Fast search algorithms for Computational Protein Design

Journal:	*Journal of Computational Chemistry*
Manuscript ID:	JCC-15-0201
Wiley - Manuscript type:	Full Paper
Date Submitted by the Author:	14-Apr-2015
Complete List of Authors:	Barbe, Sophie; INRA, LISBP
Key Words:	Computational protein design, Computer-aided protein design, Exact combinatorial optimization, Cost Function Networks, Deterministic search methods, Search heuristics, Global Minimum Energy Conformation, Near-optimal solutions

Journal of COMPUTATIONAL CHEMISTRY WWW.C-CHEM.ORG FULL PAPER

Fast search algorithms for Computational Protein Design

Seydou Traoré[1-3], Kyle E. Roberts[4], David Allouche[5], Bruce R. Donald[4], Isabelle André[1-3], Thomas Schiex[5], Sophie Barbe[1-3*]

[1] Université de Toulouse; INSA,UPS,INP; LISBP, 135 Avenue de Rangueil, F-31077 Toulouse, France
[2] INRA, UMR792, Ingénierie des Systèmes Biologiques et des Procédés, F-31400 Toulouse, France
[3] CNRS, UMR5504, F-31400 Toulouse, France
[4] Department of Biochemistry, Department of Computer Science, Department of Chemistry, Duke University, Durham, NC, USA
[5] Unité de Biométrie et Intelligence Artificielle, UR 875, INRA, F-31320 Castanet Tolosan, France

* Correspondence to: Sophie Barbe, Laboratoire d'Ingénierie des Systèmes Biologiques et des Procédés – INSA; CNRS UMR5504; UMR INRA 792; 135, Avenue de Rangueil; F-31077 Toulouse cedex 4, France. Tel: +33 561 559 963; Fax: +33 561 559 400; E-mail: sophie.barbe@insa-toulouse.fr

ABSTRACT

In a previous work, we demonstrated the efficiency of Cost Function Network (CFN) optimization on various Computational Protein Design (CPD) problems against a broad range of combinatorial optimization technologies including 0/1 Linear and Quadratic Programming, 0/1 Quadratic Optimization, Weighted Partial MaxSAT, Graphical Model optimization problems and the commonly used CPD dedicated framework Dead-End Elimination/A* (DEE/A*). Motivated by these results, the aim of the work disclosed herein is to integrate the CFN optimization technology into the well-established CPD package *Osprey* and to further develop new CPD-dedicated search algorithms and heuristics.

After the direct implementation of all major CFN algorithms and heuristics in *Osprey*, new Best-First methods using the CFN lower bound as heuristics have been developed. The performance of the algorithms was examined using a benchmark set of 30 CPD instances. These approaches bring important speedups compared to the well-known DEE/A^* framework for solving numerous CPD problems. New Side Chain Positioning (SCP)–based upper bounding heuristics have been also introduced in order to improve the CFN-based search. This led to an important increase in the pruning efficiency at the root node. Based on these upper bounding strategies, new amino acid-based branching schemes were designed to incrementally perform SCP-based upper bounding during the search. The immediate and important speedups provided by CFN technology, these developments enable the design of larger protein systems. Finally, this new branching scheme enables also to efficiently enumerate sub-optimal solutions over a larger energy range.

Introduction

Computational Protein Design (CPD) has become a valuable tool for creating proteins with desired biophysical and functional properties and for assessing our understanding of protein sequence-structure-function relationships. By combining physico-chemical models governing relations between protein amino-acid composition and the protein three-dimensional structure with advanced computational algorithms, CPD seeks to identify one or a set of amino-acid sequences that fold into a given 3D-scaffold and that will bestow the re-designed protein with targeted properties. Such rational protein design approaches have been successfully applied to alter intrinsic properties (stability, binding affinity…) of existing proteins or to endow them with new functionalities, leading to the generation of novel enzymatic catalysts, therapeutic proteins, protein-protein interfaces and self-assembling protein structures [1]–[6]. The applications of this technology is broad, ranging from medicine, biotechnology, and synthetic biology to nanotechnologies [7].

Despite notable results, substantial methodological advances are still needed to improve CPD performances and extend its effective application. The success of CPD predictions depends on several elements, which include the biologically meaningful modeling of the design problem, the accuracy of the energy and objective functions used to assess fitness of the predicted sequence-structures, and the efficiency of the search algorithms to find solutions in a timely manner. However, CPD approaches have to strike a compromise between speed and accuracy to face the exponential size of the search space defined by the

composition of protein sequences and conformations. Most of the CPD methods rely on: 1) a coarse-graining of the structure as a sequence of discrete side-chains rotamers, 2) an assumption of modest backbone conformational flexibility, where often a fixed backbone or a set of possible backbones are used, and 3) an approximation of the energy model as a pairwise decomposition. Since the problem of searching for an optimal solution (GMEC : Global Minimum-Energy Conformation) over the conformational space of rotamers and possibly backbones is NP-hard [8], a variety of methods, both meta-heuristics (Monte Carlo simulated annealing [9], [10], genetic algorithms,…[11]) and provable algorithms (Dead-End Elimination (DEE), Branch-and-Bound algorithms (BB), Integer Linear Programming (ILP), Dynamic Programming (DP),…[11]–[16]) have been proposed over the years. However, there is still a need for more efficient optimization techniques, capable of exploring vaster combinatorial spaces, representing more realistic and flexible protein models.

