Computational Protein Design as an Optimization Problem

David Allouche, Jessica Davies, Simon de Givry, George Katsirelos, Thomas Schiex*

UBIA, UR-875, INRA, F-31320 Castanet Tolosan, France

Seydou Traoré, Isabelle André, Sophie Barbe

LISBP, INSA, UMR INRA 792/CNRS 5504, F-31400 Toulouse, France

Steve Prestwich, Barry O'Sullivan

Insight Centre for Data Analytics, University College Cork, Ireland

Abstract

Proteins are chains of simple molecules called amino acids. The three-dimensional shape of a protein and its amino acid composition define its biological function. Over millions of years, living organisms have evolved a large catalog of proteins. By exploring the space of possible amino acid sequences, protein engineering aims at similarly designing tailored proteins with specific desirable properties. In Computational Protein Design (CPD), the challenge of identifying a protein that performs a given task is defined as the combinatorial optimization of a complex energy function over amino acid sequences.

In this paper, we introduce the CPD problem and some of the main approaches that have been used by structural biologists to solve it, with an emphasis on the exact method embodied in the dead-end elimination/A* algorithm (DEE/A*). The CPD problem is a specific form of binary Cost Function Network (CFN, aka Weighted CSP). We show how DEE algorithms can be incorporated and suitably modified to be maintained during search, at reasonable computational cost.

We then evaluate the efficiency of CFN algorithms as implemented in our solver `toulbar2`, on a set of real CPD instances built in collaboration with structural biologists. The CPD problem can be easily reduced to 0/1 Linear Programming, 0/1 Quadratic Programming, 0/1 Quadratic Optimization, Weighted Partial MaxSAT and Graphical Model optimization problems. We compare `toulbar2` with these different approaches using a variety of solvers. We observe tremendous differences in the difficulty that each approach has on these instances.

Overall, the CFN approach shows the best efficiency on these problems, improving by several orders of magnitude against the exact DEE/A* approach. The introduction of dead-end elimination before or during search allows to further improve these results.

Keywords: weighted constraint satisfaction problem, soft constraints, neighborhood substitutability, constraint optimization, graphical model, cost function networks, integer linear programming, quadratic programming, computational protein design, bioinformatics, maximum a posteriori inference, maximum satisfiability

1. Introduction

A protein is a sequence of basic building blocks called *amino acids*. Proteins are involved in nearly all structural, catalytic, sensory, and regulatory functions of living systems [26]. Performing these functions generally requires that proteins are assembled into well-defined three-dimensional structures specified by their amino acid sequence. Over millions of years, natural evolutionary processes have shaped and created proteins with novel structures and functions by means of sequence variations, including mutations, recombinations and duplications. Protein engineering techniques coupled with high-throughput automated procedures make it possible to mimic the evolutionary process on a greatly accelerated time-scale, and thus increase the odds to identify the proteins of interest for technological uses [71]. This holds great interest for medicine, synthetic biology, nanotechnologies and biotechnologies [67, 75, 39]. In particular, protein engineering has become a key technology to generate tailored enzymes able to perform novel specific transformations under specific conditions. Such biochemical transformations enable to access a large repertoire of small molecules for various applications such as biofuels, chemical feedstocks and therapeutics [45, 11]. The development of enzymes with required substrate selectivity, specificity and stability can also be profitable to overcome some of the difficulties encountered in synthetic chemistry. In this field, the *in vitro* use of artificial enzymes in combination with organic chemistry has led to innovative and efficient routes for the production of high value molecules while meeting the increasing demand for ecofriendly processes [61, 13]. Nowadays, protein engineering is also being explored to create non-natural enzymes that can be combined in vivo with existing biosynthetic pathways, or be used to create entirely new synthetic metabolic pathways not found in nature to access novel biochemical products [28]. These latest approaches are central to the development of synthetic biology. One significant example in this field is the full-scale production of the antimalarial drug (artemisinin) from the engineered bacteria *Escherichia coli* [66].

With a choice among 20 naturally occurring amino acids at every position, the size of the combinatorial sequence space is out of reach for current experimental methods, even for short sequences. Computational protein design (CPD) methods therefore try to intelligently guide the protein design process by producing a *collection* of proteins, that is rich in functional proteins, but small enough to be experimentally evaluated. The challenge of choosing a sequence of amino acids to perform a given task is formulated as an optimization problem, solvable computationally. It is often described as the inverse problem of protein folding [70]: the three-dimensional structure is known and we have to find amino acid sequences that fold into it. It can also be considered as a highly combinatorial variant of side-chain positioning [82] because of possible amino acid mutations.

Various computational methods have been proposed over the years to solve this problem and several success stories have demonstrated the outstanding potential of CPD methods to engineer proteins with improved or novel properties. CPD has been successfully applied to increase protein thermostability and solubility; to alter specificity towards some other molecules; and to design various binding sites and construct *de novo* enzymes (see for example [46]).

*Corresponding author

Despite these significant advances, CPD methods must still mature in order to better guide and accelerate the construction of tailored proteins. In particular, more efficient computational optimization techniques are needed to explore the vast combinatorial space, and to facilitate the incorporation of more realistic, flexible protein models. These methods need to be capable of not only identifying the optimal model, but also of enumerating solutions close to the optimum.

We begin by defining the CPD problem with rigid backbone, and then introduce the approach commonly used in structural biology to exactly solve CPD. This approach relies on dead-end elimination (DEE), a specific form of dominance analysis that was introduced in [24], and later strengthened in [37]. If this polynomial-time analysis does not solve the problem, an A^* algorithm is used to identify an optimal protein design.

We observe that the rigid backbone CPD problem can be naturally expressed as a Cost Function Network (aka Weighted Constraint Satisfaction Problem). In this context, DEE is similar to neighbourhood substitutability [27]. We show how DEE can be suitably modified so as to be maintained during search at reasonable computational cost, in collaboration with the usual soft local consistencies.

To evaluate the efficiency of the CFN approach, we model the CPD problem using several combinatorial optimization formalisms. We compare the performance of the 0/1 linear programming and 0/1 quadratic programming solver `cplex`, the semidefinite programming based Boolean quadratic optimization tool `biqmac`, several weighted partial MaxSAT solvers, the Markov random field optimization solvers `daoopt` and `mplp` [80], and the CFN solver `toulbar2`, against that of a well-established CPD approach implementing DEE/A^*, on various realistic protein design problems. We observe drastic differences in the difficulty that these instances represent for different solvers, despite often closely related models and solving techniques.

2. The Computational Protein Design approach

A protein is a sequence of organic compounds called amino acids. All amino acids consist of a common *peptidic core* and a *side chain* with varying chemical properties (see Figure 1). In a protein, amino acid cores are linked together in sequence to form the *backbone* of the protein. A given protein *folds* into a 3D shape that is determined from the sequence of amino acids. Depending upon the amino acid considered, the side chain of each individual amino acid can be rotated along up to 4 dihedral angles relative to the backbone. After Anfinsen's work [3], the 3D structure of a protein can be considered to be defined by the backbone and the set of side-chain rotations. This is called the *conformation* of the protein and it determines its chemical reactivity and biological function.

Computational Protein Design is faced with several challenges. The first lies in the exponential size of the conformational and protein sequence space that has to be explored, which rapidly grows out of reach of computational approaches. Another obstacle to overcome is the accurate structure prediction for a given sequence [47, 38]. Therefore, the design problem is usually approached as an inverse folding problem [70], in order to reduce the problem to the identification of an amino acid sequence that can fold into a target 3D-scaffold that matches the design objective [9]. In structural biology, the stability of aconformation can be directly evaluated through the energy of the conformation, a stable fold being of minimum energy [3].

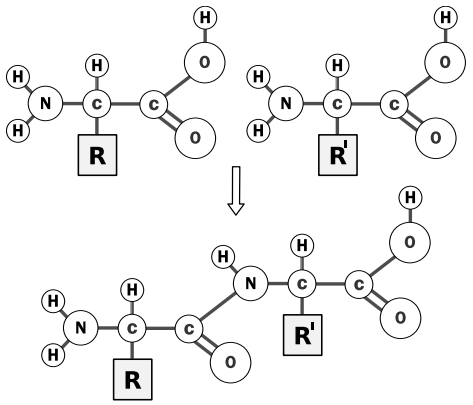

Figure 1: A representation of how amino acids, carrying specific side chains R and R', can link together through their core to form a chain (modified from wikipedia). One molecule of water is also generated in the process.

In CPD, two approximations are common. First, it is assumed that the resulting designed protein retains the overall folding of the chosen scaffold: the protein *backbone* is considered fixed. At specific positions chosen by the computational biologist (or automatic selection), the amino acid can be modified by changing the *side chain* as shown in Fig. 2. Second, the domain of conformations available to each amino acid side chain is actually continuous. This continuous domain is approximated using a set of discrete conformations defined by the value of their inner dihedral angles. These conformations, or *rotamers* [44], are derived from the most frequent conformations in the experimental repository of known protein structures, PDB (Protein Data Bank, `www.wwpdb.org`). Different discretizations have been used in constraint-based approaches to protein structure prediction [10].

The CPD is then formulated as the problem of identifying a conformation of minimum energy via the mutation of a specific subset of amino acid residues, i.e. by affecting their identity and their 3D orientations (rotamers). The conformation that minimizes the energy is called the *GMEC* (Global Minimum Energy Conformation).

In order to solve this problem, we need a computationally tractable energetic model to evaluate the energy of any combination of rotamers. We also require computational optimization techniques that can efficiently explore the sequence-conformation space to find the sequence-conformation model of global minimum energy.

Energy functions. Various energy functions have been defined to make the energy computation manageable [7]. These energy functions include non-bonded terms such as van der Waals and electrostatics terms, often in conjunction with empirical contributions describing hydrogen bonds. The surrounding solvent effect is generally treated implicitly as a continuum. Statistical terms may be added in order to approximate the effect of mutations on the unfolded state or the contribution of conformational entropy. Finally, collisions between atoms (steric clashes) are also taken into account. In this work, we used the state-of-the-art energy functions implemented in the CPD dedicated tool `osprey` 2.0 [30].

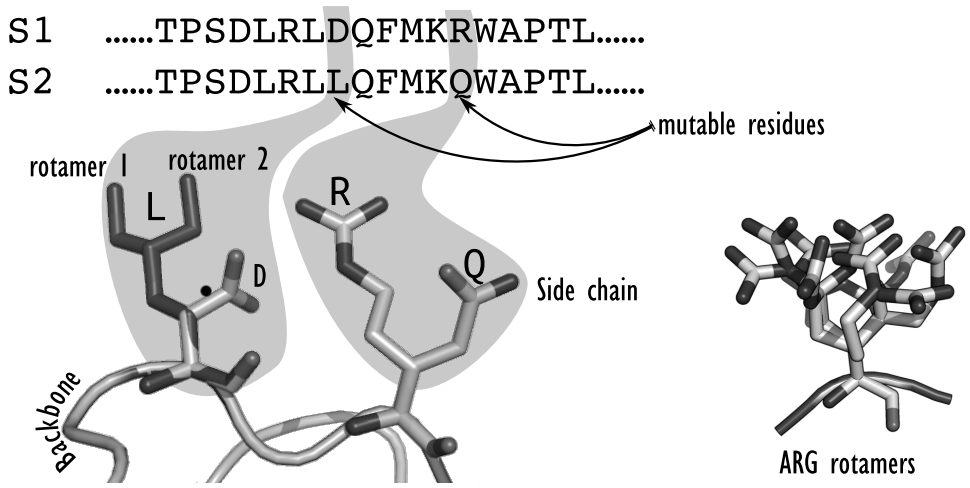

Figure 2: A local view of combinatorial sequence exploration considering a common backbone. Changes can be caused by amino acid identity substitutions (*for example D/L* or *R/Q*) or by amino acid side-chain reorientations (rotamers) for a given amino acid. A typical rotamer library for one amino acid is shown on the right (ARG=Arginine).

These energy functions can be reformulated in such a way that the terms are locally decomposable. Then, the energy of a given protein conformation, defined by a choice of one specific amino acid with an associated conformation (rotamer) for each residue, can be written as:

$$E = E_{\varnothing} + \sum_{i} E(i_r) + \sum_{i} \sum_{j>i} E(i_r, j_s) \tag{1}$$

where E is the potential energy of the protein, $E_{\varnothing}$ is a constant energy contribution capturing interactions between fixed parts of the model, $E(i_r)$ is the energy contribution of rotamer r at position i capturing internal interactions (and a reference energy for the associated amino acid) or interactions with fixed regions, and $E(i_r, j_s)$ is the pairwise interaction energy between rotamer r at position i and rotamer s at position j [24]. This decomposition brings two properties:

- Each term in the energy can be computed for each amino acid/rotamer (or pair for $E(i_r, j_s)$) independently.
- These energy terms, in $kcal/mol$, can be precomputed and cached, allowing to quickly compute the energy of a design once a specific rotamer (an amino acid-conformation pairing) has been chosen at each non-rigid position.

