complete methods rely on tree search, while incomplete methods rely on local search. Among them, we study Variable Neighborhood Search (VNS) for graphical models. In this paper, we propose an iterative approach above VNS which uses (partial) tree search inside its local neighborhood exploration. The resulting hybrid method offers a good compromise between completeness and anytime behavior than existing tree search methods while still being competitive for proving optimality. We further propose a parallel version of our method improving its anytime behavior on difficult instances coming from a large graphical model benchmark. Last we experiment on the challenging minimum energy problem found in Computational Protein Design, showing the practical benefit of our parallel version. Solver at `www.inra.fr/mia/T/toulbar2` v1.0.

1 INTRODUCTION

Probabilistic graphical models (Koller and Friedman, 2009) are formed by variables linked to each other by stochastic relationships. They enable to model complex systems with heterogeneous data and to capture uncertainty. Graphical models have been applied in a wide range of areas such as image analysis, speech recognition, bioinformatics, and ecology.

We focus on models with discrete variables like Markov Random Field and Bayesian Network. Our goal is to find a global assignment of all the variables with maximum a posteriori probability. This optimization task defines an NP-complete problem (Shimony, 1994). Solving methods can be categorized in two groups: exact and local search methods. Exact methods rely on tree search, variable elimination, linear programming, or a combination of them (Marinescu and Dechter, 2009; Otten and Dechter, 2012; Allouche *et al.*, 2015). Graph-cut and message-passing algorithms like loopy belief propagation and variational approaches (Kolmogorov, 2006; Wainwright and Jordan, 2008; Sontag *et al.*, 2008; 2012; Wang and Koller, 2013) are exact only in some particular cases (*e.g.,* Potts model or tree structure). Local search methods are stochastic algorithms like Gibbs sampling, Guided Local Search (Park, 2002; Hutter *et al.*, 2005), and Stochastic Greedy Search (Mengshoel *et al.*, 2011). Some of them have theoretical asymptotic proof of convergence, *i.e.,* the optimal solution is guaranteed to be found if infinite time is available. In practice, they may exhibit a better anytime behavior than exact methods on large and difficult problems (Marinescu *et al.*, 2003; Hutter *et al.*, 2005; Mengshoel *et al.*, 2011), *i.e.,* they produce better solutions in less time.

A few attempts have been done to combine exact and local search methods. A simple way is to run sequentially a local search algorithm then tree search, where solutions found by local search will be used as initial upper bounds for branch and bound exact methods. Another approach is to design a local search framework where the neighborhood exploration is performed by tree search in a systematic or non-systematic way. VNS/LDS+CP (Loudni and Boizumault, 2003) combines a metaheuristic, Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) with Limited Discrepancy Search (LDS) (Harvey and Ginsberg, 1995),

a partial tree search method (Section 2). We propose in Section 3 an iterative variant of VNS/LDS+CP called UDGVNS, adapted to graphical models and able to prove optimality when the neighborhood size is maximal and with unbounded tree search. In Section 4, we describe a coarse-grained parallel version called UPDGVNS with asynchronous cooperative execution of UDGVNS processes with centralized information exchange as in (Davidovic and Crainic, 2012; Ouali *et al.*, 2015). We report experimental results in Sections 5-6.

2 PRELIMINARIES

2.1 GRAPHICAL MODEL

Definition 1 *A* probabilistic graphical model *(or Gibbs distribution) (Koller and Friedman, 2009) is a triplet* $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ *with* $\mathcal{X} = \{X_1, \ldots, X_n\}$*, a set of* n *random variables,* $\mathcal{D} = \{D_1, \ldots, D_n\}$*, a set of finite domains of values of maximum size* $d = \max_{i=1}^{n} |D_i|$*, and* $\mathcal{F}$*, a set of potential functions. Each variable* X_i *takes values in* D_i*. An assignment of* $\mathcal{X}$ *is a set* $x = (x_1, \ldots, x_n)$*, with* $x_i \in D_i$*. The set of all possible assignments of* $\mathcal{X}$ *is denoted* $\Delta = \prod_{i=1}^{n} D_i$*. Let* $A = \{D'_1, \ldots, D'_n\}$ *with* $D'_i \subseteq D_i$ *represent a restricted set of* Δ *called a* partial assignment*. If* S *is a subset of* $V = \{1, \ldots, n\}$*,* X_S*,* x_S *and* Δ_S *are respectively the subset of random variables* $\{X_i, i \in S\}$*, the assignment* $(x_i, i \in S)$ *obtained from* x*, and the set of all possible assignments of* X_S*. Given a set* $\mathcal{S}$ *of part of* V*, the set* $\mathcal{F} = \{f_S\}_{S \in \mathcal{S}}$ *of maps from* Δ_S *to* $\mathbb{R}^+$*, called* potential functions*, is said to factorize a joint probability distribution* $\mathbb{P}$ *iff:*

$$\mathbb{P}(x) = \frac{1}{Z} \prod_{f_S \in \mathcal{F}} f_S(x_S) \tag{1}$$

where $Z = \sum_{x \in \Delta} \prod_{f_S \in \mathcal{F}} f_S(x_S)$ *is the normalizing constant, also called partition function.*

Among the various tasks, the *Most Probable Explanation* (MPE) problem is to find the most likely assignment $x \in \Delta$ to all the variables in $\mathcal{X}$ maximizing $\mathbb{P}(x)$. By taking the opposite of the logarithm of $\mathbb{P}(x)$, *i.e.,* $-\log \mathbb{P}(x) = \sum_{f_S \in \mathcal{F}} -\log f_S(x_S) - \log Z$, we obtain an additive model with $\varphi(x_S) = -\log f_S(x_S)$ called *energy functions*. Finding a solution of minimum energy is equivalent to MPE. In the rest of the paper, we consider energy minimization. When $\varphi(x_S)$ maps to $\mathbb{N}^+ \cup \{\infty\}$, the corresponding deterministic graphical model is called a *Cost Function Network* (CFN) (Meseguer *et al.*, 2006). Finding a solution of minimum cost is the same as doing energy minimization on the equivalent probabilistic model (Hurley *et al.*, 2016).

2.2 TREE DECOMPOSITION

Definition 2 *A tree decomposition of a connected graphical model is a pair* (C_T, T) *where* $T = (I, A)$ *is a tree with nodes set* I *and edges set* A *and* $C_T = \{C_i \mid i \in I\}$ *is a family of subsets of* $\mathcal{X}$*, called clusters, such that: (i)* $\cup_{i \in I} C_i = \mathcal{X}$*, (ii)* $\forall f_S \in \mathcal{F}$*,* $\exists\ C_i \in C_T$ *s.t.* $S \subseteq C_i$*, (iii)* $\forall i, j, k \in I$*, if* j *is on the path from* i *to* k *in* T*, then* $C_i \cap C_k \subseteq C_j$*.*

Definition 3 *The intersection of two clusters* C_i *and* C_j *is called a separator, and noted* $sep(C_i, C_j)$*.*

Definition 4 *A graph of clusters for a tree decomposition* (C_T, T) *is an undirected graph* $G = (C_T, E)$ *that has a vertex for each cluster* $C_i \in C_T$*, and there is an edge* $(C_i, C_j) \in E$ *when* $sep(C_i, C_j) \neq \emptyset$*.*

As finding an optimal tree decomposition is NP–hard, we use fast approximate algorithms like *min-fill* heuristic.