This paper focuses on exact optimization techniques. Provable methods know when a global optimum is reached, the search can be stopped with confidence and an exact solution obtained, sometimes in significantly less time than with meta-heuristics. It has also been observed that the accuracy of meta-heuristic approaches tends to degrade in unpredictable ways as the problem size increases [9]. Finally, exact methods are useful for improving biophysical models because they ensure that discrepancies between CPD predictions and experimental results come exclusively from modeling inadequacies and not from the algorithm. Currently, the most usual provable and deterministic methods for CPD rely on the Dead-End Elimination theorem and the A^* algorithm [12]. DEE is used as a pre-processing method. It removes rotamers which are energetically dominated by other rotamers and therefore useless to identify a global optimum. This idea has later been extended to prune pairs or higher order combinations of rotamers at different residues in order to improve the pruning power [17]–[19]. However, because CPD is NP-hard, the polynomial time DEE algorithms usually cannot identify a unique sequence-conformation model. It is followed by an A^* search (a provable Best-First search) which expands a sequence-conformation tree by tentatively assigning rotamers to residues. By relying on a dedicated admissible heuristic, A^* is able to produce an energy-sorted list of solutions. But the worst-case exponential time and memory consumption of A^* means that it can easily choke on problems with many undominated rotamers.

In a recent work, we have shown that the rigid backbone discrete rotamer CPD problem could be formulated and efficiently solved as a Cost Function Network (CFN)[20]–[22]. CFN algorithms are able to handle complex combinatorial spaces which are out of reach of usual DEE/*A** approaches as implemented in the CPD-dedicated software *Osprey* [1], [23]. The *toulbar2* CFN solver provides speedups of several orders of magnitude both to provably find the GMEC and to exhaustively enumerate ensembles of near-optimal solutions, offering an attractive alternative to the usual DEE/*A** approaches.

Herein, we integrated the CFN technology in a well-established CPD framework to take advantage of the speedup benefits offered by CFN algorithms on the large variety of problems tackled by *Osprey*, including those that offer increased flexibility modeling in CPD [24]. A wider description of the CPD-dedicated algorithms present in the *Osprey* package can be found elsewhere [25]. It is noteworthy that *Osprey* has been prospectively used, with experimental validation, to redesign enzymes toward non-cognate substrates [1], [26], [27], design new drugs [28], predict drug resistance mutations [2], design peptide inhibitors of protein-protein interactions [29], and design epitope-specific antibody probes [30]. All CPD-dedicated methods already integrated in *Osprey* to handle these various design problems have motivated the selection of this software to render our CFN-based approach accessible to the CPD community.

We also report in the current study further methodological advances derived from this CFN-based method [19]–[21] which were implemented within the *Osprey* software. The performances of these methods have been assessed on the design of more stable proteins and cofactor-bound proteins, as well as protein-ligand and protein-protein interfaces. The results were compared to those obtained using the DEE/A* approach implemented in *Osprey*[1].

Background

The CPD problem formulation

The rigid backbone and discrete rotamer CPD problem is defined by: *i*) a fixed backbone of a protein structure; *ii*) a set of amino-acid residues to be designed, called 'designable residues'; *iii*) a group of allowed amino-acids for each designable residue and their respective set of discrete low energy side-chain conformations, called rotamers and *iv*) pairwise atomic energy functions to rank the models. Rotamers correspond to cluster centers of well represented amino-acid side-chain conformations mined from a database of 3D protein structures. In the case of protein-ligand systems, the conformational flexibility of peptide ligands is described similarly by discrete rotamer libraries. In the case of non-peptide ligands, the treatment of organic molecule flexibility is often left to the user to pre-calculate an ensemble of low energy conformers for the ligand (playing the role of rotamer library) [4].

[1] The modified *Osprey* source code consistent with this work is available at http://www.cs.duke.edu/donaldlab/osprey.versions.php#CFNosprey

Journal of COMPUTATIONAL CHEMISTRY WWW.C-CHEM.ORG FULL PAPER

A sequence-conformation model is defined by the choice of one specific amino-acid with one associated conformation (rotamer) for each designable residue. Its total energy (E_{total}) is defined by:

$$\mathrm{E_{total}} = \mathrm{E_c} + \sum_{\mathrm{i}} \mathrm{E(i_r)} + \sum_{i} \sum_{\mathrm{j}<i} \mathrm{E(i_r, j_s)} \quad (1)$$

where E_c is a constant energy contribution capturing interactions between fixed parts of the model, $E(i_r)$ depends on rotamer r at position i (and its reference energy) and $E(i_r, j_s)$ is the pairwise interaction energy between rotamer r at position i and rotamer s at position j.

The combinatorial optimization problem is to find a complete rotamer assignment (a sequence and a conformation) that provably minimizes E_{total}.

Modeling CPD as a Cost Function Network

The problem of finding the set of rotamers that will minimize the total energy (E_{total}) can be easily formulated as a Cost Function Network problem (CFN), also known as a Weighted Constraint Satisfaction Problem (WCSP) [20]–[22].

A CFN P is defined by a set of variables which are each involved in a set of local cost functions [31]. Formally, a CFN P is a triple $P = (X, D, C)$ where $X = \{1,2,..,n\}$ is a set of n variables. Each variable $i \in X$ has a discrete domain $d_i \in \mathrm{D}$ that defines the set of values that it can take. A set of local cost functions C defines a network over X. Each cost function $c_S \in C$is defined over a subset of variables $S \subseteq X$ (called its scope), has a domain $\prod_{i \in S} d_i$ and takes integer values in $\{0,1,2,..,k\}$. The cost k represents a maximum tolerable cost, and can be infinite or set to a finite upper bound. When cost functions involve at most two variables, the CFN is said to be binary. Values or pairs of values that are forbidden by a cost function are simply mapped to k. The global cost of a complete assignment A is defined as the sum of all cost functions on this assignment (or k if this sum is larger than k). The WCSP defined by P consists in finding an assignment of all variables that minimizes this global cost. Notice that it is usually assumed that C contains one constant cost function, with an empty scope, denoted as $c_\emptyset$. Since all cost functions in a CFN are non-negative, this constant cost function $c_\emptyset \in C$ defines a lower bound on the optimization problem.