The rigid backbone dicrete rotamer Computational Protein Design problem is therefore defined by a fixed backbone with a corresponding set of positions (residues), a rotamer library and a set of energy functions. Each position i of the backbone is associated with a subset D_i of all (amino-acid,rotamer) pairs in the library. The problem is to identify at each position i a pair from D_i such that the overall energy E is minimized. In

practice, based on expert knowledge or on pecific design protocols, each position can be fixed (D_i is a singleton), flexible (all pairs in D_i have the same amino-acid) or mutable (the general situation).

2.1. Exact CPD methods

The protein design problem as defined above, with a rigid backbone, a discrete set of rotamers, and pairwise energy functions has been proven to be NP-hard [74]. Hence, a variety of meta-heuristics have been applied to it, including Monte Carlo simulated annealing [53], genetic algorithms [77], and other algorithms [25]. The main weakness of these approaches is that they may remain stuck in local minima and miss the GMEC without notice.

However, there are several important motivations for solving the CPD problem exactly. First, because they know when an optimum is reached, exact methods may stop before meta-heuristics. Voigt et al. [84] reported that the accuracy of meta-heuristics also degrades as problem size increases. More importantly, the use of exact search algorithms becomes crucial in the usual experimental design cycle, that goes through modelling, solving, protein synthesis and experimental evaluation: when unexpected experimental results are obtained, the only possible culprit lies in the CPD model and not in the algorithm.

Current exact methods for CPD mainly rely on the dead-end elimination (DEE) theorem [24, 19] and the A^* algorithm [58, 33]. DEE is used as a pre-processing technique and removes rotamers that are locally dominated by other rotamers, until a fixpoint is reached. The rotamer r at position i (denoted by i_r) is removed if there exists another rotamer u at the same position such that [24]:

$$E(i_r) + \sum_{j \neq i} \min_s E(i_r, j_s) \geq E(i_u) + \sum_{j \neq i} \max_s E(i_u, j_s) \tag{2}$$

This condition guarantees that for any conformation with this r, we get a conformation with lower energy if we substitute u for r. Then, r can be removed from the list of possible rotamers at position i. This local dominance criterion was later improved by Goldstein [37] by directly comparing energies of each rotamer in the same conformation:

$$E(i_r) - E(i_u) + \sum_{j \neq i} \min_s [E(i_r, j_s) - E(i_u, j_s)] \geq 0 \tag{3}$$

where the best and worst-cases are replaced by the worst difference in energy. It is easy to see that this condition is always weaker than the previous one, and therefore applicable to more cases. These two properties define polynomial time algorithms that prune dominated values.

Since its introduction in 1992 by Desmet, DEE has become the fundamental tool of exact CPD, and various extensions have been proposed [73, 63, 32]. All these DEE criteria preserve the optimum but may remove suboptimal solutions. However CPD is NP-hard, and DEE cannot solve all CPD instances. Therefore, DEE pre-processing is usually followed by an A^* search. After DEE pruning, the A^* algorithm allows to expand a sequence-conformation tree, so that sequence-conformations are extracted and sorted on the basis of their energy values. The admissible heuristic used by A^* is described in [33].

When the DEE algorithm does not significantly reduce the search space, the A^* search tree can be too slow or memory demanding and the problem cannot be solved. Therefore, to circumvent these limitations and increase the ability of CPD to tackle problems with larger sequence-conformation spaces, novel alternative methods are needed. We now describe alternative state-of-the-art methods for solving the GMEC problem that offer attractive alternatives to DEE/A^*.

3. From CPD to CFN

CPD instances can be directly represented as Cost Function Networks.

Definition 1. *A Cost Function Network (CFN) is a pair (X, W) where $X = \{1, \ldots, n\}$ is a set of n variables and W is a set of cost functions. Each variable $i \in X$ has a finite domain D_i of values that can be assigned to it. A value $r \in D_i$ is denoted i_r. For a set of variables $S \subseteq X$, D_S denotes the Cartesian product of the domains of the variables in S. For a given tuple of values t, $t[S]$ denotes the projection of t over S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D_S \mapsto [0, k]$ where k is a maximum integer cost used for forbidden assignments.*

We assume, without loss of generality, that every CFN includes at least one unary cost function w_i per variable $i \in X$ and a nullary cost function $w_\varnothing$. All costs being non-negative, the value of this constant function, $w_\varnothing$, provides a lower bound on the cost of any assignment.

The Weighted Constraint Satisfaction Problem (WCSP) is to find a complete assignment t minimizing the combined cost function $\bigoplus_{w_S \in W} w_S(t[S])$, where $a \oplus b = \min(k, a + b)$ is the k-bounded addition. This optimization problem has an associated NP-complete decision problem. Notice that if $k = 1$, then the WCSP is nothing but the classical CSP (and not the Max-CSP).

Modeling the CPD problem as a CFN is straightforward. The set of variables X has one variable i per residue i. The domain of each variable is the set of *(amino acid,conformation)* pairs in the rotamer library used. The global energy function can be represented by 0-ary, unary and binary cost functions, capturing the constant energy term $w_\varnothing = E_\varnothing$, the unary energy terms $w_i(r) = E(i_r)$, and the binary energy terms $w_{ij}(r, s) = E(i_r, j_s)$, respectively. In the rest of the paper, for simplicity and consistency, we use notations $E_\varnothing$, $E(\cdot)$ and $E(\cdot, \cdot)$ to denote cost functions and restrict ourselves to binary CFN (extensions to higher orders are well-known).

Notice that there is one discrepancy between the original formulation and the CFN model: energies are represented as arbitrary floating point numbers while CFN uses positive costs. This can simply be fixed by first subtracting the minimum energy from all energies. These positive costs can then be multiplied by a large integer constant M and rounded to the nearest integer if integer costs are required.

3.1. Local consistency in CFN

The usual exact approach to solve a CFN is to use a depth-first branch-and-bound algorithm (DFBB). A family of efficient and incrementally computed lower bounds is defined by local consistency properties.

Node consistency [54] (NC) requires that the domain of every variable i contains a value r that has a zero unary cost ($E(i_r) = 0$). This value is called the unary support for i. Furthermore, in the scope of the variable i, all values should have a cost below k ($\forall r \in D_i, E_\varnothing + E(i_r) < k$).

Soft arc consistency (AC*) [79, 54] requires NC and also that every value r of every variable i has a support on every cost function $E(i_r, j_s)$ involving i. A support of i_r is a value $j_s \in D_j$ such that $E(i_r, j_s) = 0$.

Stronger local consistencies such as Existential Directional Arc Consistency (EDAC) have also been introduced [55]. See [14] for a review of existing local consistencies.

As in classical CSP, enforcing a local consistency property on a problem P involves transforming $P = (X, W)$ into a problem $P' = (X, W')$ that is equivalent to P (all complete assignments keep the same cost) and that satisfies the considered local consistency property. Enforcing a local consistency may increase $E_\varnothing$ and thus improve the lower bound on the optimal cost. This bound is used to prune the search tree during DFBB.

Local consistency is enforced using *Equivalence Preserving Transformations* (EPTs) that move costs between different cost functions [79, 54, 57, 18, 55, 15, 17, 16, 14]. For example, a variable i violating the NC property because all its values i_r have a non-zero $E(i_r)$ cost, can be made NC by subtracting the minimum cost from all $E(i_r)$ and adding this cost to $E_\varnothing$. The resulting network is equivalent to the original network, but it has an increased lower bound $E_\varnothing$.

Interestingly, in CPD, the admissible heuristic used in the DEE/A^* algorithm at depth d of the search tree is [33]:

$$\underbrace{\sum_{i=1}^{d}\Big[E(i_r) + \sum_{j=i+1}^{d} E(i_r, j_s)\Big]}_{\text{Assigned}} + \sum_{j=d+1}^{n} \Big[\underbrace{\min_s (E(j_s) + \sum_{i=1}^{d} E(i_r, j_s)}_{\text{Forward checking}} + \underbrace{\sum_{k=j+1}^{n} \min_u E(j_s, k_u))}_{\text{DAC counts}}\Big]$$

From a WCSP perspective, interpreting energies as cost functions, this heuristic is exactly the PFC-DAC lower bound [85, 56] used in WCSP. In WCSP, this lower bound is considered obsolete, and indeed it is proven to be weaker than soft arc consistency [79].

3.2. Maintaining dead-end elimination

Dead-end elimination is the key algorithmic tool of exact CPD solvers. From an AI perspective, in the context of CSP (if $k = 1$), the DEE Equation 3 is equivalent to neighborhood substitutability [27]. For MaxSAT, it is equivalent to the *Dominating 1-clause rule* [68]. In the context of CFN, the authors of [59] introduced partial soft neighborhood substitutability with a definition that is equivalent to Equation 3 for pairwise decomposed energies.

The DEE Equation 3 (cf. Section 2.1) can be strengthened and adapted to the CFN context as follows:

$$E(i_r) - E(i_u) + \sum_{j \neq i} \min_{s \atop E_\varnothing + E(i_r) + E(j_s) + E(i_r, j_s) < k} [E(i_r, j_s) - E(i_u, j_s)] \geq 0 \qquad (4)$$

This new condition differs from Equation 3 by the fact that some values have been discarded from the min operation. These values correspond to forbidden assignments

because the sum of the corresponding binary term plus the two unary costs plus the current lower bound $E_\varnothing$ (produced by soft arc consistency) is greater than or equal to the current upper bound k. Such values s do not need to be considered by the min operation because $\{i_r, j_s\}$ does not belong to any optimal solution, whereas $\{i_u, j_s\}$ may[1].

Example 1. *Let $X = \{1, 2, 3\}$ be a set of three variables with domains $D_1 = \{a, b, c\}$, $D_2 = \{e, f\}$, and $D_3 = \{g, h\}$. Suppose there are three cost functions, where $E(1_b) = 2$, $E(1_a, 2_e) = 2$, $E(1_b, 2_e) = E(1_c, 2_f) = 1$, $E(1_a, 3_g) = E(1_c, 3_h) = 2$, and all other costs are null. Let $k = 3$. The problem is EDAC. Then, 1_a dominates 1_b as shown by the new rule of Equation 4 that is satisfied: $E(1_b) - E(1_a) + E(1_b, 2_f) - E(1_a, 2_f) + \min(E(1_b, 3_g) - E(1_a, 3_g), E(1_b, 3_h) - E(1_a, 3_h)) = 2 - 0 + 0 - 2 \geq 0$, discarding tuple $\{2_e\}$ because $E(1_b, 2_e) + E(1_b) + E(2_e) + E_\varnothing = 1 + 2 + 0 + 0 \geq k$, whereas the old rule of Equation 3 is unsatisfied: $E(1_b) - E(1_a) + \min(E(1_b, 2_e) - E(1_a, 2_e), E(1_b, 2_f) - E(1_a, 2_f)) + \min(E(1_b, 3_g) - E(1_a, 3_g), E(1_b, 3_h) - E(1_a, 3_h)) = 2 - 0 - 1 - 2 < 0$.*

In the following, we recall how to enforce Equation 4 by an immediate adaptation of the original algorithm in [59]. Then, we present a modified version to partially enforce a novel combination of Equation 4 and Equation 2 with a much lower time complexity.

3.2.1. Enforcing DEE

Assuming a soft arc consistent WCSP (see *e.g.,* W-AC*2001 algorithm in [57]), enforcing DEE is described by Algorithm 1. For each variable i, all the pairs of values $(u, r) \in D_i \times D_i$ with $u < r$ are checked by the function DominanceCheck to see if r is dominated by u or, if not, vice versa (line 3). At most one dominated value is added to the value removal queue Δ at each inner loop iteration (line 2). Removing dominated values (line 4) can make the problem arc inconsistent, requiring us to enforce soft arc consistency again. Procedure AC*-DEE successively enforces AC* and DEE until no value removal is made by the enforcing algorithms.

Function DominanceCheck(i, u, r) computes the sum of worst-cost differences as defined by Equation 4 and returns a non-empty set containing value r if Equation 4 is true, meaning that r is dominated by value u. It exploits early breaks as soon as Equation 4 can be falsified (lines 5 and 6). Worst-cost differences are computed by the function getDifference(j, i, u, r) applied to every binary cost function related to i, discarding forbidden assignments with $\{i_r, j_s\}$ (line 8), as suggested by Equation 4. Worst-cost differences are always negative or zero (line 7) due to AC*.

The worst-case time complexity of getDifference is $O(d)$ for binary WCSPs. DominanceCheck is $O(nd)$ assuming a complete graph. Thus, the time complexity of one iteration of Algorithm 1 (DEE) is $O(nd^2nd + nd) = O(n^2d^3)$. Interleaving DEE and AC* until a fixed point is reached is done at most nd times, resulting in a worst-case time complexity in $O(n^3d^4)$. Its space complexity is $O(nd^2)$ when using the *residues* structure [59].

Note that using the new Equation 4 (line 8) or the Equation 3 (without line 8) does not change the complexities.

[1] Depending on the definition of soft arc consistency, from [54] (as presented in Section 3.1) or from [18], Equation 4 is stronger than or equivalent to Equation 3.