2.3 DFBB & LIMITED DISCREPANCY SEARCH

Depth-First Branch and Bound (DFBB) methods explore a search tree in a systematic way by recursively choosing the next unassigned variable to assign and by choosing a value in its domain for the assignment (the *branch* part) until a better solution is found or it can be proved that the subtree rooted at the current search node has no better solutions and it can be pruned (the *bound* part). DFBB depends on its variable and value ordering heuristics for branching in order to find good solutions rapidly and to reduce the size of the search tree to be explored.

Limited Discrepancy Search (LDS) (Harvey and Ginsberg, 1995) is a heuristic method that explores the search tree in a non-systematic way by making a limited number of *wrong* decisions w.r.t. its value ordering heuristic. We assume a binary search tree where at each search node either the selected variable is assigned to its chosen preferred value (left branch) or the value is removed from the domain (right branch). Each value removal corresponds to a wrong decision made by the search, it is called a *discrepancy*. The number of discrepancies is limited by a parameter denoted ℓ. See Algorithm 1 where $\texttt{lb}(A)$ gives a lower bound on the minimum energy $\min_{x \in \prod_{D_i \in A} D_i} -\log \mathbb{P}(x)$ of the partial assignment A.

In order to produce good quality solutions as time passes, a simple strategy is to iterate LDS with an increasing number of discrepancies ℓ going from ℓ_{min} to ℓ_{max}. Each iteration does at most ℓ discrepancies along the path from the root search node to a terminal node. Let h be the maximum height of the explored search tree, the number of terminals with ℓ discrepancies is bounded by $\binom{h}{\ell}$. Thus, LDS has a time complexity in $O(h^{\ell+1})$ (for $\ell < h/2$ (Larrosa *et al.*, 2016)) and a linear space complexity (thanks to the depth-first search principle as in DFBB). Notice that $h \leq n + \ell$ (ℓ right branches followed by n left branches). Without any discrepancy limit, we have at most $n(d-1)$ value removals (right branches)

Algorithm 1: Pseudo-code of mono LDS

```
Procedure
  LDS(ℓ, A, ub : In/Out, x : In/Out, opt : In/Out)
    if (∃D_i ∈ A, |D_i| > 1) then
      Choose an unassigned variable X_i ∈ 𝒳, |D_i| > 1 ;
      Choose a value x_i ∈ D_i ;
      A' ← (A \ {D_i}) ∪ {{x_i}} ;  // left branch
      if (lb(A') < ub) then LDS(ℓ, A', ub, x, opt) ;
1     if (ℓ > 0) then
        A'' ← (A \ {D_i}) ∪ {D_i \ {x_i}} ; // right b
        if (lb(A'') < ub) then
          LDS(ℓ − 1, A'', ub, x, opt)
      else
2       opt ← false ;
    else
3     ub ← lb(A), x ← A ;  // new sol. found
```

to reach a terminal node, where d is the maximum domain size of all variables. In order to be able to prove optimality, we set $\ell_{max} = n(d-1)$, *i.e.,* the last iteration is complete. In practice, optimality may be proved with a much smaller ℓ[1], and we just have to check if the discrepancy limit was reached during the search (falsified condition at line 1 of Alg. 1) to detect an incomplete search (flag *opt* set to false at line 2).

2.4 VARIABLE NEIGHBORHOOD SEARCH

Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) is a metaheuristic that uses a finite set of pre-selected neighborhood structures $N_k, k = 1, 2, ..., k_{max}$ to escape from local minima by systematically changing the neighborhood structure if the current one does not improve the current incumbent solution. VNS repeatedly performs three major steps. In the first one, called *shaking*, a solution x' is randomly generated in the neighborhoods of x denoted $N_k(x)$. In the second one, a local search method is applied from x' to obtain a local optimum x''. In the third one, called *neighborhood change*, if x'' is better than x, x is replaced with x'' and k is set to 1; otherwise, k is increased by one.

2.5 VNS METHODS IN GRAPHICAL MODELS

The use of VNS scheme for solving deterministic graphical models started with VNS/LDS+CP (Loudni and Boizumault, 2003). This approach is related to LNS (Shaw, 1998), but it adjusts dynamically the neighborhood size when search seems to stagnate as in VNS.

VNS/LDS+CP algorithm. Let N_k be the neighborhood structure of size k; N_k denotes the set of all subsets of k variables among $\mathcal{X}$. Let x be the current solution. First, x is partially destroyed by un-assigning a subset of k variables and an exploration of its (large) neighborhood is performed until the solution is repaired with a new one. The core of VNS/LDS+CP is its reconstruction phase. It runs LDS with a fixed discrepancy to explore the neighborhood of the solution. One advantage of this choice is its exploration speed that improves the quality profile and allows a more balanced exploration of the search tree.

Decomposition Guided VNS. More recently, Fontaine *et al.* (Fontaine *et al.*, 2013) investigated the incorporation of tree decomposition within VNS/LDS+CP in order to efficiently guide the exploration of large neighborhoods. They proposed DGVNS, a first local search approach that exploits the graph of clusters provided by a tree decomposition to build relevant neighborhood structures. For both DGVNS and VNS/LDS+CP, the reconstruction phase is performed using LDS with a fixed discrepancy ℓ, but neighborhood structures are managed in a different way. Instead of the neighborhood structures N_k used by VNS/LDS+CP, DGVNS uses neighborhood structures $N_{k,c}$, where k is the neighborhood size and C_c is the cluster where the variables will be selected from. If $(k > |C_c|)$, we complete the set of candidate variables to be unassigned by adding clusters adjacent to C_c. The neighborhood change in DGVNS is performed in the same way as in VNS. However, DGVNS considers successively all the clusters. This ensures a better diversification by covering a large number of different regions.

3 AN ITERATIVE DGVNS METHOD

We present UDGVNS (see Algorithm 2), unifying two complete and incomplete search methods. As done by iterative LDS, it restores the completeness of DGVNS by applying successive calls with an increasing discrepancy limit, controlling if tree search was partial using *opt* flag and the fact that the current neighborhood kept some variables assigned in A (test at line 9). In UDGVNS, optimality can be proven in two cases: (i) when the current neighborhood corresponds to the whole problem and the discrepancy value is greater than or equal to the maximum number of right branches or (ii) by examining the bounds at the root node ($ub = \mathtt{lb}(\mathcal{D})$ at lines 5 and 8). In this case, the search space is implicitly explored by the algorithm, therefore optimality is proven.