The CPD optimization problem, in its pairwise-decomposed form, can be easily formulated as a binary CFN. Every designable amino-acid residue i is represented by a variable i and the set of rotamers available to the residue defines its domain d_i. Then, each energy term in E_{total} is represented as a cost function [20]–[22]. The constant term E_c is captured as the constant cost function with empty scope ($c_\emptyset$) and terms $E(i_r)$ and $E(i_r, j_s)$ are represented by unary and binary cost functions involving the variables of the corresponding residues. Energy terms can be mapped to positive integers through shifting and scaling according to desired precision [20]–[22]. Such operations preserve the set of optimal solutions and an optimal solution of the CFN is an assignment that defines a GMEC for the CPD problem.

In contrast with CPD, where dominance analysis through the DEE theorem is widely used, the fundamental idea in CFNs relies on so-called *Local Consistencies*. From a given CFN, it is always possible to define a family of small (local) sub-problems. A local consistency property requires that each sub-problem must be sufficiently explicit about local optimal costs. As an example, the node consistency of a variable i with associated cost function c_i requires that d_i contains at least one value v such that $c_i(v) = 0$ and no value w such that that $c_\emptyset + c_i(w) \geq k$ (the forbidden cost). Equivalently, this means that there is at least one value that does not increase cost locally and no value that is locally infeasible. If a problem does not satisfy a local consistency property, it is possible to transform it in an equivalent problem (defining the same joint cost/energy distribution) satisfying the property thanks to Equivalence Preserving Transformations (EPTs). An EPT is a local transformation of the CFN which can shift cost (or energy) between cost functions of intersecting scopes without changing the global energy distribution. These EPTs are iteratively applied in so-called local consistency enforcing algorithms that iterate EPTs until the CFN satisfies the local consistency property. For example, if a variable violates node consistency then by deleting infeasible values and by shifting costs to $c_\emptyset$, the variable can be made node consistent. Beyond node consistency, many of these local consistency properties and associated polynomial time enforcing algorithms have been defined [32]–[34]. Depending on the locality of the property, which may apply to one variable, one cost function or more, they are called Node, Arc or higher order consistencies. Globally, the essential effect of enforcing local consistencies is that infeasible values, which can only lead to solutions of cost larger than k may be pruned and that the constant cost function $c_\emptyset$, a global lower bound on the optimum, may be increased. The amount of pruning increases as the upper bound k decreases. All this is obtained without changing the global energy distribution.

Compared to local consistency enforcing, DEE, which has also been studied in CFN under the name of substitutability ([22], [35], [36]) does not preserve the global energy distribution as it may remove feasible sub-optimal solutions.

Since local consistency algorithms are polynomial time algorithms, they cannot solve all CFNs. Instead of *A**, an exhaustive Depth First Branch and Bound (DFBB) tree search is used to provably solve the problem. Furthermore, dominance and local

consistency analysis are not performed only at the beginning of the search, as preprocessing, but are also incrementally maintained at each node of the tree search. By increasingly simplifying the problem and strengthening the lower bound $c_\emptyset$, they give information that can be used to prune and heuristically guide the search in a tree defined by branching.

Our recent work [20]–[22] highlighted the power of these CFN DFBB-based method to efficiently solve various CPD problems and hence, spurred new developments to tackle more complex and challenging CPD problems.

Branching schemes

When preprocessing is unable to solve the problem, tree search algorithms such as A^* or DFBB try to simplify the problem by making assumptions on variables. Each additional assumption generates a new node, son of the previous node. In the DFBB search, successive assumptions are made until either all variables become assigned to a single value and a new solution is found or the current lower bound $c_\emptyset \geq k$. In the first case, the upper bound k can be updated to the new solution cost (we are only interested in better solutions), In the last case, we know that with the current assumptions, we cannot reach a cost less than $c_\emptyset$ and therefore less than k. We need to backtrack, that is we reconsider our last assumption and *branch* on a new assumption.

The most obvious type of branching uses a chosen variable i and considers all its current values as possible assumptions. This is called *n*-ary branching. An alternative branching scheme consists in selecting a variable i and a value a and uses as assumptions the fact that the variable i either takes the value a, or not (and the value can be removed from the domain). This is called binary branching. By exploiting results in proof theory, binary branching has been shown to be more powerful than n-ary branching (with a given local consistency being maintained at each node, it may explore an exponentially smaller number of nodes than the previous one, and the converse is impossible [37]. *n*-ary branching can however be polynomially better on some problems). An even more general branching method is dichotomic branching where the domain of a chosen variable i is split in two chosen sets. The split can for example be done in the middle of the domain range or in the middle of the unary energy range (similarly to [38]) where low energy rotamers are part of the first set and high energy rotamers forms the second set.

Variable and value ordering heuristics

Besides a branching strategy, DFBB also needs to dynamically choose the next assumption to make. Variable and value ordering heuristics are used to choose respectively the next variable to branch on and the next value to consider for this chosen variable.

Variable ordering heuristics may have a tremendous effect on the efficiency of the search algorithm. They are all based on the 'fail-first' principle [39] : 'To succeed, try first where you are most likely to fail'. Several measures have been used to try to evaluate the likelihood of failing by fixing a variable. One simple measure is the current size of the domain (*dom*) [39]. Under this measure, the variable that has the smallest domain should be assigned next. To take into account the number of cost functions that involve a variable (the so-called degree of the variable), more sophisticated heuristics select the variable that has the minimum ratio of the domain size over the current degree (*dom/ddeg* [40]), over the degree weighted by the number of failures observed in the past for each cost function (*dom/wdeg* [20], [41]) or by the sum of the median of cost functions (*dom/cmed* [21], [22]). Additionally, the last conflict heuristics [42] simply tries to select the last variable that led to inconsistency during search (if any).