3.2.2. Enforcing DEE[1]

In order to reduce the time (and space) complexity of pruning by dominance, we test only one pair of values per variable. Hence the name, DEE[1], for the new algorithm described in Algorithm 2. We select the pair $(u, r) \in D_i \times D_i$ in an optimistic way such that u is associated with the minimum unary cost and r to the maximum unary cost (lines 9 and 10). Because arc consistency also implies node consistency, we always have $E(i_u) = 0$.[2] If all the unary costs (including the maximum) are equal to zero (line 11), we select as r the maximum domain value (or its minimum if this value is already used by u). By doing so, we should favor more pruning on max-closed or submodular subproblems[3].

Instead of just checking the new Equation 4 for the pair (u, r) alone, we use the opportunity to also check the original DEE rule of Equation 2 for all the pairs (u, v) such that $v \in D_i \setminus \{u\}$. This is done in the function MultipleDominanceCheck (lines 15 and 16). Notice that Equation 2 simplifies to $E(i_v) \geq ub_u$ (line 16) due to AC*. This function computes at the same time the sum of maximum costs ub_u for value u (lines 12 and 13) and the sum of worst-cost differences δ_{ur} for the pair (u, r). The new function getDifference-Maximum(j, i, u, r) now returns the worst-cost difference, as suggested by Equation 4, and also the maximum cost in $E(i, j)$ for i assigned u. When the maximum cost of a value is null for all its cost functions, we can directly remove all the other values in the domain avoiding any extra work (line 14). Finally, if the selected pair (u, r) for the variable i satisfies Equation 4, removing the value r of D_i, then a new pair for i will be checked at the next iteration of Algorithm 2 in the modified procedure AC*-DEE[1] (replacing Algorithm 1 by Algorithm 2 in AC*-DEE).

Notice that DEE[1] is equivalent to DEE on problems with Boolean variables, such as MaxSAT. For problems with non-Boolean domains, DEE[1] is still able to detect and prune several values per variable. Clearly, its time (resp. space) complexity is $O(n^3d^2)$ (resp. $O(n)$ using only one residue per variable), reducing by a factor d^2 the time and space complexity compared to DEE.

4. Computational Protein Design instances

In our initial experiments with CPD in [2], we built 12 designs using the CPD dedicated tool `osprey` 1.0. A new version of `osprey` being available since, we used this new 2.0 version [30] for all computations. Among different changes, this new version uses a modified energy field that includes a new definition of the "reference energy" and a different rotamer library. We therefore rebuilt the 12 instances from [2] and additionally created 35 extra instances from existing published designs, as described in [83]. We must insist on the fact that the 12 rebuilt instances do not define the same energy landscape or search space as the initial [2]'s instances (due to changes in rotamers set).

These designs include protein structures derived from the PDB that were chosen for the high resolution of their 3D-structures, their use in the literature, and their distribution of sizes and types. Diverse sizes of sequence-conformation combinatorial spaces

[2] In practice, we set the value u to the *unary support* offered by NC [54] or EDAC [55].

[3] Assuming a problem with two variables i and j having the same domain and a single submodular cost function, *e.g.,* $E(i_u, j_s) = 0$ if $u \leq s$ else $u - s$, or a single max-closed constraint, *e.g.,* $u < s$, then DEE[1] assigns $min(D_i)$ to i and $max(D_j)$ to j.

are represented, varying by the number of mutable residues, the number of alternative amino acid types at each position and the number of conformations for each amino acid. The *Penultimate* rotamer library was used [64]. Over these 47 designs, we only report results on the 40 designs for which a GMEC could be identified and proven by one of the tested solvers. All 47 designs are available for download both in native and WCSP formats at `http://genotoul.toulouse.inra.fr/~tschiex/CPD-AIJ`.

Preparation of CPD instances. Missing heavy atoms in crystal structures and hydrogen atoms were added with the *tleap* module of the AMBER9 software package [12]. Each molecular system was then minimized in implicit solvent (Generalized Born model [42]) using the *Sander* program and the all-atom *ff99* force field of AMBER9. All $E_{\varnothing}$, $E(i_r)$, and $E(i_r, j_s)$ energies of rotamers (see Equation 1) were pre-computed using `osprey` 2.0. The energy function consisted of the Amber electrostatic, van der Waals and the solvent terms. Rotamers and rotamer pairs leading to sterical clashes between molecules are associated with huge energies (10^{38}) representing forbidden combinations. For n residues to optimize with d possible (amino acid,conformation) pairs, there are n unary and $\frac{n.(n-1)}{2}$ binary cost functions that can be computed independently.

Translation to WCSP format. The native CPD problems were translated to the WCSP format before any pre-processing. To convert the floating point energies of a given instance to non-negative integer costs, we subtracted the minimum energy to all energies and then multiplied energies by an integer constant M and rounded to the nearest integer. The initial upper bound k is set to the sum, over all cost functions, of the maximum energies (excluding forbidden sterical clashes). High energies corresponding to sterical clashes are represented as costs equal to the upper bound k (the forbidden cost). The resulting WCSP model was used as the basis for all other solvers (except `osprey`). To keep a cost magnitude compatible with all the compared solvers, we used $M = 10^2$. Experiments with a finer discretization ($M = 10^8$) was used in previous experiments [83] with no significant difference in computing efforts.

4.1. A new cost-based variable ordering heuristics

We analyzed the distribution of costs for the CPD problem in order to infer a new variable ordering heuristics. Figure 3-left shows the histogram of a typical binary cost function for one of the CPD instances (1ENH, one of the open instances). Although the distribution has several modes, we chose to collect as an *important feature of a cost function* its median cost, which is less sensitive to extrema than the mean cost.

Figure 3-right shows the histogram of median costs for this instance. The problem has 666 binary cost functions and we collected the median cost in every cost function. The distribution of median costs has a heavy right tail. This feature can be exploited during search to focus on the most important variables first. For that, we define a new dynamic variable ordering heuristics selecting at each node of the search tree the variable minimizing the ratio of its current domain size divided by the sum of the median costs of all its current cost functions (including its unary cost function). The sum of the median costs gives a rough estimate of the average lower bound increase if we select that variable, relating this heuristics to *strong branching* in Operations Research [62, 1]. In order to save computation time, median costs of binary cost functions are computed only once, just after enforcing EDAC (and DEE), before the search.

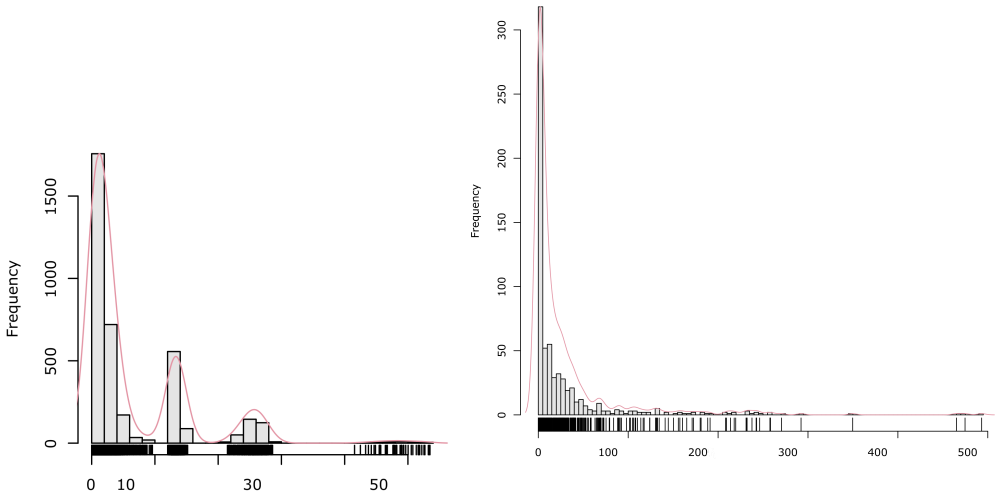

Figure 3: Histograms of $E(1, 12)$ costs (left) and of median costs of all binary cost functions (right) for the 1ENH instance ($M = 10^2$).

5. Alternative models for the CPD

The rigid backbone CPD problem has a simple formulation and can be easily written in a variety of combinatorial optimization frameworks. To evaluate CFN algorithms, the new DEE[1] algorithm and our domain specific heuristics, we compared these different variants with a variety of other solvers, coming from different fields. We present now the different models used in the comparison.

5.1. CPD as a probabilistic graphical model

The notion of *graphical model* has been mostly associated with *probabilistic graphical models*, the most famous examples of these are Markov random fields and Bayesian networks [49]. In those formalisms, a concise description of a joint distribution of probabilities over a set of variables is obtained through a factorization in local terms, involving only few variables. For terms involving at most two variables, if vertices represent variables and edges represent terms, a factorization can be represented as a graph, hence the name of *graphical models*. The same idea is used for concisely describing set of solutions (relations) in CSP or cost distributions in CFN.

Definition 2. *A discrete Markov random field (MRF) is a pair* (X, Φ) *where* $X = \{1, \ldots, n\}$ *is a set of* n *random variables and* Φ *is a set of potential functions. Each variable* $i \in X$ *has a finite domain* D_i *of values that can be assigned to it. A potential function* $\phi_S \in \Phi$*, with scope* $S \subseteq X$*, is a function* $\phi_S : D_S \mapsto \mathbb{R}$*.*

A discrete Markov random field (MRF) implicitly defines a non-normalized probability distribution over X. For a given tuple t, the probability of t is defined as:

$$P(t) = \frac{\exp(-\sum_{\phi_S \in \Phi} \phi_S(t[S]))}{Z}$$

where Z is a normalizing constant.

From the sole point of view of optimization, the problem of finding an assignment of maximum probability, also known as the maximum a posteriori (MAP) assignment in a MRF or a minimum cost solution of a CFN (the Weighted CSP) are equivalent by monotonicity of the exp() function. Some technical differences remain: CFN are restricted to non-negative costs (and some tools are restricted to integer costs). Being focused on optimization, CFN also emphasizes the possible existence of a finite upper bound k that leads to the use of bounded addition to combine costs instead of plain addition of potentials in MRFs.

The CPD problem can therefore directly be modeled as the MAP problem in a MRF exactly as we have described for CFN before, additive using potentials to capture energies (see for example [86]). Combinations of values with cost k (forbidden) are mapped to an infinite additive potential or a 0 value if multiplicative (exponential) potentials are used.

These models can be solved using MAP-MRF solvers such as `daoopt` [69] (winner of the Pascal Inference Challenge in 2011[4]) or the recent version of the `mplp` [80] solver.

5.2. Integer linear programming model

A 0/1 linear programming (01LP) problem is defined by a linear criterion to optimize over a set of Boolean variables under a conjunction of linear equalities and inequalities. The previous optimization problem over a graphical model can also be represented as a 01LP problem using the encoding proposed in [51].

For every assignment i_r of every variable i, there is a Boolean variable d_{ir} that is equal to 1 iff $i = r$. Additional constraints enforce that exactly one value is selected for each variable. For every pair of values of different variables (i_r, j_s) involved in a binary energy term, there is a Boolean variable p_{irjs} that is equal to 1 iff the pair (i_r, j_s) is used. Constraints enforce that a pair is used iff the corresponding values are used. Then, finding a GMEC reduces to the following ILP:

$$\min \sum_{\substack{i,r \\ E(i_r)\neq k}} E(i_r).d_{ir} + \sum_{\substack{i,r,j,s \\ j>i, E(i_r,j_s)\neq k}} E(i_r, j_s).p_{irjs}$$

$$\text{s.t.} \quad \sum_r d_{ir} = 1 \quad (\forall i) \tag{5}$$

$$\sum_s p_{irjs} = d_{ir} \quad (\forall i, r, j) \tag{6}$$

$$d_{ir} = 0 \quad (\forall i, r) E(i_r) = k \tag{7}$$

$$p_{irjs} = 0 \quad (\forall i, r, j, s) E(i_r, j_s) = k \tag{8}$$

$$d_{ir} \in \{0, 1\} \quad (\forall i, r) \tag{9}$$

$$p_{irjs} \in \{0, 1\} \quad (\forall i, r, j, s) \tag{10}$$

[4] See `http://www.cs.huji.ac.il/project/PASCAL/`.

This model is also the ILP model IP1 proposed in [48] for side-chain positioning. It has a quadratic number of Boolean variables. Constraints (7) and (8) explicitely forbid values and pairs with cost k (sterical clashes).

This model can be simplified by relaxing the integrality constraint on the p_{irjs}: indeed, if all d_{ir} are set to 0 or 1, the constraints (5) and (6) enforce that the p_{irjs} are set to 0 or 1. The same observation has been previously done for in the context of the linearization of a quadratic optimization model for wind farm design in [87]. In the rest of the paper, except where it is otherwise mentioned, we relax constraint (10). This type of ILP model can be handled by various ILP solvers such as IBM ILOG `cplex`.

5.3. 0/1 quadratic programming model

A 01QP problem is defined by a quadratic criterion to optimize over a set of Boolean variables under a conjunction of linear equality and inequality constraints. A compact encoding of the problem can be obtained using the ability of expressing the product of Boolean variables, getting rid of a quadratic number of p_{irjs} variables of the 01LP model.