The initial solution of UDGVNS is obtained at line 4 by a modified version of LDS, denoted $\mathtt{LDS}^r$, that stops immediately after a first solution is found[2]. Thus $\mathtt{LDS}^r(\infty, \mathcal{D}, ub, x, opt)$ will return a solution with non-zero probability (*i.e.,* satisfying the constraints).

UDGVNS has two main parameters for tuning its compromise between optimality proof and anytime behavior: the discrepancy limit (ℓ) and the local search neighbor-

[1] It was less than or equal to 128 for all instances completely solved by LDS and VNS methods in Section 5.

[2] It stops the recursive LDS procedure at line 3 of Alg. 1 and set the optimality flag *opt* to false.

Algorithm 2: Unified DGVNS algorithm.

```
Function UDGVNS(ℓ_min, ℓ_max, +_ℓ, k_min, k_max, +_k, ub :
  In/Out, x : In/Out) : boolean
    let (C_T, T) be a tree decomposition of (X, D, F) ;
    opt ← true ;
4   LDS^r(∞, D, ub, x, opt) ;  // initial solution
5   if (ub = lb(D)) then opt ← true;
    c ← 1 ;              // current cluster index
    r ← 0 ;               // number of iterations
    ℓ ← ℓ_min ;    // initial discrepancy limit
    while (¬opt ∧ ℓ ≤ ℓ_max) do
        i ← 0 ;      // nb. of failed neighbor.
6       k ← k_min ;      // init. neighbo. size
        while (¬opt ∧ k ≤ k_max) do
            A ← getNeighborhood(x, C_c, k) ;
            ub' ← ub, opt ← true ;
7           LDS^r(ℓ, A, ub', x', opt) ; // nei. search
8           if (ub' = lb(D)) then opt ← true;
9           else if (A ≠ D) then opt ← false;
10          if (ub' < ub) then
                x ← x', ub ← ub' ; // new best sol
11              i ← 0, k ← k_min ;
12              r ← 0, ℓ ← ℓ_min ;
            else
                i ← i + 1 ;
                if (k < k_max) then
                    k ← min(k_max, k_min +_k i) ;
                else k ← ∞;
13          c ← 1 + c mod |C_T| ; // get next clus.
        r ← r + 1 ;
        if (ℓ < ℓ_max) then
            ℓ ← min(ℓ_max, ℓ_min +_ℓ r) ;
        else ℓ ← ∞;
14  return opt ;
```

hood size (k). Recall that inside DGVNS, the maximum tree height for $\texttt{LDS}^r$ is $h \leq k + \ell \leq k(d-1)$, so $\texttt{LDS}^r$ has time complexity in $O((k \min(\ell, d))^{\ell+1})$. The complexity increases exponentially with ℓ and polynomially with k. As soon as a better solution is found by the current neighborhood search (line 7), we stop the search in order to reinitialize the two parameters to their minimum value, as it is faster to explore small neighborhoods (lines 11-12). For each parameter, ℓ and k, we have tested three updating rules: increase by one at each iteration ($+_{\ell/k} = +$), multiply by two at each iteration ($a +_{\ell/k} b = \texttt{mult2}(a, b) = a \times 2^b$), and apply a Luby strategy (Luby *et al.*, 1993) ($a +_{\ell/k} b = \texttt{Luby}(a, b) = a \times luby(1 + b)$)[3].

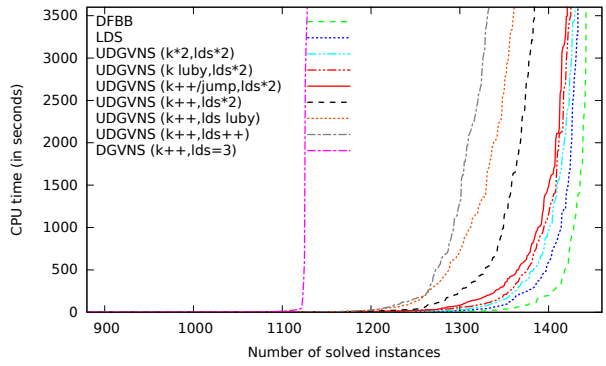
Operator $+_k$ controls the compromise between intensification and diversification. The goal of the Luby strategy is to exponentially increase the number of small neighborhoods explored compared to the number of larger ones. Whereas classical VNS algorithms will stuck on large problems, trying to diversify the search by exploring larger neighborhoods, VNS using Luby will spend more time on small neighborhoods in order to locally improve the current solution, favoring intensification. The `mult2` strategy reduces the number of neighborhood explorations at a given discrepancy limit, in order to try larger discrepancy limits more rapidly. If the problem is solvable by a complete search within the time limit, it will also speed-up the optimality proof.

Operator $+_\ell$ controls the compromise between incomplete and complete search. Using a fast growing strategy emphasis completeness whereas a slow growth should favor anytime behavior. We noticed that it is worthwhile to cover all the variables by the union of the explored neighborhoods in order to not miss some important variables. We tested a forth strategy for k which consists in a slow increment (by $+1$) at the beginning until $k = \max_{i \in I}(|C_i|) + |C_T| - 1$ then it *jumps* directly to $k = k_{max}$. This ensures that k grows slowly until the largest cluster has been totally explored by at least one neighborhood search. Then, when $k = k_{max} = |\mathcal{X}|$, `UDGVNS` does a restart, looking for an improved starting solution, using $\texttt{LDS}^r$ applied on the whole problem. If it fails to find a better solution, a larger discrepancy can be selected and `UDGVNS` continues its intensification process starting with a small neighborhood size (line 6).

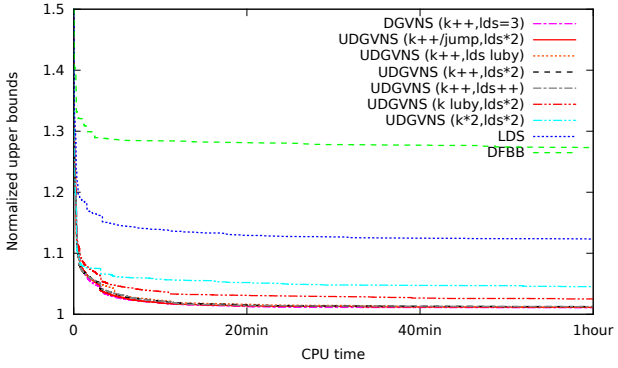
[3] Recall $luby(i) = \{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots\}$.