Once a variable is chosen, binary and *n*-ary schemes need to choose the next value to consider. The effect of value ordering heuristics is often less dramatic than for variables. However, a good value ordering heuristics may help to quickly find a good solution. Because we are then interested only in strictly better solutions, we may set k to the cost of this new solution. This improves pruning. The most usual value ordering heuristics in CFNs is to choose first a value a that has a unary cost $c_i(a) = 0$. Such a value always exists thanks to Node Consistency enforcing [43].

Limited Discrepancy Search

Good value ordering heuristics may allow the search to quickly find a good solution, but they may also fail. This can be because the very first assumption made may not be reconsidered before a complete subtree is explored. Limited Discrepancy Search (LDS) [44] tries to overcome this situation by only exploring paths that have a bounded number of variable assignments that differ from those defined by the value heuristics (called discrepancies). LDS can often find good solutions much faster than DFBB search. The best solution found can be used to set the initial upper bound k. Obviously, the power of LDS strongly depends on the quality of the heuristic used for variable and value ordering.

Contributions of the current work

The work reported herein makes most of the CFN algorithms and strategies accessible to the CPD community by implementing them in the open source CPD framework *Osprey* 2.0 [23]. It also provides novel CFN-based methodological advances targeted at

Journal of COMPUTATIONAL CHEMISTRY WWW.C-CHEM.ORG FULL PAPER

highly complex CPD problems. A first contribution of this work is to make all the technology described above available in *Osprey*. We focus in the next section on new methodological developments that refine and extend these for CPD.

Since the initial value of the upper bound k has a strong influence on pruning and therefore efficiency, we developed two new amino-acid based upper bound heuristic to tighten the initial upper bound. These heuristics were also used to build new branching strategies based on amino-acid types.

A^* search has several advantages over DFBB search. It is known that it always develops fewer nodes than DFBB if the same lower bound (or admissible heuristics in A^* terminology) is used. It is also able to directly produce a sorted list of solutions of increasing energies. In order to assess the efficiency of local consistency enforcing and associated lower bound in this context, two A^*variants were developed. The first one uses the CFN lower bound as its admissible heuristics instead of the usual CPD lower bound [45]. It will be denoted as "A^*-CFN". The second one uses a strengthened CFN lower bound, obtained by a bounded DFBB exploration of the current node (using the minimum lower bound found at the leaves of this depth limited search as the strengthened lower bound). This method is referred to as "A^*-CFN-BDF". The performances of both search methods were compared to those of the A^* search commonly used in CPD, denoted as "A^*-vanilla" in the rest of the paper.

Methods

Best-First Search with a CFN lower bound

The completeness of the A^* search method used in the DEE/A^* approach [45] relies on the use of a so-called "admissible heuristic function". This function must provide an optimistic estimate of the total energy of all conformations below the current node, i.e. must be a lower bound on the optimum of the current problem. Thus, the lower bound defined by $c_\emptyset$ and local consistency enforcing can be directly plugged in the A^* algorithm and replace the existing heuristics.

The lower bound used in current DEE/A^* implementations is equivalent to CFN 'Directed Arc Consistency Counts' [46]. This lower bound has been obsoleted by several Arc consistency properties [47]. Hence, one might expect that replacing A* lower bound with stronger local consistency based bounds, might improve the efficient of A*.

A^* search maintains a priority queue of open nodes for expansion. CFN processing by local consistency is performed on each node, pruning the domains of every associated sub-problem and offering the lower bound $c_\emptyset$ as a priority. When a node is considered for the expansion, any branching scheme mentioned above can be used and local consistency is applied on each generated independent sub-problem. Those that have a lower bound less than the current upper bound k are inserted in the queue. This defines the A^*-CFN algorithm. Hence, A*-CFN uses local consistencies both to compute the lower bound and as a pruning device during search. This contrasts with DEE/A* where DEE is only used as a preprocessing pruning technique. Hence, A*-CFN has a permanent opportunity to reduce its search space and uses a stronger lower bound than the traditional A* lower bound [47].

One of the major weaknesses of A^* is its worst-case exponential space complexity associated to this queue. By strengthening the lower bound, we can prune more node and reduce space complexity. With this aim, a Bounded Depth First branch and bound search was implemented leading to the A^*-CFN-BDF algorithm. If the lower bound of a node exceeds k, it is not inserted in the queue. The depth limit used is an adjustable parameter.

Side chain positioning-based upper bounding

To quickly identify a good upper bound, we extract a Side Chain Positioning (SCP) problem from the initial CPD problem by restricting the domain of every mutable position to rotamers corresponding to a unique amino-acid. Clearly, the cost of any solution of this restricted SCP problem defines an upper bound of the original CPD problem. Two heuristics to select the amino-acid type (AA) at position i were considered. The first one relies on the usual CFN value ordering heuristics and selects the amino-acid of a rotamer v with a cost $c_i(v) = 0$ (*Zero-Cost−AA*). The second one selects the wild-type amino-acid if it is still available after local consistency enforcing and otherwise, uses the *Zero-Cost−AA* choice (*Wt−Zero-Cost−AA*).

After solving the SCP problem and updating the upper bound k, local consistency enforcing is maintained on the initial CPD problem. All this is done only if at least one residue has more than one amino-acid type in its domain following the initial local consistency enforcing.

Side Chain Positioning-based branching schemes

The two previous amino-acid selection heuristics can also be used to define novel dichotomic branching schemes. Dynamically during search, for a chosen variable i, an amino-acid aa_i is selected according to the heuristics. In a first branch, the variable i is restricted to the rotamers of type aa_i and in the second all remaining rotamers (type $\neq aa_i$) are kept. This branching

incrementally reduces CPD to SCP: each time a branching is performed, the domain of a variable is restricted to a single amino-acid in the first branch. Once there is no mutable residue left in a given branch, either a binary or n-ary branching scheme is applied to select rotamers. This branching strategy is denoted as SCP-based branching in what follows.