For every value i_r, there is again a Boolean variable d_{ir} that is equal to 1 iff $i = r$. Additional linear constraints enforce that exactly one value is selected for each variable. The use of a given pair of rotamers at positions (i_r, j_s) can then be simply captured by the product $d_{ir}.d_{js}$. Then, finding a GMEC reduces to the following compact QP:

$$\min \sum_{i,r} E(i_r).d_{ir} + \sum_{\substack{i,r,j,s \\ j>i}} E(i_r, j_s).d_{ir}.d_{js}$$

$$\begin{aligned}
\text{s.t.} \quad \sum_r d_{ir} &= 1 \quad (\forall i) \\
d_{ir} &\in \{0,1\} \quad (\forall i, r) \\
d_{ir} &= 0 \quad (\forall i, r) E(i_r) = k \qquad (11) \\
d_{ir} + d_{js} &\leq 1 \quad (\forall i, r, j, s) E(i_r, j_s) = k \qquad (12)
\end{aligned}$$

Values and pairs generating sterical clashes are explicitely forbidden by constraints (11) and (12). This model can be handled by the QP solver of IBM ILOG CPLEX.

5.4. 0/1 quadratic optimization model

Another compact model can be obtained in the more restricted case of pure Boolean Quadratic Optimization (BQO), where a quadratic criterion is optimized but no linear constraints can be expressed.

For every value i_r, there is again a Boolean variable d_{ir} that is equal to 1 iff $i = r$. We must integrate the fact that exactly one value must be selected in each domain in the criterion itself. To capture the fact that there is at most one value selected per domain, we penalize the simultaneous selection of every pair i_r, i_s of rotamers of the same variable i with a sufficiently large penalty M. To guarantee that at least one value will be selected in each domain, we shift all finite energies by a constant negative term N such that all shifted finite energies are strictly negative. If an assignment selects no value in a given domain, then selecting one value can only result in an assignment with a lower cost, by

introducing new negative terms in the global energy. An optimal solution must therefore contain exactly one value per domain.

The corresponding model can be written as:

$$\min \sum_{i,r} (E(i_r) - N).d_{ir} + \sum_{\substack{i,r,j,s \\ j>i}} (E(i_r, j_s) - N).d_{ir}.d_{js} + \sum_{\substack{i,r,s \\ s>r}} M.d_{ir}.d_{is}$$

For N,we just use the largest negative integer that is strictly below the opposite of the largest finite energy in a given instance. M must be chosen in such a way that no combination of energy can compensate for the cost M. The selection of one additional value i_r can just contribute to the criterion by the addition of the energy $E(i_r)$ and the energies $E(i_r, j_s)$ for all other variables j and their rotamers j_s. M is therefore set to the opposite of the largest negative integer below the most negative sum of these energies, overall all variables i and rotamers i_r.

The corresponding Boolean quadratic optimization problem can be solved using the semidefinite programming based exact best-first branch-and-bound solver `biqmac` [78].

5.5. Weighted partial MaxSAT

Definition 3. *A weighted partial MaxSAT (WPMS) instance is a set of pair $\langle C, w\rangle$, where C is a clause and w is a number in $\mathbb{N} \cup \{\infty\}$, which is called the* weight *of that clause. A clause is a disjunction of literals. A literal is a Boolean variable or its negation.*

If the weight of a clause is ∞, it is called a *hard* clause, otherwise it is a *soft* clause. The objective is to find an assignment to the variables appearing in the clauses such that all hard clauses are satisfied and the weight of all falsified soft clauses is minimized.

The CPD problem can be encoded into a WPMS instance. We present two encodings, which are based on existing translations of CSP into SAT: the *direct* encoding [5], which is closer to the CFN model, and the *tuple* encoding, which was presented but not named by Bacchus [6] and is quite similar to the ILP model.

Direct encoding. In the direct encoding, we have one proposition d_{ir} for each variable/value pair (i, r), which is true if variable i is assigned the value r. We have hard clauses $(\neg d_{ir} \vee \neg d_{is})$ for all $i \in [1, n]$ and all $r < s$, $r, s \in D_i$, as well as a hard clause $(\bigvee_r d_{ir})$ for all i. These clauses ensure that the propositional encoding of the CFN allows exactly one value for each variable. The cost functions are represented respectively by an empty clause with weight $E_\varnothing$, unit clauses $\neg d_{ir}$ with weight $E(i_r)$ and binary clauses $\neg d_{ir} \vee \neg d_{js}$ with weight $E(i_r, j_s)$.

Tuple encoding. The tuple encoding encodes variable domains the same way as the direct encoding, therefore we have a proposition d_{ir} for each variable/value pair $i = r$, along with clauses that enforce that each variable is assigned exactly one value. The constant and unary energy terms are also respectively represented as an empty soft clause with weight $E_\varnothing$ and soft unit clauses $\neg d_{ir}$ with weight $E(i_r)$.

For all non-zero pairwise energy term $E(i_r, j_s)$, we have a proposition p_{irjs} as well as the soft clause $(\neg p_{irjs})$ with weight $E(i_r, j_s)$. This represents the cost to pay if the corresponding pair (energy term) is used. We also have the hard clauses $(d_{ir} \vee \neg p_{irjs})$ and $(d_{js} \vee \neg p_{irjs})$. These enforce that if a pair is used, the corresponding values must

be used. Finally, for all the pairs of variables (i, j) and all the values i_r, hard clauses $(\neg d_{ir} \vee \bigvee_{s \in D_j} p_{irjs})$ enforce that if a value i_r is used, one of the pair $p_{irj.}$ must be used.

This encoding is similar to the 01LP encoding and was originally proposed in the context of SAT encodings for classical CSP [6]. Unit Propagation (UP) on the tuple encoding enforces arc consistency in the original CSP (the set of values that are deleted by enforcing AC have their corresponding literal set to false by UP).

5.6. Constraint programming model

In [72], a generic translation of WCSPs into crisp CSPs with extra cost variables has been proposed. In this transformation, the decision variables remain the same as in the original WCSP and every cost function is reified into a constraint, which applies on the original cost function scope augmented by one extra variable representing the assignment cost. This reification of costs into domain variables transforms a WCSP in a crisp CSP with more variables and augmented arities. Typically, unary and binary cost functions are converted into `table` constraints of arity two and three respectively. Another extra cost variable encodes the global GMEC criterion, related by a `sum` constraint to all the unary and binary cost variables. All the cost variables are positive integer bounded by the same initial upper bound k as in the WCSP format.

The resulting CSP model has been expressed in the `minizinc` [65] constraint programming (CP) language. It can be solved using any CP solvers such as `gecode`, `mistral`, or `Opturion/CPX`, the recent winner of the MiniZinc Challenge 2013.

6. Experimental results

For computing the GMEC, all computations were performed on a single core of an AMD Operon 6176 at 2.3 GHz, 128 GB of RAM, and a 9,000-second time-out. These computations were performed on the GenoToul cluster.

6.1. Solvers tested

The solvers tested have different configurability in terms of parameters. Solvers such as `mplp` offer essentially no tuning, while others offer a large number of options. SAT solvers that participate routinely in the SAT competition have excellent default settings and those settings were kept unmodified. For one solver that explicitly requires tuning, we contacted the author for some advice. There is always a question whether dramatically different results could be obtained by different settings. The situation here corresponds to the situation of a non-naive user faced with several optimization tools.

DEE/A optimization.* The underlying principles of DEE/A* have been described in Section 2.1. To solve the different protein design cases, we used `osprey` version 2.0 (`cs.duke.edu/donaldlab/osprey.php`). The procedure starts by extensive DEE preprocessing (*algOption* = 3, includes simple Goldstein, Magic bullet pairs, 1 and 2-split positions, Bounds and pairs pruning) followed by A^* search. Only the GMEC conformation is generated by A^* (*initEw*=0).

CFN solver. `toulbar2` is a depth-first branch-and-bound solver using soft local consistencies for bounding and specific variable and value ordering heuristics for efficiency. The default EDAC [55] consistency may simultaneously reformulate all the cost functions involving one variable (a star subgraph). The default variable ordering strategy is based on the Weighted Degree heuristics [8] with Last Conflict [60], while the default value ordering consists in choosing for each variable its *fully supported value* as defined by EDAC.

We used `toulbar2` version 0.9.6 (`mulcyber.toulouse.inra.fr/projects/toulbar2/`) using binary branching and an initial limited discrepancy search phase [41] with discrepancy less than or equal to 1. We tested this vanilla version (options `-d: -l=1 -dee=0`) and incrementally introduced our new cost-based variable ordering heuristics (option `-m`) and different levels of DEE processing: maintaining DEE[1] during search (`-dee=1`), pre-processing with DEE (`-dee=4`), both together (`-dee=2`), or maintaining DEE during search (`-dee=3`).

`daoopt` solver. We decided to include `daoopt` as the winning solver of the 2011 PASCAL probabilistic inference challenge in the ''MAP" category. We downloaded `daoopt` version 1.1.2 from its repository (`https://github.com/lotten/daoopt`) and contacted the author for some advice. The distributed version of `daoopt` is not the same as the PIC challenge version. It lacks the Dual Decomposition bound strengthening component [69] that relies on private code.

This solver relies on Stochastic Local Search for finding initial solutions followed by depth-first AND/OR search [22] and mini-bucket lower bounds [23] for pruning. Mini-bucket lower bounds require space and time in $O(d^i)$ (where i is a user controlled parameter). CPD is certainly not an ideal domain for `daoopt`: the complete graph makes AND/OR search useless and the large maximum domain size d makes mini-buckets space and time intensive. We used the ''1 hour" settings for the PIC challenge from [69], modified to account for the complete graph that makes optimization of the AND/OR decomposition useless. This leads to the parameters `-i 35 --slsX 10 --slsT 6 -lds 1` and tried to allocate different amounts of memory to mini-buckets (option `-m` with 500MB, 5GB or 50GB), the i parameter being then automatically set by the solver to use a maximum amount of memory. We kept only the results for the best tuning (5GB, the worst results being obtained with 50GB). Note that because of large domain sizes, and the $O(d^i)$ space complexity of mini-buckets, a fine tuning of this parameter should have limited influence on the results.

The WCSP instances were transformed into the UAI ''MARKOV" format through the application of an exponential transformation of costs into multiplicative potentials. Costs above the upper bound were translated to zero potentials to preserve pruning. The exponential basis was chosen so that the largest multiplicative potentials are equal to 1.

MPLP MAP-MRF solver. We downloaded the sources for the recent version 2 of the `mplp` (Message Passing Linear Programming) implementation [80, 81] available at `http://cs.nyu.edu/~dsontag/`.

This solver uses a Message Passing based bound and duality theory to identify optimal solutions of a MAP-MRF problem through successive tightening of subsets of variables. The message passing used in `mplp` defines reparametrizations of the underlying MRF. These reparametrizations are similar to the reformulations done by local consistencies in

CFN [79, 18]. The solver is unique in all the solvers considered in that it does not use branching but only increasingly strong inference by applying reparametrizations to set of variables that initially contain only pairwise potentials, reasoning on stars [35], and are incrementally enlarged to include several potentials and strengthen the corresponding bound [81, 80].

All costs were divided by 1,000 and the optimality gap threshold kept to the default of $2 \cdot 10^{-4}$. The solver does not have any parameter.

ILP and QP optimization. We used `cplex` version 12.2 with parameters EPAGAP, EPGAP, and EPINT set to zero to avoid premature stop. No other tuning was done.

Boolean quadratic optimization. We used the `biqmac` [78] solver (`http://biqmac.uni-klu.ac.at/biqmaclib.html`) from sources provided by Angelika Wiegele. `biqmac` is a branch-and-bound solver relying on a strong Semi-Definite Programming (SDP) bound for Boolean quadratic optimization. The SDP framework is known to provide strong bounds for a variety of combinatorial optimization problems among which MaxCut and Max2SAT, with guaranteed approximation ratios [36]. Two solver settings (with branching rule set to 2 or 3, as advised by the author) were tried with no significant difference in the performances.

Weighted partial MaxSAT optimization. The same problems have been translated to WPMS using the two previously described encodings. There are two categories of complete WPMS solvers that we consider here: branch-and-bound (B&B) solvers and sequence-of-SAT solvers.

- Sequence-of-SAT solvers reformulate the WPMS problem as a series of SAT instances that allow us to successively increase the lower bound or decrease the upper bound for the optimal solution of the WPMS instance. A particular technique used by several sequence-of-SAT solvers, such as `WPM1` [4] and `maxhs` [20], is identifying *unsatisfiable cores* of the WPMS instance. An unsatisfiable core is a subset of the soft clauses of the instances which, taken together with the hard clauses of the instances, cannot all be satisfied by any assignment. The sequence of SAT instances then builds a collection of cores. The last SAT instance produces an assignment that violates at least one clause from each core but satisfies all other clauses. This assignment can be shown to be optimal.

- B&B solvers explore a backtracking search tree. At each node of the tree, they compute a lower bound on the cost of the best solution that can be found in the subtree rooted at that node. If that lower bound is higher than the cost of the best solution found so far, the solver backtracks. The solver `minimaxsat` [43] employs a method that is typically used in B&B solvers. In its case, the lower bound computation consists in performing unit propagation over the entire formula, including soft clauses. Unit propagation is able to detect some but not all unsatisfiable cores of the reduced formula at the current node. These cores are collected and used to transform the formula into an equivalent formula with a higher lower bound.