4 A PARALLEL VERSION OF UDGVNS

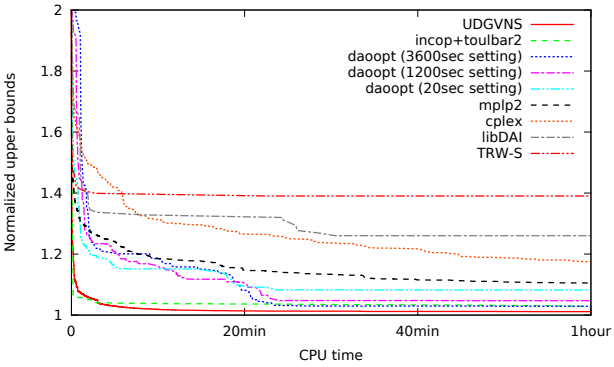
This section describes how `UDGVNS` can been parallelized. We denote the resulting algorithm `UPDGVNS` for Unified Parallel DGVNS. It relies on asynchronous cooperative execution of several `UDGVNS` (i.e. workers) processes with centralized information exchange. In the proposed parallelization one global best solution is shared among the worker processes. Initially, the master initiates the search by launching npr worker processes in parallel with the same initial solution (line 4 done in the master). Each worker process obtains from the master a copy of the current best solution x, the index c of the cluster to be processed and performs destroy and repair operations on its local copy. As soon as a new solution is found by a worker, it is sent to the master and a new cluster is requested (line 13 done in the master) to restart a new exploration starting from the best available overall solution. Like `UDGVNS`, each worker process controls locally how the discrepancy limit ℓ and the neighborhood size k evolve during successive explorations (except that ub comes from the master at line 10). The whole process stops as soon as a worker terminates (line 14) with an optimality proof or it reaches ℓ_{max} and k_{max}.



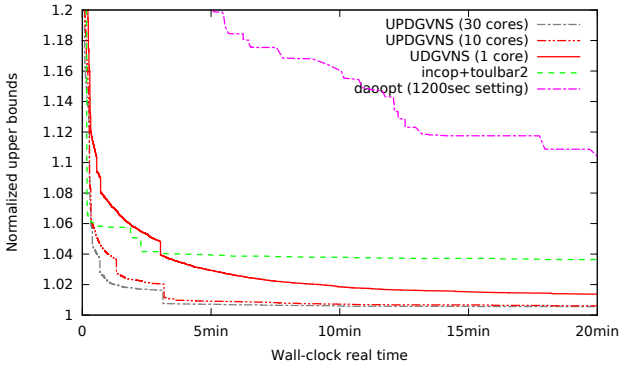
(a) Number of instances solved by each approach as time passes.



(b) Comparing the anytime behaviors of methods on 114 difficult instances.



(c) Comparing the anytime behavior of UDGVNS against state-of-the-art methods.



(d) Comparing the anytime behavior of parallel UPDGVNS.

Figure 1: Comparative evaluation on probabilistic and deterministic graphical models

5 UAI EVALUATION RESULTS

Benchmarks description. We performed experiments on probabilistic and deterministic graphical models coming from a large multi-paradigm benchmark repository (Hurley *et al.*, 2016)[4]. Among the 3016 available instances, we selected all the instances that were used in previous UAI competitions, in image analysis, or in Cost Function Network optimization. It includes 319 instances from *UAI 2008 Evaluation* and *Probabilistic Inference Challenge* (PIC 2011)[5], 1461 instances from the Computer Vision and Pattern Recognition (CVPR) OpenGM2 benchmark[6] (Kappes *et al.*, 2015), and 281 instances from the Cost Function Library[7]. In order to have a fair comparison between solvers, we preprocessed all the instances by polynomial-time problem reformulations and simplifications that remove variables (using bounded and functional variable elimination (Favier *et al.*, 2011)), values (using *dead-end elimination* (de Givry *et al.*, 2013)), and fixed-value potentials, after an initial lower bound computation by *Equivalence Preserving Transformations* (Cooper *et al.*, 2010) (enforcing *Virtual Arc Consistency* (VAC) as a message-passing algorithm). The resulting instances are smaller while preserving the same optimum. We used TOULBAR2 with options *-A -z=2* for this preprocessing step. We kept 1669 non-trivial instances (with more than one variable) for the experiments. The number of variables n ranges from 4 (CVPR-GeomSurf-3-gm13) to 48,566 (CVPR-ColorSeg-8-crops-small) with mean value $n \approx 403.4$ (instead of $1,316.5$ before preprocessing). For DGVNS methods, we built tree decompositions using *min-fill* heuristic. Because the number of clusters $m = |C_T|$ can be very large ($m \approx 256.7$), we merged any pair of connected clusters (C_i, C_j) when the separator size is too large compared to the individual cluster sizes ($|sep(C_i, C_j)| > 0.7 \min(|C_i|, |C_j|)$), resulting in $m \approx 19.9$ and mean tree width $\max_{i \in I}(|C_i|) \approx 76.9$ (instead of 49.6 without merging). In order to experiment sequential and parallel methods on the most difficult instances, we selected a subset of instances unsolved in 1 hour by all our DFBB, LDS, and VNS algorithms. We took at most twenty instances per problem category (avoiding over-representation issues by some categories), resulting

[4] genoweb.toulouse.inra.fr/~degivry/evalgm

[5] graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks, www.cs.huji.ac.il/project/PASCAL

[6] hci.iwr.uni-heidelberg.de/opengm2

[7] costfunction.org/benchmark

in a selection of 114 difficult instances[8].

Experimental protocol. LDS^r employs a randomized (for breaking ties) dynamic variable ordering heuristic[9]. Its value ordering heuristic chooses the EAC value as the preferred value and lower bounds are deduced by enforcing EDAC, as explained in (Larrosa *et al.*, 2005). In the following, we set $k_{min} = 4$, $k_{max} = n = |\mathcal{X}|$, $l_{min} = 1$, and $l_{max} = n(d-1)$. Here, DFBB corresponds to UDGVNS $(\infty, \infty, +, \infty, \infty, +)$, LDS to UDGVNS $(1, \infty, \texttt{mult2}, |\mathcal{X}|, |\mathcal{X}|, +)$, and DGVNS to UDGVNS $(3, 3, +, k_{min}, k_{max}, +)$.