Our CFN-based CPD framework

The overall strategy of the Cost Function-based CPD framework developed in this work is described in Figure 1. Local consistency (with integrated DEE dominance analysis, DEE[1] [22], [36]) is maintained during the whole process. A first upper bound is sought by LDS and optionally, improved by the SCP-based upper bounding. Then, the remaining search space is explored to extract the GMEC and enumerate suboptimal solutions when desired. Two main alternative search strategies can be performed: an A^* (Best-First) search or a DFBB search. Then for either strategy, binary, *n*-ary, dichotomic or SCP-branching may be used. In all the search strategies, a dynamic variable and value ordering can optionally be performed. Finally, when a lower bound strengthened by depth bounded DFBB search is used, we get the A^*-CFN-BDF search strategy.

The complete approach has been implemented in *Osprey* (version 2.0, [23]) which allows for discrete and continuous modeling of the protein conformation at the side-chain and backbone level [48]. All those models, including those capturing backbone flexibility or continuous rotamers are ultimately reduced to in *Osprey* to a CPD pairwise energy matrix, which will also benefit from the CFN-based algorithms. Thus, the present work increases the efficiency of this CPD package without losing provability while extending access to high dimensional flexible protein design problems.

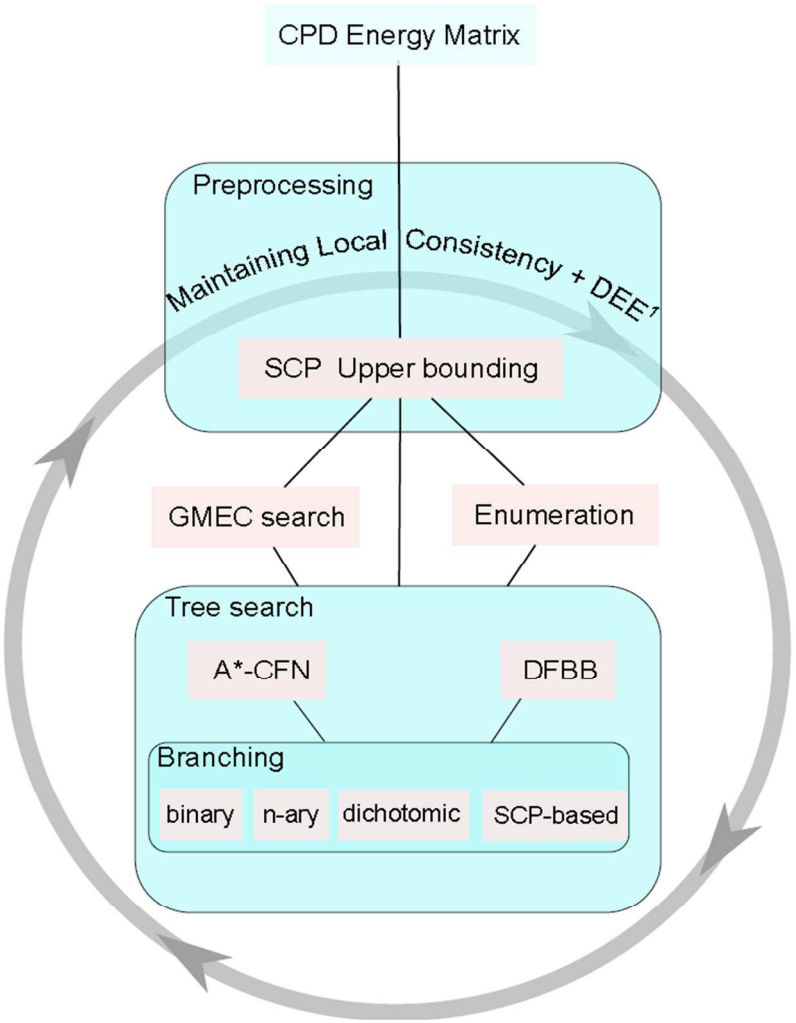

Figure 1 CFN-Based CPD Framework

Journal of COMPUTATIONAL CHEMISTRY WWW.C-CHEM.ORG FULL PAPER

Results and discussion

The benchmark set of 30 CPD instances previously prepared [21], [22] was used here to assess the performances brought by the contributions of the current work. This benchmark set includes a variety of protein structures, alone or in complex with a protein or a ligand, derived from high resolution structures deposited in the Protein Data Bank (PDB). In these selected systems, diverse sizes of sequence-conformation spaces are present, varying by the number of mutable residues, the number of alternative amino-acid types at each position and the number of rotamers considered for each amino-acid.

All computations were performed on one core of an AMD OpteronTM Processor 6176@2.3 GHz. We used 128GB of RAM and a 9,000 sec timeout. Problems not solved within this limit are considered as using 9,000 seconds (a lower bound on the real time they would require). The integrated CFN with incremental DEE (DEE1 option) pruning option previously described was activated in all CFN-based experiments (both during preprocessing and during search) [22]. Unless otherwise specified, the median cost variable ordering heuristic (dom/cmed) was used in all the experiments (with the "last conflict" heuristics [42]) in conjunction with the zero unary cost value ordering heuristic. The statistic metrics reported hereafter ignore the few instances for which the difference in the time needed to solve the GMEC was essentially similar (difference of less than 1sec).

For the GMEC problem, we first compare the A^*-CFN to the A^*-vanilla algorithm. We then compare these A*-based strategies to the DFBB method. Next, the two heuristics for SCP-based upper bound tuning were assessed against each other in order to identify the strongest pruning. Then, within the DFBB search method, we assessed the performance of the new SCP-based branching. Finally, we evaluated the efficiency of the new developments to enumerate an ensemble of sub-optimal solutions and not just the GMEC.