As B&B solvers, we have used `akmaxsat` [52] as it was among the best B&B performers in the latest MaxSAT evaluation and `minimaxsat` [43], which was shown to be one of the

best solvers over all the instances of all MaxSAT evaluations in [21]. Among sequence-of-SAT solvers, we have used `bin-c-d`, `wpm1` and `wpm2`, which are among the best performers in recent evaluations, as well as `maxhs`, which was shown to be the best solver for the entire ensemble of instances of MaxSAT evaluations [21].

We can observe that there exists a bijection between cores of the direct encoding of an instance and cores of the tuple encoding. However, there exist cores in the tuple encoding that can be detected just by unit propagation, but require a longer refutation in the direct encoding. On the other hand, the tuple encoding is larger and hence unit propagation is slower. Since both B&B and sequence-of-SAT solvers essentially rely on collecting cores of the formula, both types of solver can benefit from the tuple encoding by detecting more cores with less search. However, the overhead of performing unit propagation on a larger formula may not pay off in runtime.

CP solvers. We used `gecode` version 4.2.0 (`http://www.gecode.org/`), `mistral` version 1.3.40 (using its Python interface `numberjack` at `http://numberjack.ucc.ie/` and `http://github.com/eomahony/Numberjack/tree/fzn`), and `Opturion/CPX` version 1.0.2 (`http://www.opturion.com/cpx.html`). `mistral` uses a Weighted Degree heuristics [8] and a restart strategy with geometric factor 1.3 and base 256. `Opturion/CPX` combines CP and SAT solving techniques, learning clauses from failures. By default, it uses a *Luby* restart policy. No tuning was done for `gecode` nor `Opturion/CPX` (both using *free search* mode).

All the Python and C translating scripts used are available together with the CPD instances at `http://genotoul.toulouse.inra.fr/~tschiex/CPD-AIJ`.

6.2. Results

Several solvers were unable to solve any of the instances in the 9,000 seconds allocated per problem for each overall approach (including any preprocessing used in the method such as DEE in the DEE/A* approach). Despite the compact associated models, neither `cplex` for the quadratic programming model, nor `biqmac` for the quadratic Boolean optimization model could solve any single instance in less than 9,000 seconds. Similarly, most of the WPMS solvers failed to solve any instance, in either of the two encoding tested. The only exception to this is the `maxhs` solver when applied to the tuple encoding. Finally, neither `Opturion/CPX` nor `gecode` nor `mistral` could solve any single instance. In Table 1, we therefore only report the results obtained by the WPMS solver `maxhs`, the CPD solver `osprey`, the ILP solver `cplex`, the MAP-MRF solvers `daoopt` and `mplp`, and the CFN solver `toulbar2` in its vanilla version (using the default variable ordering heuristics and no DEE).

The detailed results are given in Table 1. The table shows that the cpu-times are very well correlated across different models and solvers, and show a clear ordering in terms of difficulty of these problems for all solvers, from WPMS/`maxhs`, MAP-MRF/`daoopt`, DEE/A*/`osprey`, ILP/`cplex`, MAP-MRF/`mplp`, and CFN/`toulbar2`.

The variant of the ILP model originally proposed by [51], where the p_{irjs} variables are constrained to be 0/1 variables was also tested. It was overall less efficient than the relaxed model we used. The ratio in terms of speedup was never very important (between 0.2 and 3.4 with a mean of 1.4 over all the solved instances) showing the robustness of `cplex`. It is often claimed, following [86], that LP technology is not able to deal with large instances of MRF. This experiment, on realistically designed instances of CPD,

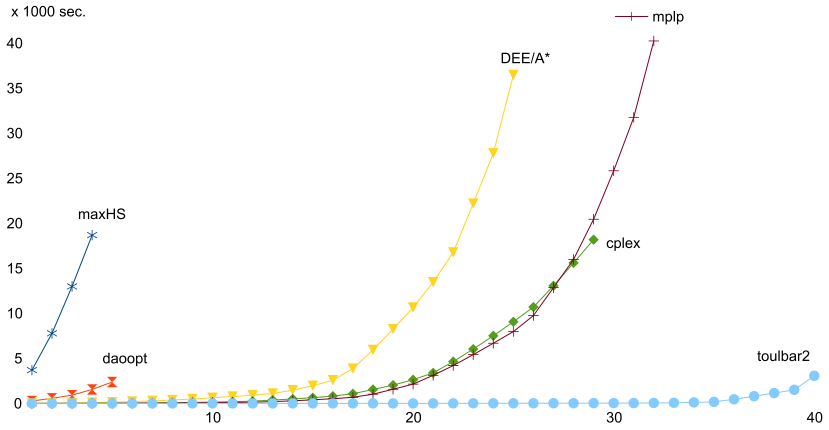

Figure 4: A figure showing the number of problems that can be solved by each approach (X-axis) as a function of cumulative time (Y-axis), assuming that each solver tackles problems in increasing order of cpu-time needed to solve it.

using state-of-the-art energy functions, including sterical clashes, shows that the recent 12.2 version of `cplex` gives reasonably good results on these problems.

6.3. Non-vanilla `toulbar2`

The results obtained on the same 40 CPD instances using the vanilla `toulbar2`, enhanced with our new variable ordering heuristics and increasingly stronger DEE processing are given in Table 2. None of the 7 open unreported instances could be solved by these new variants.

The new variable ordering heuristics consistently offer improved results. The effect of additional DEE processing is mostly visible on the difficult instances, the most visible and persistent improvements being obtained when using DEE in pre-processing, and for some instances (*e.g.,* 1BRS, 1RIS) also maintaining DEE^1 during search. They offer speedups up to 6 (on 1GVP). Further tests on a variety of CFN benchmarks (`http://costfunction.org`) are reported in [34]. They show that DEE^1 allows to solve more problems and DEE^1 is now a default option of `toulbar2`.

6.4. Analysis of results

It is unusual to apply such a wide range of NP-complete solving methods on a common set of benchmarks. Most comparisons are usually performed on a closely related family of solvers, sharing a common modeling language (SAT, CSP, MRF...).

Solvers are complex systems involving various mechanisms. The effect of their interactions during solving is hard to predict. Therefore, explaining the differences in efficiency observed between the different approaches is not straightforward. However, given the simplicity of our encodings, the fact that these instances are challenging for some approaches while at the same time being simpler to solve for other approaches should provide a source of inspiration for solver designers.

Quadratic programming and Quadratic optimization. One of the first surprising results is the difficulty of these instances for quadratic programming with `cplex`. The quadratic model is very dense with nd Boolean variables only. `cplex` is a totally closed-source black box but the behavior of the solver provides some information on its weak spot here. On the simplest problems, QP/`cplex` consumes memory very quickly and grows a very large node file. On the simple 2TRX problem ($n = 11$, first line of Table 1), QP/`cplex` solver explored $51,003,970$ nodes and was interrupted by the time-out with an optimality gap of 774%. This indicates a poor lower bound that leads to memory intensive best first search. On bigger problems, the number of nodes is never large because each node takes quite a time to explore. On the 1UBI problem ($n = 13, d = 148$), it explored only 5,233 nodes with an unbounded gap. It is therefore reasonable to assume that the lower bound used by `cplex` is too slow to compute on these problems and does not provide the additional strength that would compensate for the computing cost.

The model we devised for BQO using `biqmac` is compact, with the same $n.d$ 0/1 variables. `biqmac` uses a semidefinite programming lower bound that is known to provide among the strongest polynomial time lower bounds for a variety of optimization problems [76]. Despite this, even the smallest CPD instances could not be solved. We tried to extend the 9,000-second deadline for the simplest instance. After several hours of computing, `biqmac` stopped and reported that only a few nodes had been explored. The SDP technology used in `biqmac` may provide excellent bounds, but the time needed to compute them is currently too large to offer a viable alternative for CPD. The `biqmac` library at `http://biqmac.uni-klu.ac.at/biqmaclib.html` contains a variety of QP and (closely related) MaxCut problems that can be used for benchmarking. We tested `toulbar2` on the 10 `beasley` instances of size $n = 100$. They are solved in less than 1 second each by `toulbar2`, whereas `biqmac` took around 1 minute each, as reported in [78].

Integer linear programming. Considering 01LP, it is known that the continuous LP relaxation of the 0/1 linear programming model we used in Section 5.2 is the dual of the LP problem encoded by Optimal Soft Arc Consistency (OSAC) [17, 14] when the upper bound k used in CFN is infinite. OSAC is known to be stronger than any other soft arc consistency level, including EDAC and Virtual Arc Consistency (VAC) [16]. However, as soon as the upper bound k used for pruning in CFN decreases to a finite value, soft local consistencies may prune values and EDAC becomes incomparable with the dual of these relaxed LPs. To better evaluate the pruning power of `cplex`, we compared the number of nodes it explored with those explored by `toulbar2` in its vanilla mode or with the new heuristics and DEE pre-processing. Table 3 shows that among the 28 instances solved, 18 are solved by `cplex` before search starts, 7 are solved by the non-vanilla version of `toulbar2` w/o backtracks. For the remaining less trivial problems, the number of nodes explored by `cplex` and `toulbar2` are often similar with no clear winner. Overall, these results show comparable pruning power. It also shows that the problems solved by `cplex` are relatively simple problems but the computation of the lower bound is quite expensive in `cplex`. It typically develops from 1 to 50 nodes per minute while `toulbar2` develops from 1 to 40 thousand nodes per minute. Note that the problems that are not solved by `cplex` are much harder, requiring more than 120,000 nodes to explore for the hardest solved problem (not shown in the Table).

Markov Random Field MAP. The relaxed LP is also equivalent, in the pairwise case, to the LP relaxation of MRFs in the so-called *local polytope* [81]. In its original version [35], `mplp` is only guaranteed to produce this LP bound if domains are Boolean. It is therefore weaker than OSAC for CFN and comparable to Virtual AC [14]. With the recent additions described in [81, 80], `mplp` has the ability to incrementally tighten its bound by performing local inference on several potentials (or cost functions) organized in cyclic structures. The strength of this lower bound is such that `mplp` version 2 is often able to prove optimality based just on this bound and the cost of the assignment that optimizes unary reparameterized potentials. Still, weaker but faster CFN lower bounds combined with search apparently offer a better solution on these realistic CPD instances.

In the MRF community, the inefficiency of pure LP on large MRF instances is well-known [86]. These experiments show that, combined with branching, the incrementality of the LP bound allows 01LP to get very decent results on these problems. However, the quadratic size of the 01LP model probably explains the better efficiency of `mplp`.

MaxSAT. The most surprising result is probably the difficulty of these problems for MaxSAT solvers, either branch-and-bound based or core-based. To analyze branch and bound based algorithm behavior, we instrumented the two solvers MiniMaxSat and `akmaxsat` to report the best upper bound found and the number of nodes explored. Additionally, `akmaxsat` reports the lower bound computed at the root node of the search tree. In the direct encoding, MiniMaxSat is fast and may explore up to 36 thousand nodes per second (two orders of magnitude faster than `toulbar2`). In 15 problems, it was able to identify sub-optimal solutions ending up with a non-trivial upper bound (within 3.2% to 0.26% of the optimum) but never started the final optimality proof, showing a weak lower bound. Indeed, the lower bound computed by `akmaxsat` at the root of the search tree is never higher than 27% of the optimum. In contrast, the lower bound computed by `toulbar2` at the root was often 99% of the optimum and never less than 97%. We conclude that the direct encoding does not allow for strong propagation and lower bouds.

The tuple encoding was chosen to put WPMS solvers in a situation where UP applied to a hardened version of the formula would be able to detect more unsatisfiable cores. Since this operation is at the heart of the lower bounding procedures of such solvers, it should allow the derivation of a stronger lower bound. Additionally, unit propagation on the hardened version of the tuple encoding is equivalent to enforcing arc consistency on the hardened CFN. In CFN, VAC precisely identifies subproblems whose hardened version is arc inconsistent (and therefore define inconsistent cores) to increase the lower bound. VAC is known to be capable of producing stronger lower bounds than the default local consistency EDAC [55] used in `toulbar2`. Hence, the tuple encoding provides enough information to give better lower bounds than what EDAC computes. The empirical results verify that the lower bound computed at the root is much stronger with the tuple encoding than with the direct encoding. For several instances `akmaxsat` computes a lower bound that is 92% of the optimum. However, this is still far from the lower bound computed by `toulbar2`. But the more important problem with this encoding is that with a quadratic number of extra variables, both minimaxsat and `akmaxsat` were extremely slow, exploring at most 2 nodes before the 9,000 second time-out and in several instances timed out before even finishing the lower bound computation at the root node. They never produced a single incumbent assignment.