We compared with state-of-the-art exact solvers. DAOOPT[10] won PIC 2011. It has a time-bounded initial phase of lower bound computation based on *Message Passing Linear Programming* algorithm (Sontag *et al.*, 2008; 2012; Ihler *et al.*, 2012) and mini-bucket elimination (Dechter and Rish, 2003) with iterative *min-fill* heuristic. It also finds good initial upper bounds using stochastic local search *GLS*$^+$ (Hutter *et al.*, 2005). We tested three parameter settings as suggested in (Otten *et al.*, 2012), controlling the time spent to compute initial lower and upper bounds. In the 3600sec setting, SLS is run 20 times with 10 seconds per run. The best solution found is used as an initial upper bound for an AND/OR exhaustive tree search. INCOP+TOULBAR2[11] (Hurley *et al.*, 2016) won the *UAI 2010 Evaluation* at 20 min. time limit. TOULBAR2 takes a starting solution from the best result of three runs of the *IDWalk* (Neveu *et al.*, 2004) local search algorithm (100,000 local moves per run). It is followed by an exhaustive hybrid best-first search (Allouche *et al.*, 2015). IBM ILOG CPLEX 12.7.0.0 (using parameters EPAGAP, EPGAP, and EPINT set to zero to avoid premature stop) was reported as being very competitive on some image analysis (Kappes *et al.*, 2015) and Markov Random Field problems (Hurley *et al.*, 2016). CPLEX explores its search tree using best-first search. It applies several heuristics methods to find good solutions before and during the search. We also compared with message-passing algorithms: LIBDAI[12] (Mooij, 2010), winner of the *UAI 2010 Evaluation* at 20sec. and 1hour time limits, MPLP2[13] (Sontag *et al.*, 2008; 2012), and TRW-S[14] (Kolmogorov, 2006). Note that LIBDAI and TRW-S are applied on the original instances rather than the preprocessed ones as we found they produced better results without applying VAC first. All solvers read problems in *uai* tabular format, except CPLEX which uses the local polytope formulation (called support encoding in (Hurley *et al.*, 2016)). All computations were performed on a cluster of 48-core AMD Opteron 6176 at 2.3 GHz and 384 GB of RAM with a 1-hour CPU time limit[15].

Optimality results. The efficiency of DFBB, LDS, and VNS methods to prove optimality is shown in the cactus plot of Figure 1a. DFBB was slightly more efficient than LDS and solved 1442 (resp. 1433) instances among 1669 in 1-hour time limit. They are followed by three UDGVNS strategies with $(k\ \texttt{mult2}, \ell\ \texttt{mult2})$ (1430 solved), $(k\ \texttt{Luby}, \ell\ \texttt{mult2})$ (1425 solved) and $(k\ \texttt{add1/jump}, \ell\ \texttt{mult2})$ (1421 solved), remaining very close in terms of performance. Next, a set of three less-and-less efficient UDGVNS strategies rise: $(k\ \texttt{add1}, \ell\ \texttt{mult2})$ (1384 solved), $(k\ \texttt{add1}, \ell\ \texttt{Luby})$ (1361 solved) and $(k\ \texttt{add1}, \ell\ \texttt{add1})$ (1333 solved), showing the importance of faster discrepancy increase to speed up optimality proofs. The worst strategy here was using a fixed discrepancy level ($\ell = 3$ as originally proposed in (Fontaine *et al.*, 2013)) which solved 1128 instances. For comparison, DAOOPT (3600sec setting) solved 1409 instances, CPLEX solved 1423, and INCOP+TOULBAR2 1440 instances (results in Supp. material).

Anytime upper bound profiles. In order to summarize the evolution of upper bounds as time passes, we took a subset of 114 difficult instances, unsolved in 1 hour by our DFBB, LDS, and VNS methods (whereas CPLEX could solve 17 instances). Specifically, for each instance I we normalize all energies as follows: the best, potentially suboptimal solution found by any algorithm is 1, the worst solution is 2. This normalization is invariant to translation and scaling. Figure 1b shows the upper bound behavior for different VNS strategies compared to DFBB and LDS (see Supp. material for a zoom). The ranking of best methods is the opposite of the cactus plot order, except for $(k\ \texttt{add1/jump}, \ell\ \texttt{mult2})$ which comes in second position. The $\ell = 3$ strategy got the best upper bounds in average, but still very close to the other VNS strategies, except may-be $(k\ \texttt{Luby}, \ell\ \texttt{mult2})$ and $(k\ \texttt{mult2}, \ell\ \texttt{mult2})$. We conclude that our new iterative UDGVNS method (especially $(k\ \texttt{add1/jump}, \ell\ \texttt{mult2})$) offers a good compromise between anytime behavior and optimality proof. These results also show that variable neighborhood search is by far superior to classical systematic (DFBB) or non-systematic (LDS) tree search methods, improving by more than 20% (resp. 10%) the quality of the solu-

[8] UAI DBN, Grid, Linkage, ObjectDetection, CVPR ChineseChars, ColorSeg-8, InPainting-4, InPainting-8, ProteinInteraction, and CFN CELAR, ProteinDesign, SPOT5, Warehouse categories.

[9] *Weighted degree heuristic* (Boussemart *et al.*, 2004).

[10] graphmod.ics.uci.edu/group/DAOOPT-UAI12

[11] www.inra.fr/mia/T/toulbar2 version 0.9.8 with parameters *-i -dee -hbfs*.

[12] bitbucket.org/jorism/libdai.git version 0.3.2 using UAI 2010 settings.

[13] cs.nyu.edu/~dsontag/code/README_v2.html using 2.10^{-7} int gap thres.

[14] github.com/opengm/opengm v2.3.5 with TRW-S v1.3 stopping after $100,000$ iterations or 10^{-5} gap thres.

[15] Using parameter *-pe parallel_smp* $\min(2x, 30)$ on a SUN Grid Engine for a method exploiting x core(s) to ensure half-load of the nodes on the cluster.

tions. In the following figures, we assume UDGVNS with (k add1/jump, ℓ mult2) strategy.

Figure 1c compare UDGVNS with state-of-the-art methods. Message-passing algorithms like TRW-S and LIBDAI gave the worst results. They could not find any solution for 20 (resp. 19) instances (mostly in UAI/Linkage and CFN/SPOT5 categories, both containing hard constraints). The same problem occurred for MPLP2 on 5 instances (SPOT5), but it obtained much better results on the remaining instances. CPLEX ran out of memory on two instances without producing any solution due to the heavy local polytope encoding (CFN/Warehouse/capb,capmq5). All other methods got better results in average and produced at least one solution per instance. According to its initial phase setting, DAOOPT provides different anytime behaviors, very close to the best solutions in 1 hour. UDGVNS performed the best, slightly better than INCOP+TOULBAR2, improving by 1.7% (resp. 2.3%) on average after 1 hour (20 min).

Parallelization. Finally, in order to evaluate the impact of core numbers, we consider the anytime upper bound behavior of the parallel release: UPDGVNS using (k add1/jump, ℓ mult2) with $\ell_{min} = 3$, taken from the best strategies enlightened by UDGVNS. We made a comparison with CPLEX using 10 and 30 cores (results in Supp. material), improving its anytime behavior, but still being far from the other competitors (30 cores gave solutions 10% higher than UPDGVNS after 1 hour). We could not compare with the parallel version of DAOOPT as it is based on a different cluster engine (*condor*) and it does not parallelize its initial phase.

Figure 1d shows that UPDGVNS (with 10 or 30 cores) provides slightly better upper bounds than UDGVNS (using 1 core) in less than 20 min. The results seems to be, in average, poorly sensitive to the cores number, due to the fact than the 10-core curve is extremely close from the 30-core one. The 10 and 30-core UPDGVNS curves converge quickly in less than 2min. INCOP+TOULBAR2 quickly drops out around 1 min and never reaches the same quality level. UDGVNS converges slower but still going down after 20 min. DAOOPT (1200sec setting) gave results 10% in average worse than UPDGVNS with either 10 and 30 processors.