Effects of the CFN lower bound

Performances for solving the GMEC identification problem of the new A^*-CFN method (using the CFN lower bound) were compared to those of the A^*-vanilla algorithm (usually used in CPD after a DEE pre-processing). In order to ensure that the difference of performances between both methods is only due to the quality of the lower bound, a static variable ordering was used. To compare the search methods fairly, we performed a full DEE preprocessing (*Osprey* algOption=3). and a local consistency preprocessing step on each instance and then applied either A^*-vanilla or A^*-CFN. Thence, both strategies start with an identical search space.

A^*-vanilla and A^*-CFN solved, respectively, 22 and 23 cases before timeout (Table 1). A^*-CFN outperforms A^*-vanilla for 15 out of 18 discriminant cases (with more than 1 sec. difference). If we assign 9,000 sec to unsolved cases, A^*-CFN allowed to save 978.9 sec in average (Table 2). Except for 1MJC problem which was directly solved at the end of the preprocessing step, A^*-CFN expanded fewer nodes than A^*-vanilla in all cases solved by both methods. For example, A^*-vanilla expanded 63,974 nodes to solve 1C9O problem while A^*-CFN expanded only 20 (corresponding to 3,199 fold decrease in the number of expanded nodes). As expected, the inexpensive A^*-vanilla lower bound led to a higher number of nodes expanded per minute with A^*-vanilla than with the A^*-CFN method. However, this is not sufficient to compensate for the much higher number of nodes that A^*-vanilla explores.

We then compared the performances of the DFBB implemented in *Osprey* against those of A^*-CFN (Table 2 and Supplementary Table S-1), both using the same lower bounding mechanism. Since it allows for it, value and variable ordering heuristics were activated within DFBB.

Out of the 30 design cases, DFBB solved 24 instances within the timeout (Table 2 and Supplementary Table S-1). DFBB was faster than A^*-CFN in 15 out of 21 discriminant cases, with an average runtime gain of 388.8 sec. It was also faster than A^*-vanilla in 16 out of the 20 discriminant case, with an average runtime gain of 1,289.4 sec. Therefore, DFBB search equipped with incremental local consistency enforcing algorithm remains overall more efficient than A^* search methods.

While A^*, as a Best-first search method, is known to always explore less nodes than Depth-First search when the same tree is explored with the same lower bound, DFBB can quickly improve its upper bound k while A^* cannot. This probably explains the performance gain of the Depth-First search strategy combined with the CFN lower bound. Indeed, in 18 out of 21 cases (excluding 1MJC), DFBB explored fewer nodes than A^* (Table 2). The polynomial space use of DFBB is also likely to help in the context of CPU processors using multiple levels of increasingly slow caches.

Table 1: Solving the GMEC identification problem using *A**-vanilla and *A**-CFN. For each method are reported the number of nodes expanded (nd), the runtime in second (time) and the speed (nd/min).

PDB accession code	Sequence - Conformation Space Size	*A** -vanilla			*A**-CFN		
		nd	time (s)	nd/min	nd	time(s)	nd/min
1MJC	4.36E+26	0	5.1	-	0	4.8	-
1CSP	5.02E+30	127	94.6	80	10	38.7	15
1BK2	1.18E+32	5,234	285.3	1,100	21	25.6	49
1SHG	2.13E+32	558	77.5	432	28	76.8	21
1CSK	4.09E+32	113	7.8	869	16	3.7	259
1SHF	1.05E+34	21	13.9	90	8	14.5	33
1FYN	5.04E+36	58,380	3,355.0	1,044	81	160.7	30
1PIN	5.32E+39	24	1,480.4	0	11	1,515.8	0
1NXB	2.61E+41	45	3.8	710	9	2.0	270
1TEN	6.17E+43	243	37.1	392	9	37.9	14
1POH	8.02E+43	169	29.8	340	14	29.1	28
2DRI	1.16E+47	-	-	-	-	-	-
1FNA	3.02E+47	109,523	638.6	10,290	298	571.8	31
1UBI	2.43E+49	193,077	6,895.0	1,680	829	1,913.0	26
1C9O	3.77E+49	63,974	960.6	3,995	20	245.0	4
1CTF	3.95E+51	22,658	2,363.0	575	417	1,949.0	12
2PCY	2.34E+52	14,929	299.1	2,994	119	50.4	141
1DKT	3.94E+58	202,611	1,078.6	11,270	111	855.9	7
2TRX	9.02E+59	29,845	123.5	14,499	245	63.0	233
1CM1	3.73E+63	-	-	-	-	-	-
1BRS	1.67E+64	-	-	-	-	-	-
1CDL	5.68E+65	-	-	-	-	-	-
1LZ1	1.04E+72	-	-	-	4,899	1,481.2	198
1GVP	1.51E+78	-	-	-	-	-	-
1RIS	1.23E+80	-	-	-	-	-	-
2RN2	3.68E+80	15,845	1,276.7	744	104	1,146.3	5
1CSE	8.35E+82	26	79.0	19	8	187.6	2
1HNG	3.70E+88	16	1,739.0	0	4	1,901.5	0
3CHY	2.36E+92	-	-	-	-	-	-
1L63	2.17E+94	19,567	1,409.9	832	74	1,358.7	3
# Nb of cases solved in 900 sec			14			16	
# Nb of cases solved in 9000 sec			22			23	

Table 2: Cross comparison of search strategies for solving the GMEC identification problem.

	*A**-vanilla	*A**-CFN
*A**-CFN	**-978.9** **15/18**	
DFBB	**-1289.4** **16/20**	**-388.8** **15/21**

In each cell, the upper number is an average runtime difference of the row algorithm vs. the column one. Hence, a negative value is in favor of the algorithm in the cell's row. In the lower part of the cell is indicated the number of instances solved faster by the algorithm of the cell's row (the denominator is the total number of discriminate instances). Only cases for which the runtime difference is greater than 1 sec. are considered.