On the other hand, the `maxhs` core-based solver is able to exploit the stronger tuple encoding, being able to solve 4 problems to optimality. Analyzing the behaviour of `maxhs` on these instances reveals that the solver spends almost all of its time trying to reduce the size of the cores it finds, using a greedy minimization algorithm. This is because these protein design instances contain some very large cores (tens of thousands of clauses) which can not be significantly reduced in size. Such cores arise, for example, from the binary cost functions in the CFN model. In this case, the core expresses the condition that the cost of at least one tuple in the cost function will be incurred, and the core therefore contains as many clauses as there are tuples in the originating cost function. We observed that as a result, `maxhs` is usually unable to complete its initial disjoint core phase within the timeout. Given this observation, we also experimented with running `maxhs` with the core minimization option turned off, on the set of 12 instances which update those in [2], using the tuple encoding. With core minimization turned off, `maxhs` was able to complete the disjoint core phase on all 12 instances. This allowed us to compare the lower bound produced by the disjoint core phase of `maxhs` with the lower bound produced at the root node by `toulbar2`. Over the 12 instances, the lower bound produced by `maxhs` was between 84% and 98% of the lower bound produced by `toulbar2`, and it was calculated within only 35 seconds except for two cases. Note that this bound calculated by `maxhs` differs from that of `toulbar2` in that `maxhs` uses a complete SAT solver to find the cores, and the cores are strictly disjoint. Based on these observations, we believe the potential to improve the performance of the `maxhs` approach on these instances is very promising.

For other core-based solvers, either because of the quadratic number of variables or because of a different exploitation of non-AC/UP cores, these CPD instances remain very hard. Whether it is a fundamental, technical, or implementation difference, identifying the cause of this difference should allow to improve the existing WPMS technology.

Constraint programming. The generic translation of WCSPs into crisp CSPs suffers from the large magnitude of costs, resulting in large domains for the extra cost variables with very slow arc consistency propagation of `table` constraints for `mistral`, developing approx. 215 nodes per minute. In comparison, `Opturion/CPX` develops 721 nd/min. It also requires huge memory space for expressing the `table` constraints. Only 22 instances among the 42 could fit into 128 GB during `minizinc` to `flatzinc` translation. By dividing all costs by 100 (*i.e.,* $M = 1$), `mistral` was able to solve 4 instances: 1CSK in 964 seconds and 90,562 nodes, 1HZ5 ($d = 45$) in (759 s& 179,319 nd), 1PGB ($d = 45$) in (76 s& 29,939 nd), and 2TRX ($n = 11$) in (47.3 s& 28,057 nd). `gecode` (resp. `Opturion/CPX`) solved only one: 2TRX in (2,234 s& 213,423 nd) (resp. 1PGB in (6,916 s& 204,597 nd)). For the unsolved instances, `Opturion/CPX` found better solutions than `mistral` on average. The difference in performances between the three solvers might come from the different search strategies, `mistral` used geometric restarts, whereas `Opturion/CPX` used *Luby* restarts, and `gecode` no restarts.

DEE/A.* The DEE/A* combination uses strong polynomial time dominance analysis using several variants of dead-end elimination. This pre-processing is followed by best-first search relying on an obsolete lower bound instead of the stronger lower bounds offered by soft local consistencies such as EDAC [55], or the LP relaxation bound. To confirm this, we computed the number of nodes explored by `osprey` during A* search.

Except for simple problems where DEE alone could solve the problem, `osprey` explored trees larger than those explored by ILP or CFN by several orders of magnitude. On problem 1DKT, it explored more than 10^7 nodes while ILP/`cplex` solved the problem without search and `toulbar2` explored 134 nodes. This confirms the weakness of current bounds in exact CPD algorithms. Otherwise, `osprey` is quite fast and can develop more than 110,000 nodes per minute. Despite the exploration of huge trees, no DEE/A* execution led to memory exhaustion before time-out. With an extended time-out of 100 hours [83], only 2 instances ultimately led to memory exhaustion. The replacement of A* by iterative alternatives to A^* such as IDA* [50] would therefore probably have little influence on the results of DEE/A*.

7. Conclusions

The simplest formal optimization problem underlying CPD looks for a Global Minimum Energy Conformation (GMEC) over a rigid backbone and altered side-chains (identity and conformation). In computational biology, exact methods for solving the CPD problem combine dominance analysis (DEE) and an A^* search.

The CPD problem can also be directly formulated as a Cost Function Network, with a very dense graph and relatively large domains. We have shown how DEE can be integrated with local consistency with a reasonable time complexity.

The CPD can also be easily reduced to optimization in MRF, 01LP, 01QP, weighted partial MaxSAT, and Boolean quadratic optimization, offering an ideal benchmark for a large cross-technology comparison.

On a variety of real instances, we have shown that state-of-the-art optimization algorithms on graphical models exploiting bounds based on the reformulation (or reparametrization) of the graphical model, but also 01LP algorithms, give important speedups compared to usual CPD algorithms combining dead-end elimination with $A*$. Among all the tested solvers, `toulbar2` was the most efficient solver and its efficiency was further improved by the use of DEE during search.

We also showed that these CPD problems define challenging benchmarks for a variety of solvers, including weighted partial MaxSAT solvers, either branch-and-bound or core-based, and quadratic programming or quadratic optimization solvers, including semidefinite programming based solvers.

In practice, it must be stressed that just finding the GMEC is not a final answer to real CPD problems. CPD energies functions represent an approximation of the real physics of proteins and optimizing a target score based on them (such as stability, affinity,...) is not a guarantee of finding a successful design. Indeed, some designs may be so stable that they are unable to accomplish the intended biological function. The usual approach is therefore to design a large library of proteins whose sequences are extracted from all solutions within a small threshold of energy of the GMEC. This problem is also efficiently solved by `toulbar2` [83].

Although it is easy to formulate as a discrete optimization problem, another important limitation of the rigid backbone/rotamer CPD problem lies in the restrictions generated by these two assumptions. In practice, rotamers offer a continuous range of rotations along dihedral angles and backbones also have degrees of flexibility. Several approaches have been proposed and introduced in `osprey` in the last few years that relax either

or both of these two assumptions while still offering a guarantee of optimality [40, 29, 31]. When flexibility counts, `osprey` is therefore a reference tool. All these approaches ultimately require to solve the very same type of optimization problems involving a sum of precomputed pairwise *lower bounds* on energy terms. In this context, it becomes crucial to be able to enumerate all the solutions within a threshold of the optimum. These approaches should therefore ultimately also benefit from algorithmic improvements in GMEC optimization, as far as exhaustively enumerating all the solutions within a threshold of the optimum is feasible.

Acknowledgements

This work has been partly funded by the "Agence nationale de la Recherche" (ANR-10-BLA-0214 and ANR-12-MONU-0015-03), the INRA and the Region Midi-Pyrénées. We would like to thank Damien Leroux for his help in the generation of encodings using Python. We thank the Computing Center of Region Midi-Pyrénées (CALMIP, Toulouse, France) and the GenoToul Bioinformatics Platform of INRA-Toulouse for providing computing resources and support.

The Insight Centre for Data Analytics is supported by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

[1] Achterberg, T., Koch, T., Martin, A., 2005. Branching rules revisited. Operations Research Letters 33, 42–54.

[2] Allouche, D., Traoré, S., André, I., de Givry, S., Katsirelos, G., Barbe, S., Schiex, T., 2012. Computational protein design as a cost function network optimization problem, in: Principles and Practice of Constraint Programming, Springer. pp. 840–849.

[3] Anfinsen, C., 1973. Principles that govern the folding of protein chains. Science 181, 223–253.

[4] Ansótegui, C., Bonet, M.L., Levy, J., 2009. Solving (weighted) partial maxsat through satisfiability testing, in: Theory and Applications of Satisfiability Testing-SAT 2009, Springer. pp. 427–440.

[5] Argelich, J., Cabiscol, A., Lynce, I., Manyà, F., 2008. Encoding Max-CSP into partial Max-SAT, in: Multiple Valued Logic, 2008. ISMVL 2008. 38th International Symposium on, IEEE. pp. 106–111.

[6] Bacchus, F., 2007. GAC via unit propagation, in: Principles and Practice of Constraint Programming–CP 2007, Springer. pp. 133–147.

[7] Boas, F.E., Harbury, P.B., 2007. Potential energy functions for protein design. Current opinion in structural biology 17, 199–204. URL: `http://www.ncbi.nlm.nih.gov/pubmed/17387014`, doi:`10.1016/j.sbi.2007.03.006`.

[8] Boussemart, F., Hemery, F., Lecoutre, C., Sais, L., 2004. Boosting systematic search by weighting constraints, in: ECAI, p. 146.

[9] Bowie, J.U., Luthy, R., Eisenberg, D., 1991. A method to identify protein sequences that fold into a known three-dimensional structure. Science 253, 164–170.

[10] Campeotto, F., Dal Pal, A., Dovier, A., Fioretto, F., Pontelli, E., 1991. A constraint solver for flexible protein models. Science 253, 164–170.

[11] Carothers, J.M., Goler, J.A., Keasling, J.D., 2009. Chemical synthesis using synthetic biology. Current opinion in biotechnology 20, 498–503.

[12] Case, D., Darden, T., Cheatham III, T., Simmerling, C., Wang, J., Duke, R., Luo, R., Merz, K., Pearlman, D., Crowley, M., Walker, R., Zhang, W., Wang, B., Hayik, S., Roitberg, A., Seabra, G., Wong, K., Paesani, F., Wu, X., Brozell, S., Tsui, V., Gohlke, H., Yang, L., Tan, C., Mongan, J., Hornak, V., Cui, G., Beroza, P., Mathews, D., Schafmeister, C., Ross, W., Kollman, P., 2006. Amber 9. Technical Report. University of California. San Francisco.

[13] Champion, E., André, I., Moulis, C., Boutet, J., Descroix, K., Morel, S., Monsan, P., Mulard, L.A., Remaud-Siméon, M., 2009. Design of α-transglucosidases of controlled specificity for programmed chemoenzymatic synthesis of antigenic oligosaccharides. Journal of the American Chemical Society 131, 7379–7389.

[14] Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T., 2010. Soft arc consistency revisited. Artificial Intelligence 174, 449–478.
[15] Cooper, M.C., 2005. High-order consistency in Valued Constraint Satisfaction. Constraints 10, 283–305.
[16] Cooper, M.C., de Givry, S., Sánchez, M., Schiex, T., Zytnicki, M., 2008. Virtual arc consistency for weighted CSP., in: Proc. of AAAI'08, pp. 253–258.
[17] Cooper, M.C., de Givry, S., Schiex, T., 2007. Optimal soft arc consistency, in: Proc. of IJCAI'2007, Hyderabad, India. pp. 68–73.
[18] Cooper, M.C., Schiex, T., 2004. Arc consistency for soft constraints. Artificial Intelligence 154, 199–227.
[19] Dahiyat, B.I., Mayo, S.L., 1996. Protein design automation. Protein science : a publication of the Protein Society 5, 895–903. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2143401\&tool=pmcentrez\&rendertype=abstract, doi:10.1002/pro.5560050511.
[20] Davies, J., Bacchus, F., 2011. Solving MAXSAT by solving a sequence of simpler SAT instances, in: Principles and Practice of Constraint Programming–CP 2011. Springer, pp. 225–239.
[21] Davies, J., Bacchus, F., 2013. Exploiting the power of MIP solvers in MaxSAT, in: Theory and Applications of Satisfiability Testing–SAT 2013, Springer. pp. 166–181.
[22] Dechter, R., Mateescu, R., 2007. AND/OR search spaces for graphical models. Artificial intelligence 171, 73–106.
[23] Dechter, R., Rish, I., 2003. Mini-buckets: A general scheme for bounded inference. Journal of the ACM (JACM) 50, 107–153.
[24] Desmet, J., De Maeyer, M., Hazes, B., Lasters, I., 1992. The dead-end elimination theorem and its use in protein side-chain positioning. Nature 356, 539–42. URL: http://www.ncbi.nlm.nih.gov/pubmed/21488406.
[25] Desmet, J., Spriet, J., Lasters, I., 2002. Fast and accurate side-chain topology and energy refinement (FASTER) as a new method for protein structure optimization. Proteins 48, 31–43. URL: http://www.ncbi.nlm.nih.gov/pubmed/12012335, doi:10.1002/prot.10131.
[26] Fersht, A., 1999. Structure and mechanism in protein science: a guide to enzyme catalysis and protein folding. WH. Freemean and Co., New York.
[27] Freuder, E.C., 1991. Eliminating interchangeable values in constraint satisfaction problems, in: Proc. of AAAI'91, Anaheim, CA. pp. 227–233.
[28] Fritz, B.R., Timmerman, L.E., Daringer, N.M., Leonard, J.N., Jewett, M.C., 2010. Biology by design: from top to bottom and back. BioMed Research International 2010.
[29] Gainza, P., Roberts, K.E., Donald, B.R., 2012a. Protein design using continuous rotamers. PLoS computational biology 8, e1002335.
[30] Gainza, P., Roberts, K.E., Georgiev, I., Lilien, R.H., Keedy, D.A., Chen, C.Y., Reza, F., Anderson, A.C., Richardson, D.C., Richardson, J.S., et al., 2012b. Osprey: Protein design with ensembles, flexibility, and provable algorithms. Methods Enzymol .
[31] Georgiev, I., Keedy, D., Richardson, J.S., Richardson, D.C., Donald, B.R., 2008a. Algorithm for backrub motions in protein design. Bioinformatics 24, i196–i204.
[32] Georgiev, I., Lilien, R.H., Donald, B.R., 2006. Improved Pruning algorithms and Divide-and-Conquer strategies for Dead-End Elimination, with application to protein design. Bioinformatics (Oxford, England) 22, e174–83. URL: http://www.ncbi.nlm.nih.gov/pubmed/16873469, doi:10.1093/bioinformatics/btl220.
[33] Georgiev, I., Lilien, R.H., Donald, B.R., 2008b. The minimized dead-end elimination criterion and its application to protein redesign in a hybrid scoring and search algorithm for computing partition functions over molecular ensembles. Journal of computational chemistry 29, 1527–42. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3263346\&tool=pmcentrez\&rendertype=abstract, doi:10.1002/jcc.20909.
[34] de Givry, S., Prestwich, S., O'Sullivan, B., 2013. Dead-end elimination for weighted CSP, in: Springer (Ed.), Principles and Practice of Constraint Programming–CP 2013.
[35] Globerson, A., Jaakkola, T.S., 2007. Fixing max-product: Convergent message passing algorithms for map lp-relaxations, in: Advances in neural information processing systems, pp. 553–560.
[36] Goemans, M.X., Williamson, D.P., 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) 42, 1115–1145.
[37] Goldstein, R.F., 1994. Efficient rotamer elimination applied to protein side-chains and related spin glasses. Biophysical journal 66, 1335–40. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1275854\&tool=pmcentrez\&rendertype=abstract, doi:10.1016/S0006-3495(94)80923-3.