6 COMPUTATIONAL PROTEIN DESIGN

Benchmarks description. One main challenge in Computational Protein Design (CPD) lies in the exploration of the amino-acid sequence space, while considering, to some extent, side chain flexibility. The exorbitant size of the search space urges for the development of efficient exact deterministic search methods enabling identification of low-energy sequence-conformation models, corresponding to the Global Minimum Energy Conformation (GMEC). The GMEC corresponds to a maximum probability mass due to the Boltzmann relation between molecular energy and probability, which is, for that matter, equivalent in the Markov Random Field modeling with Maximum A Posteriori probability (MAP-MRF) estimation.

CPD has been recently modeled as CFN, and various intensive benchmarking had demonstrated that in most cases CFN techniques are more efficient than other discrete optimization methods (Allouche *et al.*, 2014), especially for full design of instances lesser than 100 Amino acid (Simoncini *et al.*, 2015) . In this paper we would like to evaluate VNS methods capability to solve difficult instances. Accordingly, we tried to generate new larger ones supposed to be more difficult to solve than those generated in (Simoncini *et al.*, 2015). For this aims, in a first stage, we selected proteins structure in the PDB databases with size between 100 and 300 amino-acid. On the basis of a database query described in the supplementary material, we obtained a first set including 438 PDB references, which has been sorted according to the average volume per variable calculated using the radius of gyration of each instance (see Supp. material). At last, we extracted for benchmarking only the 20th first elements and used as benchmarking set. Each protein structure was fully redesigned according to (Simoncini *et al.*, 2015) protocol. On the basis of energy matrix generated with a modified release of PYROSETTA.4 script (Simoncini *et al.*, 2015) in order to use the last released Rosetta force-field (aka Beta November 2016) (Alford and others, 2017). The instances (see Supp. material) contain from 130 up to $n = 282$ variables with maximum domain size from 383 to 438, and between 1706 and 6208 cost/energy functions, and besides the tree width ranges from 21 to 68 and from 0.16 to 0.34 for a normalized tree width.

In this section, we evaluate the effectiveness of UDGVNS and UPDGVNS on CPD instances in terms of optimality proof (cf. **part (a)**) and solution quality vs. CPU time (cf. **part (b)**). All computations were performed on a cluster of 96-core AMD Opteron 6174 at 2.2 GHz and 256 GB of RAM. The parallelization has been done in MPI.

Experimental Protocol. We compared UDGVNS and UPDGVNS with VNS/LDS+CP and FIXBB (CPD dedicated algorithm provided by Rosetta package). We also compared with TOULBAR2. We tested TOULBAR2 using Virtual Arc Consistency (Cooper *et al.*, 2010) as preprocessing combined with hybrid best-first search with

tree decomposition (BTD−HBFS) using *min-fill* heuristic. Dead end elimination is turned off, according to (Simoncini *et al.*, 2015) (the resulting command line is: toulbar2-B=1-A-dee:-O=-3). The TOULBAR2 experiments use a 24-hour CPU time limit.

Concerning the VNS methods, in order to evaluate variability due to the random selection of neighborhoods, a set of 10 runs per instance with different seeds has been performed with a time limit of 1-hour per run. For the parallel strategy, the number of processes npr is set to 96 (i.e. maximum number of available processes). For UDGVNS and UPDGVNS, following the results observed in Section 5, we considered the following two settings for operators $+_k$ and $+_\ell$: $(k$ add1$, \ell = 3)$ which yields the best anytime performances and $(k$ add1 / jump$, \ell\, mult2)$ which offers the best compromise between both anytime performances and optimality proof. k_{min}, k_{max} and ℓ_{min} have been respectively set to 4, n (the total number of variables) and 3 (they correspond to the same parameter settings as those described in (Fontaine *et al.*, 2013)).

(a) Optimality results. Our first set of experiments aims at evaluating the efficiency of UDGVNS and UPDGVNS in terms of optimality proof comparing with TOULBAR2. As indicated before, we used the setting $(k$ add1 / jump$, \ell\, mult2)$. For all methods (see Supp. material), the CPU-time is the time to find and prove an optimal solution. TOULBAR2 clearly outperforms the two VNS methods: UDGVNS and UPDGVNS were able to prove optimality only on 3 instances among the 10 ones closed by TOULBAR2. However, the VNS methods are faster (except on 5dbl), despite the fact that BTD−HBFS benefits from the lower bounds reported by HBFS in individual clusters to improve the anytime behavior and from the global pruning lower bounds of BTD. This greatly improves the overall performance of TOULBAR2 compared to VNS methods.

(b) CPU time and solution quality results. Our second set of experiments aims at evaluating VNS capability with respect to finding the optimal solution or a solution of better quality on instances for which optimal solutions are unavailable. For this aims, we selected $(k$ add1$, \ell = 3)$ as setting for operators $+_k$ and $+_\ell$.

Table 1 shows a comparative evaluation of VNS methods with FIXBB and TOULBAR2. For each instance and each VNS method, we report the number of successful runs to reach the optimum (or the best found solution for unsolved instances) within a 3600-second time limit, the average CPU times (in seconds) over the 10 runs (for unsuccessful runs, the CPU time is set to the time limit) $\pm$ the standard deviation. The Energy gap ΔE between the best VNS solution and the two external references FIXBB and TOULBAR2 are also given. For TOULBAR2, reported CPU times correspond to times to find an optimal solution (for solved instances) or a best one (for unsolved ones) within the 24-hour time limit.

VNS methods vs. FIXBB**.** Rosetta Modeling suite is one of the most popular software package used in CPD field. It provides a Monte Carlo based Simulated Annealing algorithm called FIXBB. In this work, the best solutions exhibited by 1000 FIXBB cycles performed on each CPD instance have been used as base-line to compare solution quality of the solution provided by VNS methods when TOULBAR2 BTD-HBFS fails to solve the instance.

The FIXBB CPU times are two orders of magnitude higher than the 1h time limit imposed for VNS evaluation. They are not reported as they exceed 100-hours in sequential mode, even if FIXBB cycles are independent and thus be easy to parallelize. As we can see in Table 1 (see column ΔE of (4)), the solution quality of FIXBB is in all cases inferior to the best solution found by the VNS methods. The Energetic gap ΔE between FIXBB solution and the best VNS overall solution ranges between +0.16 and +5.20 Rosetta Energy Unit (R.E.U). As shown in (Simoncini *et al.*, 2015) such a level of energy difference can strongly impact the designed protein solution (i.e. the corresponding sequences of the two methods can be far in terms of hamming distance).