Journal of COMPUTATIONAL CHEMISTRY WWW.C-CHEM.ORG FULL PAPER

Side chain positioning-based upper bounding

When tree search is used, any value pruned at the root prunes a potentially exponential number of conformations. By strengthening the initial upper bound k, local consistencies can provide increased pruning which is likely to improve speed.

We evaluated the effect of improved initial upper bounds, as provided by solving a simpler side chain positioning sub-problem [16] and using the optimum found as an upper bound of the initial CPD problem. DFBB with a binary branching scheme is used in these experiments.

We tested our two amino-acid selection heuristics (*Zero-Cost−AA* and *Wt−Zero-Cost−AA*) and for reference, we also performed a run without these options (*None*) (Figure 2).

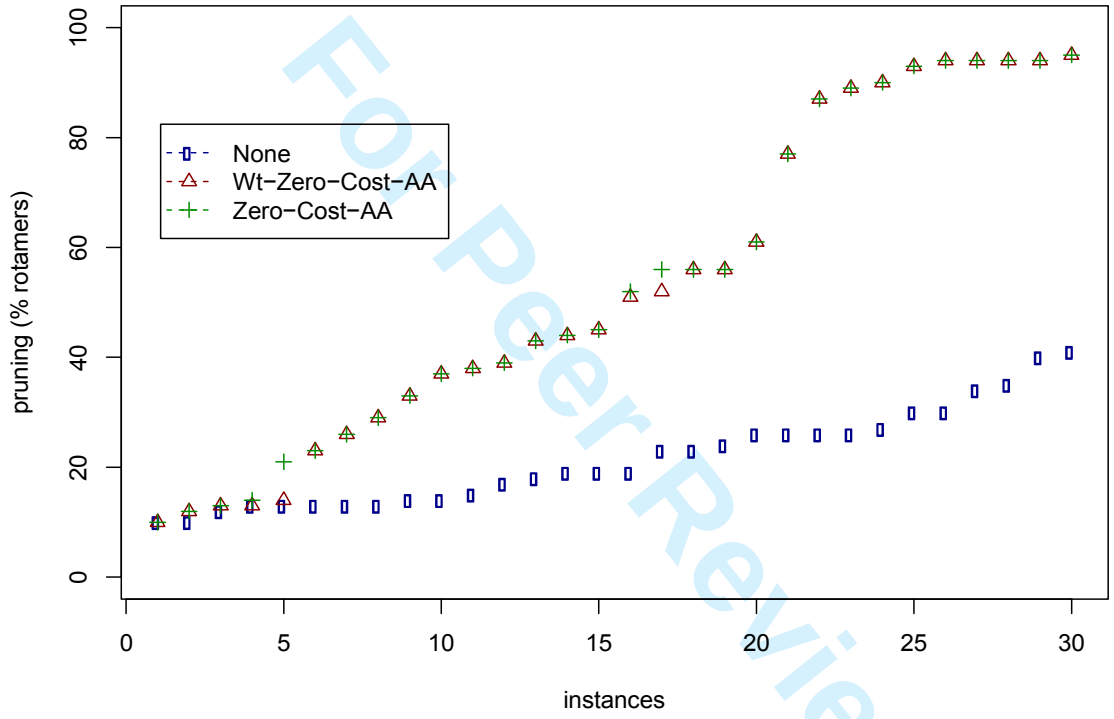

Figure 2 Preprocessing pruning

Without any initial upper bound, the GMEC could never be identified at the root node. Conversely, using the initial SCP-based upper bounding, the GMEC was immediately identified at root node in respectively 22 and 23 cases using respectively the *Zero-Cost−AA* and *Wt−Zero-Cost−AA* heuristics. With either heuristics, up to 95% of values were pruned at the root node, instead of merely 41% without upper bounding. However, the time required for the SCP upper bounding at root degrades the runtime. Indeed, the 25 instances were solved without SCP upper bounding while 24 and 23 cases were solved with *Wt−Zero-Cost−AA* and *Zero-Cost−AA* respectively. Hence, to instead incrementally perform SCP upper bounding during search, we propose a new SCP-branching mechanism in order to translate this pruning power into a search speedup.

Side Chain Positioning-based branching

In order to benefit from the pruning improvement offered by SCP-based upper bounding, we considered the development of a branching scheme that incrementally performs SCP during search. Indeed, for a given variable i, restricting allowed amino acid to a 'well' selected one (aa_i) could lead to a sub-problem with lower optimal cost compared to the opposite problem that excludes rotamers of amino acid type aa_i. Subsequent problems of the first branch become simpler not only because of the domain size reduction but also because they incrementally reduce to a side chain placement problem which has been shown

simpler to solve [16]. SCP-based branching can be defined as specific form of dichotomic branching that splits the domain in such a way that the first splitted domain contains all rotamers of a single amino acid. All remaining rotamers (of all other amino acids) are in the second splitted domain, which develops a second branch. This branching scheme contrasts to the usual dichotomic branching that splits the domain in the middle of unary cost range (the $E(i_r)$ energy terms).

To choose the selected amino-acid, we used the *Zero-Cost−AA* amino-acid selection heuristic already presented. We evaluated binary, *n*-ary, dichotomic and SCP-based branching strategies against each other (Table 3). Supplementary Table S-2 reports the results for all 30 instances.