[38] Gront, D., Kulp, D.W., Vernon, R.M., Strauss, C.E., Baker, D., 2011. Generalized fragment picking in rosetta: design, protocols and applications. PloS one 6, e23294.

[39] Grunwald, I., Rischka, K., Kast, S.M., Scheibel, T., Bargel, H., 2009. Mimicking biopolymers on a molecular scale: nano(bio)technology based on engineered proteins. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences 367, 1727–47. URL: http://www.ncbi.nlm.nih.gov/pubmed/19376768, doi:10.1098/rsta.2009.0012.

[40] Hallen, M.A., Keedy, D.A., Donald, B.R., 2013. Dead-end elimination with perturbations (deeper): A provable protein design algorithm with continuous sidechain and backbone flexibility. Proteins: Structure, Function, and Bioinformatics 81, 18–39.

[41] Harvey, W.D., Ginsberg, M.L., 1995. Limited discrepancy search, in: Proc. of the 14^{th} IJCAI, Montréal, Canada.

[42] Hawkins, G., Cramer, C., Truhlar, D., 1996. Parametrized models of aqueous free energies of solvation based on pairwise descreening of solute atomic charges from a dielectric medium. The Journal of Physical Chemistry 100, 19824–19839.

[43] Heras, F., Larrosa, J., Oliveras, A., 2008. Minimaxsat: An efficient weighted Max-SAT solver. J. Artif. Intell. Res.(JAIR) 31, 1–32.

[44] Janin, J., Wodak, S., Levitt, M., Maigret, B., 1978. Conformation of amino acid side-chains in proteins. Journal of molecular biology 125, 357–386.

[45] Khalil, A.S., Collins, J.J., 2010. Synthetic biology: applications come of age. Nature Reviews Genetics 11, 367–379.

[46] Khare, S.D., Kipnis, Y., Greisen, P., Takeuchi, R., Ashani, Y., Goldsmith, M., Song, Y., Gallaher, J.L., Silman, I., Leader, H., Sussman, J.L., Stoddard, B.L., Tawfik, D.S., Baker, D., 2012. Computational redesign of a mononuclear zinc metalloenzyme for organophosphate hydrolysis. Nature chemical biology 8, 294–300. URL: http://www.ncbi.nlm.nih.gov/pubmed/22306579, doi:10.1038/nchembio.777.

[47] Khoury, G.A., Smadbeck, J., Kieslich, C.A., Floudas, C.A., 2014. Protein folding and *de novo* protein design for biotechnological applications. Trends in biotechnology 32, 99–109.

[48] Kingsford, C.L., Chazelle, B., Singh, M., 2005. Solving and analyzing side-chain positioning problems using linear and integer programming. Bioinformatics (Oxford, England) 21, 1028–36. URL: http://www.ncbi.nlm.nih.gov/pubmed/15546935, doi:10.1093/bioinformatics/bti144.

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

[50] Korf, R.E., 1985. Depth first iterative deepening : An optimal admissible tree search. Artificial Intelligence 27, 97–109.

[51] Koster, A., van Hoesel, S., Kolen, A., 1999. Solving Frequency Assignment Problems via Tree-Decomposition. Technical Report RM/99/011. Universiteit Maastricht. Maastricht, The Netherlands.

[52] Kuegel, A., 2010. Improved exact solver for the weighted Max-SAT problem, in: Workshop Pragmatics of SAT.

[53] Kuhlman, B., Baker, D., 2000. Native protein sequences are close to optimal for their structures. Proceedings of the National Academy of Sciences of the United States of America 97, 10383–8. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=27033\&tool=pmcentrez\&rendertype=abstract.

[54] Larrosa, J., 2002. On arc and node consistency in weighted CSP, in: Proc. AAAI'02, Edmondton, (CA). pp. 48–53.

[55] Larrosa, J., de Givry, S., Heras, F., Zytnicki, M., 2005. Existential arc consistency: getting closer to full arc consistency in weighted CSPs, in: Proc. of the 19^{th} IJCAI, Edinburgh, Scotland. pp. 84–89.

[56] Larrosa, J., Meseguer, P., Schiex, T., Verfaillie, G., 1998. Reversible DAC and other improvements for solving max-CSP, in: Proc. of AAAI'98, Madison, WI.

[57] Larrosa, J., Schiex, T., 2004. Solving weighted CSP by maintaining arc consistency. Artif. Intell. 159, 1–26.

[58] Leach, A.R., Lemon, A.P., 1998. Exploring the conformational space of protein side chains using dead-end elimination and the A* algorithm. Proteins 33, 227–39. URL: http://www.ncbi.nlm.nih.gov/pubmed/9779790.

[59] Lecoutre, C., Roussel, O., Dehani, D.E., 2012. WCSP integration of soft neighborhood substitutability, in: Principles and Practice of Constraint Programming, Springer. pp. 406–421.

[60] Lecoutre, C., Saïs, L., Tabary, S., Vidal, V., 2009. Reasoning from last conflict(s) in constraint programming. Artificial Intelligence 173, 1592,1614.

[61] Lewis, J.C., Bastian, S., Bennett, C.S., Fu, Y., Mitsuda, Y., Chen, M.M., Greenberg, W.A., Wong, C.H., Arnold, F.H., 2009. Chemoenzymatic elaboration of monosaccharides using engineered cytochrome p450bm3 demethylases. Proceedings of the National Academy of Sciences 106, 16550–16555.

[62] Linderoth, J., Savelsbergh, M., 1999. A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing 11, 173–187.

[63] Looger, L.L., Hellinga, H.W., 2001. Generalized dead-end elimination algorithms make large-scale protein side-chain structure prediction tractable: implications for protein design and structural genomics. Journal of molecular biology 307, 429–45. URL: http://www.ncbi.nlm.nih.gov/pubmed/11243829, doi:10.1006/jmbi.2000.4424.

[64] Lovell, S.C., Word, J.M., Richardson, J.S., Richardson, D.C., 2000. The penultimate rotamer library. Proteins 40, 389–408. URL: http://www.ncbi.nlm.nih.gov/pubmed/10861930.

[65] Marriott, K., Nethercote, N., Rafeh, R., Stuckey, P., de la Banda, M.G., Wallace, M., 2008. The design of the zinc modelling language. Constraints 13, 229–267.

[66] Martin, V.J., Pitera, D.J., Withers, S.T., Newman, J.D., Keasling, J.D., 2003. Engineering a mevalonate pathway in escherichia coli for production of terpenoids. Nature biotechnology 21, 796–802.

[67] Nestl, B.M., Nebel, B.A., Hauer, B., 2011. Recent progress in industrial biocatalysis. Current Opinion in Chemical Biology 15, 187–193. URL: http://www.ncbi.nlm.nih.gov/pubmed/21195018, doi:10.1016/j.cbpa.2010.11.019.

[68] Niedermeier, R., Rossmanith, P., 2000. New upper bounds for maximum satisfiability. J. Algorithms 36, 63–88.

[69] Otten, L., Ihler, A., Kask, K., Dechter, R., 2012. Winning the pascal 2011 map challenge with enhanced AND/OR branch-and-bound, in: DISCML'12 Workshop, at NIPS'12, Lake Tahoe, NV, USA.

[70] Pabo, C., 1983. Molecular technology. Designing proteins and peptides. Nature 301, 200. URL: http://www.ncbi.nlm.nih.gov/pubmed/6823300.

[71] Peisajovich, S.G., Tawfik, D.S., 2007. Protein engineers turned evolutionists. Nature methods 4, 991–4. URL: http://www.ncbi.nlm.nih.gov/pubmed/18049465, doi:10.1038/nmeth1207-991.

[72] Petit, T., Régin, J., Bessière, C., 2000. Meta constraints on violations for over constrained problems, in: Proceedings of IEEE ICTAI'2000, Vancouver, BC, Canada. pp. 358–365.

[73] Pierce, N., Spriet, J., Desmet, J., Mayo, S., 2000. Conformational splitting: A more powerful criterion for dead-end elimination. Journal of computational chemistry 21, 999–1009.

[74] Pierce, N.A., Winfree, E., 2002. Protein design is NP-hard. Protein engineering 15, 779–82. URL: http://www.ncbi.nlm.nih.gov/pubmed/12468711.

[75] Pleiss, J., 2011. Protein design in metabolic engineering and synthetic biology. Current opinion in biotechnology 22, 611–7. URL: http://www.ncbi.nlm.nih.gov/pubmed/21514140, doi:10.1016/j.copbio.2011.03.004.

[76] Raghavendra, P., 2008. Optimal algorithms and inapproximability results for every CSP?, in: Proceedings of the 40th annual ACM symposium on Theory of computing, ACM. pp. 245–254.

[77] Raha, K., Wollacott, A.M., Italia, M.J., Desjarlais, J.R., 2000. Prediction of amino acid sequence from structure. Protein science : a publication of the Protein Society 9, 1106–19. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2144664\&tool=pmcentrez\&rendertype=abstract, doi:10.1110/ps.9.6.1106.

[78] Rendl, F., Rinaldi, G., Wiegele, A., 2010. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. Math. Programming 121, 307.

[79] Schiex, T., 2000. Arc consistency for soft constraints, in: Principles and Practice of Constraint Programming - CP 2000, Singapore. pp. 411–424.

[80] Sontag, D., Choe, D.K., Li, Y., 2012. Efficiently searching for frustrated cycles in MAP inference, in: Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI-12), AUAI Press, Corvallis, Oregon. pp. 795–804.

[81] Sontag, D., Meltzer, T., Globerson, A., Weiss, Y., Jaakkola, T., 2008. Tightening LP relaxations for MAP using message-passing, in: 24th Conference in Uncertainty in Artificial Intelligence, AUAI Press. pp. 503–510.

[82] Swain, M., Kemp, G., 2001. A CLP approach to the protein side-chain placement problem, in: Principles and Practice of Constraint Programming–CP 2001, Springer. pp. 479–493.

[83] Traoré, S., Allouche, D., André, I., de Givry, S., Katsirelos, G., Schiex, T., Barbe, S., 2013. A new framework for computational protein design through cost function network optimization. Bioinformatics 29, 2129–2136.

[84] Voigt, C.A., Gordon, D.B., Mayo, S.L., 2000. Trading accuracy for speed: A quantitative comparison of search algorithms in protein sequence design. Journal of molecular biology 299, 789–803. URL: http://www.ncbi.nlm.nih.gov/pubmed/10835284, doi:10.1006/jmbi.2000.3758.

[85] Wallace, R., 1995. Directed arc consistency preprocessing, in: Meyer, M. (Ed.), Selected papers from the ECAI-94 Workshop on Constraint Processing. Springer, Berlin. number 923 in LNCS, pp. 121–137.

[86] Yanover, C., Meltzer, T., Weiss, Y., 2006. Linear programming relaxations and belief propagation– an empirical study. The Journal of Machine Learning Research 7, 1887–1907.

[87] Zhang, P.Y., Romero, D.A., Beck, J.C., Amon, C.H., 2013. Solving wind farm layout optimization with mixed integer programming and constraint programming, in: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR), Springer. pp. 284–299.