VNS methods vs. TOULBAR2**.** The comparison between best solutions found by VNS methods and TOULBAR2 shows that, excepted VNS/LDS+CP, both UDGVNS and UPDGVNS provide in all case the same or even better solution than TOULBAR2 (see column ΔE of (5) in Table 1). The Energetic gap ΔE in the worst case reaches 17.86 R.U.E. Concerning the number of successful runs reported over the 10 runs, VNS/LDS+CP seems less robust respectively than UDGVNS and the parallel release UPDGVNS. This last one provides in all cases the best solution over all. Table 1 also compares the VNS methods in terms of speedups. We observe that speedup values are fluctuating from one instance to another, very likely due to the tree decomposition resulting from the 3D shape of each instance. For VNS/LDS+CP and UDGVNS, it range between 0.62 and 2.86 over the 14 instances solved by both methods. As expected, when tree decomposition and parallelization are used, not only the speed of resolution increases but the reliability too (speedup values between 4.1 and 18.5). Moreover, the comparison between UDGVNS and UPDGVNS shows significant accelerations (between 2.06 and 9.22), thus confirming the practical interest of parallelization in addition to the exploitation of problem decomposition. We can observe (in Supp. material) that both UDGVNS and UPDGVNS find optimal solutions more quickly than TOULBAR2 within the one hour time limit (with the exception of UDGVNS

Instance	ncl	(1)		(2)		(3)		(4)	(5)		Speed-up		
		Succ.	Time (s)	Succ.	Time (s)	Succ.	Time (s)	ΔE	Time (s)	ΔE	(1/2)	(1/3)	(2/3)
5dbl	87	10/10	2,761± 67	10/10	963± 53	10/10	149± 18	0.27	783	0	2.86	18.53	6.46
5jdd	168	0/10	TO	10/10	3,248 ± 162	10/10	646± 72	4.08	20,662	0.04	-	-	5.02
3r8q	157	0/10	TO	10/10	3,478± 82	10/10	397± 47	4.19	12,762	0	-	-	8.76
4bxp	108	10/10	1,213± 32	10/10	1,352± 40	10/10	216± 24	0.26	2,966	0	0.89	5.61	6.26
1f00	177	0/10	TO	10/10	2,575± 29	10/10	542± 30	4.38	9,749	0	-		4.75
2x8x	131	10/10	3,312± 111	4/10	2,801± 1,071	10/10	521± 58	4.16	69,213	3.61	1.18	6.35	5.37
1xaw	66	0/10	TO	10/10	581± 38	10/10	260± 57	2.73	2,804	0	-	-	2.23
5e10	74	10/10	1,667± 86	10/10	1,412± 21	10/10	132± 24	0.26	1,171	0	1.18	12.62	10.7
1dvo	82	10/10	940± 22	10/10	940± 22	10/10	197± 25	2.90	34,142	0.18	1	4.77	4.77
1ytq	67	10/10	2,235± 211	10/10	1,304± 40	10/10	280± 26	1.67	17,063	0.31	1.71	7.98	4.65
2af5	140	0/10	TO	10/10	2,659± 75	10/10	894± 247	4.37	86,029	0.60	-	-	2.97
1ng2	86	10/10	1,065± 69	10/10	547± 18	10/10	260± 26	1.14	38,730	5.93	1.94	4.09	2.10
3sz7	79	0/10	TO	10/10	2,952± 276	10/10	320± 50	3.11	82,625	0.54	-	-	9.22
2gee	110	10/10	1,647± 8	10/10	1,276± 19	10/10	286± 38	1.68	5,021	0	1.29	5.75	4.46
5e0z	73	10/10	621± 15	10/10	995± 18	10/10	105± 8	0.16	999	0	0.62	5.91	9.47
1yz7	87	10/10	2,148± 9	10/10	945± 24	10/10	457± 85	2.91	83,816	3.20	2.27	4.70	2.06
3lf9	72	10/10	1,636± 31	10/10	894± 41.17	10/10	216± 21	2.41	2,667	0	1.82	7.57	4.13
3e3v	91	0/10	TO	10/10	2,327± 621	10/10	263± 38	2.57	81,574	0.15	-	-	8.84
1is1	107	10/10	3,178± 51	10/10	2,274± 421	10/10	329± 48	3.45	63,832	0.42	1.39	9.65	6.91
5eqz	89	10/10	1,849± 49	10/10	697± 6	10/10	225± 16	2.20	12,768	0	2.65	8.21	3.097
4uos	118	10/10	2,304± 988	10/10	2,109± 62	10/10	469± 79	5.20	58,589	17.86	1.09	4.91	4.5

TO: TimeOut **(1):** VNS/LDS+CP(k add1, $\ell = 3$) **(2):** UDGVNS (k add1, $\ell = 3$) **(3):** UPDGVNS (npr, k add1, $\ell = 3$) **(4):** FIXBB **(5):** TOULBAR2

Table 1: Comparative evaluation on CPD instances. In bold instances solved by TOULBAR2. ncl is the number of clusters. ΔE is in Rosetta Energy unit. It has been obtained by the difference of solutions costs divided by the cost shift used during modeling, in this case 10^8.

on 5dbl and 5e10). For UDGVNS, speedup values range from 0.81 to 18.3 with a mean of 4.23 over all the solved instances. For UPDGVNS, the ratio in terms of speedup is greatly amplified (between 5.25 and 56.74 with a mean of 18.48 over all the solved instances). For unsolved instances, we measured the CPU times spent by the three VNS methods (within the 3600-second time limit) to reach the best solution produced by TOULBAR2. Results (in Supplementary material) show a clear ordering in terms of CPU times across different solvers, from TOULBAR2, VNS/LDS+CP, UDGVNS, and UPDGVNS. The speedup values are significantly improved, in particular with UPDGVNS (between 32.3 and 354.6 with a mean of 168.7 over all the unsolved instances). These results confirm the superiority of VNS methods in terms of anytime performance as compared to TOULBAR2.

CONCLUSION

In this paper we proposed a unified view of VNS methods including various LDS and neighborhood evolution strategies. Experiments performed on difficult instances, coming from a large graphical model benchmark, showed that our hybrid method has a much better anytime behavior than existing tree search methods and still being competitive for proving optimality. We further proposed a parallel version of our method improving its anytime behavior. It remains as future work to automatically choose the best parameter settings per instance.

Acknowledgements This work has been partially funded by the french "Agence nationale de la Recherche", reference ANR-16-C40-0028. We are grateful to the genotoul bioinformatics platform Toulouse Midi-Pyrenees (Bioinfo Genotoul) and the HPC HAYTHAM of university of Oran 1 for providing computing and storage resources. We thank Mathieu Fontaine for his contribution to the code of DGVNS.

References

[Alford and others, 2017] R Alford et al. The Rosetta all-atom energy function for macromolecular modeling and design. *bioRxiv*, 2017.

[Allouche *et al.*, 2014] D Allouche, J Davies, S de Givry, G Katsirelos, T Schiex, S Traoré, I André, S Barbe, S Prestwich, and B O'Sullivan. Computational Protein Design as an Optimization Problem. *Artif. Intell.*, 212:59–79, 2014.

[Allouche *et al.*, 2015] D Allouche, S de Givry, G Katsirelos, T Schiex, and M Zytnicki. Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In *Proc. of CP*, pages 12–28, 2015.

[Boussemart *et al.*, 2004] F Boussemart, F Hemery, C Lecoutre, and L Sais. Boosting systematic search by weighting constraints. In *Proc. of ECAI*, 2004.

[Cooper *et al.*, 2010] M Cooper, S de Givry, M Sanchez, T Schiex, M Zytnicki, and T Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174:449–478, 2010.

[Davidovic and Crainic, 2012] T Davidovic and T Crainic. MPI parallelization of variable neighborhood search. *Electronic Notes in Discrete Mathematics*, 39:241–248, 2012.

[de Givry *et al.*, 2013] S de Givry, S Prestwich, and B O'Sullivan. Dead-end elimination for weighted CSP. In *Proc. of CP*, pages 263–272, 2013.

[Dechter and Rish, 2003] R Dechter and I Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.

[Favier *et al.*, 2011] A Favier, S Givry, A Legarra, and T Schiex. Pairwise decomposition for combinatorial optim. in graphical models. In *Proc. of IJCAI*, pages 2126–2132, 2011.

[Fontaine *et al.*, 2013] M Fontaine, S Loudni, and P Boizumault. Exploiting tree decomposition for guiding neighborhoods exploration for VNS. *RAIRO OR*, 47(2):91–123, 2013.

[Harvey and Ginsberg, 1995] W Harvey and M Ginsberg. Limited discrepancy search. In *Proc. of IJCAI*, pages 607–615, 1995.

[Hurley *et al.*, 2016] B Hurley, B O'Sullivan, D Allouche, G Katsirelos, T Schiex, M Zytnicki, and S de Givry. Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, 21(3):413–434, 2016.

[Hutter *et al.*, 2005] F Hutter, H Hoos, and T Stützle. Efficient stochastic local search for MPE solving. In *Proc. of IJCAI*, pages 169–174, 2005.

[Ihler *et al.*, 2012] A Ihler, N Flerova, R Dechter, and L Otten. Join-graph based cost-shifting schemes. In *Proc. of UAI*, pages 397–406, 2012.

[Kappes *et al.*, 2015] J Kappes, B Andres, F Hamprecht, C Schnörr, S Nowozin, D Batra, S Kim, B Kausler, T Kröger, J Lellmann, N Komodakis, B Savchynskyy, and C Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *Int. J. of Computer Vision*, 115(2):155–184, 2015.

[Koller and Friedman, 2009] D Koller and N Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[Kolmogorov, 2006] V Kolmogorov. Convergent Tree-Reweighted Message Passing for Energy Minimization. *IEEE T. Patt. Anal. Mach. Intell.*, 28(10):1568–1583, 2006.

[Larrosa *et al.*, 2005] J Larrosa, S de Givry, F Heras, and M Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of IJCAI*, pages 84–89, 2005.

[Larrosa *et al.*, 2016] J Larrosa, E Rollon, and R Dechter. Limited discrepancy AND/OR search and its application to optimization tasks in graphical models. In *Proc. of IJCAI*, pages 617–623, 2016.

[Loudni and Boizumault, 2003] S Loudni and P Boizumault. Solving constraint optimization problems in anytime contexts. In *Proc. of IJCAI*, pages 251–256, 2003.

[Luby *et al.*, 1993] M Luby, A Sinclair, and D Zuckerman. Optimal speedup of Las Vegas algorithms. In *Proc. of TCS*, pages 128–133, 1993.

[Marinescu and Dechter, 2009] R. Marinescu and R. Dechter. Memory intensive and/or search for combinatorial optimization in graphical models. *AI*, 173(16-17):1492–1524, 2009.

[Marinescu *et al.*, 2003] R Marinescu, K Kask, and R Dechter. Systematic vs. non-systematic algorithms for solving the MPE task. In *Proc. of UAI*, pages 394–402, 2003.

[Mengshoel *et al.*, 2011] O Mengshoel, D Wilkins, and D Roth. Initialization and Restart in Stochastic Local Search: Computing a Most Probable Explanation in Bayesian Networks. *IEEE Trans. Knowl. Data Eng.*, 23(2):235–247, 2011.

[Meseguer *et al.*, 2006] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints processing. In *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.

[Mladenović and Hansen, 1997] N Mladenović and P Hansen. Variable Neighborhood Search. *Comput. Oper. Res.*, 24(11):1097–1100, November 1997.

[Mooij, 2010] Joris M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11:2169–2173, 2010.

[Neveu *et al.*, 2004] B Neveu, G Trombettoni, and F Glover. ID Walk: A Candidate List Strategy with a Simple Diversification Device. In *Proc. of CP*, pages 423–437, 2004.

[Otten and Dechter, 2012] L Otten and R Dechter. Anytime AND/OR depth-first search for combinatorial optimization. *AI Communications*, 25(3):211–227, 2012.

[Otten and Dechter, 2017] L Otten and R Dechter. And/or branch-and-bound on a computational grid. page 84p., 2017. Unpublished.

[Otten *et al.*, 2012] L Otten, A Ihler, K Kask, and R Dechter. Winning the PASCAL 2011 MAP challenge with enhanced AND/OR branch-and-bound. In *NIPS W. DISCML*, 2012.

[Ouali *et al.*, 2015] A Ouali, S Loudni, L Loukil, P Boizumault, and Y Lebbah. Replicated parallel strategies for decomposition guided VNS. *ENDM*, 47:93–100, 2015.

[Park, 2002] J Park. Using weighted MAX-SAT engines to solve MPE. In *Proc. of AAAI*, pages 682–687, 2002.

[Shaw, 1998] P Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. of CP*, pages 417–431, 1998.

[Shimony, 1994] S Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68:399–410, 1994.

[Simoncini *et al.*, 2015] D Simoncini, D Allouche, S de Givry, C Delmas, S Barbe, and T Schiex. Guaranteed discrete energy optimization on large protein design problems. *J. of Chemical Theo. and Comput.*, 11(12):5980–5989, 2015.

[Sontag *et al.*, 2008] D Sontag, T Meltzer, A Globerson, Y Weiss, and T Jaakkola. Tightening LP relaxations for MAP using message-passing. In *Proc. of UAI*, pages 503–510, 2008.

[Sontag *et al.*, 2012] D Sontag, D Choe, and Y Li. Efficiently searching for frustrated cycles in MAP inference. In *Proc. of UAI*, pages 795–804, 2012.

[Wainwright and Jordan, 2008] M Wainwright and M Jordan. Graphical models, exponential families, and variational inference. *F&T. in Machine Learning*, 1(1–2):1–305, 2008.

[Wang and Koller, 2013] H Wang and D Koller. Subproblem-tree calibration: A unified approach to max-product message passing. In *Proc. of ICML*, pages 190–198, 2013.