Algorithm 1: Enforce DEE [59]

```
Procedure DEE((X, W): AC* consistent WCSP)
    Δ := ∅ ;
1   foreach i ∈ X do
2       foreach (u, r) ∈ D_i × D_i such that u < r do
            R := DominanceCheck(i, u, r) ;
3           if R = ∅ then R := DominanceCheck(i, r, u) ;
            Δ := Δ ∪ R ;
4   foreach i_r ∈ Δ do
        remove r from D_i ;
        Q := Q ∪ {i} ;

/* Check if value u dominates value r */
Function DominanceCheck(i, u, r): set of dominated values
    δ_ur := E(i_r) − E(i_u) ;
5   if δ_ur < 0 then return ∅ ;
    foreach j ∈ X \ {i} do
        δ := getDifference(j, i, u, r) ;
        δ_ur := δ_ur + δ ;
6       if δ_ur < 0 then return ∅ ;
    return {i_r} /* δ_ur ≥ 0 */ ;

/* Compute smallest difference in costs when using a instead of b */
Function getDifference(j, i, u, r): cost
7   δ_ur := 0 ;
    foreach s ∈ D_j do
8       if E(i_r, j_s) + E(i_r) + E(j_s) + E_∅ < k then
            δ_ur := min(δ_ur, E(i_r, j_s) − E(i_u, j_s)) ;
    return δ_ur ;

/* Enforce AC* and DEE */
Procedure AC*-DEE()
    Q := X ;
    while Q ≠ ∅ do
        W-AC*2001(Q) ;
        DEE(Q) ;
```

Algorithm 2: Enforce DEE^1

```
Procedure DEE^1((X, W): AC* consistent WCSP)
    Δ := ∅ ;
    foreach i ∈ X do
9       u := arg min_{v∈D_i} E(i_v) ;
10      r := arg max_{v∈D_i} E(i_v) ;
11      if u = r /* ∀v ∈ D_i, E(i_v) = 0 */ then
            if u = max(D_i) then
                r := min(D_i) ;
            else
                r := max(D_i) ;
        R := MultipleDominanceCheck(i, u, r) ;
        if R = ∅ then R := MultipleDominanceCheck(i, r, u) ;
        Δ := Δ ∪ R ;
    foreach i_r ∈ Δ do
        remove r from D_i ;
        Q := Q ∪ {i} ;

/* Check if value u dominates value r and possibly other values */
Function MultipleDominanceCheck(i, u, r): set of dominated values
    δ_ur := E(i_r) − E(i_u) ;
    if δ_ur < 0 then return ∅ ;
12  ub_u := E(i_u) ;
    foreach j ∈ X \ {i} do
        (δ, ub) := getDifference-Maximum(j, i, u, r) ;
        δ_ur := δ_ur + δ ;
13      ub_u := ub_u + ub ;
        if δ_ur < 0 then return ∅ ;
14  if ub_u = 0 then return {i_v | v ∈ D_i \ {u}} ;
    R := {i_r} /* δ_ur ≥ 0 */ ;
15  foreach v ∈ D_i \ {u} do
16      if (E(i_v) ≥ ub_u) then R := R ∪ {i_v} ;
    return R ;

/* Compute smallest cost difference and maximum cost for value u */
Function getDifference-Maximum(j, i, u, r): pair of costs
    δ_ur := 0 ;
    ub_u := 0 ;
    foreach s ∈ D_j do
        if E(i_r, j_s) + E(i_r) + E(j_s) + E_∅ < k then
            δ_ur := min(δ_ur, E(i_r, j_s) − E(i_u, j_s)) ;
        ub_u := max(ub_u, E(i_u, j_s)) ;
    return (δ_ur, ub_u) ;

/* Enforce AC* and DEE^1 */
Procedure AC*-DEE^1()
    Q := X ;
    while Q ≠ ∅ do
        W-AC*2001(Q) ;
        DEE^1(Q) ;
```

Table 1: For each instance: protein (PDB id.), number of mutable residues, maximum domain size (maximum number of rotamers), and CPU-time for solving using `maxhs`, `daoopt`, DEE/A*, `cplex`, `mplp`, and `toulbar2`. A '-' indicates that the corresponding solver did not prove optimality within the 9,000-second time-out. A '!' indicates the solver stops with a SEGV signal.

PDB id.	n	d	`maxhs`	`daoopt`	DEE/A*	`cplex`	`mplp`	`toulbar2`
2TRX	11	44	4,086	268.6	31.5	2.6	2.8	**0.1**
1PGB	11	45	5,209	300.4	135.3	3.6	0.5	**0.1**
1HZ5	12	45	5,695	350.2	75.0	7.6	16.7	**0.1**
1UBI	13	45	-	826.9	2,812.6	139.2	37.3	**0.2**
1PGB	11	148	-	-	8,695.2	-	1,291	**4.3**
1HZ5	12	148	-	-	2,398.3	1,555	1,217	**2.4**
1UBI	13	148	-	-	-	-	-	**1,557**
2PCY	18	44	-	-	1,281.1	26.9	14.5	**0.2**
2DHC	14	148	-	-	-	-	5,388	**14.1**
1CM1	17	148	-	-	138.4	473.1	87.5	**3.3**
1MJC	28	182	3,698	631.7	4.6	4.1	0.8	**0.1**
1CSP	30	182	-	-	200.0	1,380	1,264	**0.8**
1BK2	24	182	-	-	93.2	125.0	114.9	**0.6**
1SHG	28	182	-	-	138.0	39.4	!	**0.2**
1CSK	30	49	-	-	41.7	12.5	9.6	**0.1**
1SHF	30	56	-	-	44.3	8.6	3.1	**0.1**
1FYN	23	186	-	-	622.0	2,548	3,136	**2.8**
1PIN	28	194	-	-	-	-	-	**3.7**
1NXB	34	56	-	-	11.1	17.0	4.5	**0.2**
1TEN	39	66	-	-	113.0	45.4	17.1	**0.2**
1POH	46	182	-	-	77.9	29.0	13.1	**0.3**
2DRI	37	186	-	-	-	-	4,458	**42.8**
1FNA	38	48	-	-	3,310	124.9	121.2	**0.5**
1UBI	40	182	-	-	-	2,572	979.4	**2.4**
1C9O	43	182	-	-	2,310	1,635	155.7	**1.8**
1CTF	39	56	-	-	-	263.2	549.2	**0.7**
2PCY	46	56	-	-	2,080	54.0	20.3	**0.4**
1DKT	46	190	-	-	5,420	1,254	3,103	**2.5**
2TRX	61	186	-	-	487.0	765.0	344.1	**0.9**
1CM1	42	186	-	-	-	-	-	**17.4**
1BRS	44	194	-	-	-	-	-	**346.5**
1CDL	40	186	-	-	-	-	-	**341.8**
1LZ1	59	57	-	-	-	601.6	1,084	**1.5**
1GVP	52	182	-	-	-	-	-	**361.8**
1RIS	56	182	-	-	-	-	8,483	**288.4**
2RN2	69	66	-	-	-	480.8	565.2	**1.2**
1CSE	97	183	-	-	367.0	172.9	60.9	**0.7**
1HNG	85	182	-	-	5,590	2,360	5,934	**2.8**
3CHY	74	66	-	-	-	-	8,691	**59.6**
1L63	83	182	-	-	-	1,480	1,779	**2.9**
			4	5	25	29	33	**40**

Table 2: For each instance: CPU-time for solving using `toulbar2` and different combinations of options for DVO and DEE. A '-' indicates that the corresponding solver did not prove optimality with the 9,000-second time-out. The `tb2` column gives the results obtained using the vanilla `toulbar2` for reference. The DVO corresponds to the activation of the new variable ordering heuristics described in Section 4.1. This option is kept activated in all the remaining columns. These columns correspond respectively to additionally maintaining DEE^1 during search, pre-processing using DEE, doing both, and maintaining DEE during search. The last line reports the number of times a method was faster than the others.

PDB	n	d	`tb2`	DVO	DEE^1	DEE_{pre}	DEE_{pre} +DEE^1	DEE
2TRX	11	44	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
1PGB	11	45	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
1HZ5	12	45	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
1UBI	13	45	**0.2**	**0.2**	**0.2**	**0.2**	0.3	0.5
1PGB	11	148	4.3	3.8	3.4	**3.1**	8.6	15.1
1HZ5	12	148	2.4	2.4	2.3	**2.2**	3.1	3.5
1UBI	13	148	1,557	**1,068**	1,736	1,133	1,162	-
2PCY	18	44	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
2DHC	14	148	14.1	8.0	**7.0**	**7.0**	14.5	52.0
1CM1	17	148	3.3	**3.1**	3.2	**3.1**	**3.1**	**3.1**
1MJC	28	182	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
1CSP	30	182	0.8	0.6	**0.5**	0.7	0.7	0.8
1BK2	24	182	0.6	0.6	0.6	**0.5**	0.7	**0.5**
1SHG	28	182	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
1CSK	30	49	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
1SHF	30	56	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
1FYN	23	186	2.8	2.9	**2.6**	3.0	3.2	3.8
1PIN	28	194	3.7	**3.0**	**3.0**	4.8	6.2	12.0
1NXB	34	56	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
1TEN	39	66	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
1POH	46	182	**0.3**	**0.3**	**0.3**	0.4	0.4	0.4
2DRI	37	186	42.8	16.4	37.7	**9.6**	15.5	51.2
1FNA	38	48	0.5	0.4	**0.3**	0.4	0.4	0.5
1UBI	40	182	2.4	1.0	**0.7**	0.9	0.9	1.3
1C9O	43	182	1.8	**1.5**	1.7	2.3	2.4	3.6
1CTF	39	56	0.7	0.9	**0.6**	**0.6**	0.7	0.8
2PCY	46	56	**0.4**	**0.4**	**0.4**	**0.4**	**0.4**	**0.4**
1DKT	46	190	2.5	2.8	**2.4**	2.6	2.7	3.9
2TRX	61	186	**0.9**	**0.9**	**0.9**	1.8	1.7	1.9
1CM1	42	186	17.4	11.8	13.2	**8.6**	11.6	20.0
1BRS	44	194	346.5	241.4	135.4	70.4	**60.1**	129.0
1CDL	40	186	341.8	198.1	159.0	**79.6**	128.8	286.4
1LZ1	59	57	1.5	1.1	1.0	**0.9**	1.0	1.1
1GVP	52	182	361.8	248.5	408.2	**38.3**	66.8	163.5
1RIS	56	182	288.4	147.4	77.9	37.8	**28.8**	122.8
2RN2	69	66	1.2	**1.1**	**1.1**	1.2	**1.1**	1.2
1CSE	97	183	0.7	0.8	**0.6**	**0.6**	**0.6**	**0.6**
1HNG	85	182	2.8	2.4	**2.3**	3.1	2.8	3.6
3CHY	74	66	59.6	27.9	10.7	**10.6**	14.9	20.3
1L63	83	182	2.9	2.8	**2.3**	2.4	2.5	2.7
			14	19	26	25	16	14

Table 3: For each instance solved by both `cplex` and `toulbar2`, we report the number of nodes explored by each solver (with the number of backtracks in parentheses when available) and the number of nodes per minute developed. `toulbar2` is the vanilla version, `toulbar2`$^+$ uses the new variable ordering heuristics and DEE as pre-processing.

PDB id.	n	d	`cplex`		`toulbar2`		`toulbar2`$^+$	
			nodes	nd/min	nodes (bt)	nd/min	nodes (bt)	nd/min
2TRX	11	44	0	-	8 (0)	6857	10 (1)	7500
1PGB	11	45	0	-	17 (1)	11333	16 (1)	10667
1HZ5	12	45	0	-	25 (5)	15000	29 (7)	17400
1UBI	13	45	51	22.0	143 (61)	39000	82 (31)	24600
1HZ5	12	148	0	-	89 (34)	2225	54 (16)	1453
2PCY	18	44	0	-	53 (6)	13826	40 (9)	10909
1CM1	17	148	0	-	14 (0)	258	0 (0)	0
1MJC	28	182	0	-	22 (0)	14667	2 (0)	1714
1CSP	30	182	547	23.8	540 (245)	42078	42 (16)	3877
1BK2	24	182	3	1.4	28 (3)	2800	19 (3)	2375
1SHG	28	182	214	326	268 (101)	69913	51 (16)	19125
1CSK	30	49	0	-	38 (5)	20727	14 (3)	7636
1SHF	30	56	0	-	35 (4)	17500	12 (0)	6000
1FYN	23	186	0	-	84 (20)	1819	43 (8)	863
1NXB	34	56	0	-	30 (0)	9474	14 (0)	4667
1TEN	39	66	0	-	75 (6)	20455	22 (5)	6600
1POH	46	182	0	-	111 (5)	21484	15 (0)	2195
1FNA	38	48	0	-	189 (47)	25200	84 (28)	12600
1UBI	40	182	287	6.7	1,669 (766)	41900	539 (228)	36337
1C9O	43	182	49	1.8	222 (57)	7525	82 (16)	2112
1CTF	39	56	94	21.4	294 (95)	24845	110 (33)	10820
2PCY	46	56	0	-	62 (5)	10629	20 (0)	3158
1DKT	46	190	0	-	210 (36)	5122	134 (24)	3045
2TRX	61	186	6	0.5	111 (14)	7239	85 (16)	2818
1LZ1	59	57	735	73.5	807 (308)	31855	178 (53)	11609
2RN2	69	66	0	-	105 (17)	5385	110 (17)	5500
1CSE	97	183	0	-	94 (0)	8418	9 (0)	915
1HNG	85	182	48	1.2	411 (110)	8745	96 (21)	1870
1L63	83	182	0	-	196 (17)	4055	58 (3)	1468