As expected, binary branching was faster (by an average of 115 sec.) and solved more problems (11 over 19 discriminant problems) than *n*-ary branching. Compared to the (unary cost-based) dichotomic branching, it also showed a favorable average runtime gain (578.7 sec). This does not translate however in a larger number of problems that are solved faster (only 5 cases out of 18). No single usual strategy appears more effective than others. In contrast, the SCP-based branching scheme appears unambiguous better compared to all the other branching schemes. Out of 20 cases and all other contenders, a minimum of 18/20 cases were solved faster by SCP-based branching with a minimum average runtime gain of 426.8 sec per instance. Hence, these results showed that only SCP-based branching outperforms all the other three branching schemes, both in terms of runtime and number of CPD problems solved faster. The poor performance of dichotomic branching is somehow unexpected, since it splits apart rotamers at a given position into two groups depending on their energies.

Table 3: Cross comparison of branching schemes for solving the GMEC identification problem.

	Binary	*n*-ary	Dichotomic
n-ary	115.5 8/19		
Dichotomic	578.7 **13/18**	456.8 **12/18**	
SCP	**-426.8** **19/20**	**-596.1** **18/18**	**-947.6** **18/20**

In each cell, the upper number is an averaged runtime difference between the row and the column strategy. Hence, a negative value is in favor of the branching strategy in the cell's row. In the lower part of the cell is indicated the number of instances solved faster by the method of the cell's row (the denominator is the total number of instances considered).The runtime assigned to unsolved instances is 9,000 sec (which is a runtime lower bound for unsolved cases). Only cases for which the runtime difference is greater than 1 sec. are considered.

Enumeration of sub-optimal solutions

The problem of enumerating sequence-conformation models within a range of 2 $kcal.mol^{-1}$ above the GMEC was considered for all 30 instances. First, we tried the *A**-CFN and DFBB schemes using the binary branching scheme. Secondly, we tried with the SCP-based branching activated. Finally, we also compared the performances of these methods to enumerate ensembles of sub-optimal unique *sequences*. Since CPD seeks a set of sequences, enumeration can be restricted to a single conformation per sequence [21]. The SCP-based branching scheme gives a straightforward solution to that problem because it processes amino acids first. In a GMEC search, the upper bound is updated every time a new solution is found. For enumeration, there is no such upper bound updating. Since SCP-based branching defines the amino acid identity at all positions prior to conformation search, it can instead locally update the upper bound (in the current unique sequence sub-problem) in order to keep only the best conformation for the current sequence. Within a DFBB search strategy, the upper bound is locally updated once there is no mutable residue left in a given SCP branch. Similarly, for *A**-CFN with SCP-branching scheme, each time there is no mutable residue left, we simply use a depth first branch and bound search without any depth limit to get the best solution for the current sequence before adding the node to the expansion queue. This mechanism enables *A**-CFN to automatically enumerate unique sequences in an energy sorted order (the SCP part being a depth first search).

A summarized cross comparison of search strategies is given in Table 3 (all further detailed results are included in Supplementary Table S-3 and S-4). Again, only cases showing a difference in runtime of more than 1 sec. were considered and unsolved cases were assigned a runtime of 9,000 sec.

*A**-CFN is outperformed by DFBB in 13 cases out of 15 (only 8 were actually solved by *A**-CFN while DFBB solved all 15). *A**-CFN-SCP solved a total of 14 cases and outperformed *A**-CFN in 13 out of 14 cases. However, DFBB turned out to be

Journal of COMPUTATIONAL CHEMISTRY WWW.C-CHEM.ORG FULL PAPER

slightly more efficient (an average runtime gain of 205.1 sec in favor of DFBB and 9 out of 17 cases solved faster). Ultimately, DFBB-SCP outperforms A^*-CFN, DFBB, and A^*-CFN-SCP. Hence, SCP improves both A^*-CFN and DFBB search strategies

Ensembles of unique *sequences* were generated for 21 cases by both A^*-CFN-SCP and DFBB-SCP. Because the generation of unique sequences is inherently encoded by the SCP branching scheme, and thanks to upper bounding; SCP-branching translates into an important improvement of the performances, both in terms of runtime and number of problems solved. A significant gain in runtime is always obtained when unique sequence SCP is activated (more than 2,000 sec in average and up to 4675.2 sec).

Table 4: Cross comparison of search methods for the generation of sub-optimal sequence-conformation ensembles as well as enumeration of sub-optimal unique sequences.

	A^*-CFN	DFBB	A^*-CFN-SCP	DFBB-SCP	A^*-CFN-SCP-unique
DFBB	**-2239.7** **13/15**				
A^*-CFN-SCP	**-2150.7** **13/14**	205.1 8/17			
DFBB-SCP	**-2934.7** **14/15**	**-613.2** **10/17**	**-927.4** **9/15**		
A^*-CFN-SCP-unique	**-4675.2** **20/21**	**-3075.4** **18/21**	**-3241.4** **19/21**	**-2579.0** **17/21**	
DFBB-SCP-unique	**-4125.7** **20/21**	**-2525.9** **17/21**	**-2691.9** **18/21**	**-2131.0** **16/20**	549.5 9/21

In each cell, the upper number is an averaged runtime difference between the row and the column strategy. Hence, a negative value is in favor of the search method in the cell's row. In the lower part of the cell is indicated the number of instances solved faster by the method of the cell's row (the denominator is the total number of instances considered). The runtime assigned to unsolved instances is 9,000 sec (which is a runtime lower bound for unsolved cases). Only cases for which the runtime difference is greater than 1 sec. are considered.

Ultimately, enumeration of unique sequences provided by SCP-based branching scheme also offers the chance of enumerating sequences over a larger energy interval above the GMEC. Indeed, taking as an illustration the 1MJC instance and an interval of 2 $kcal.mol^{-1}$, the A^*-CFN required 232.6 sec to enumerate 2,110,737 sequence-conformations (4,408,541 nodes). DFBB, A^*-CFN-SCP and DFBB-SCP required, respectively, 140.7 sec (4,537,935 nodes), 143.7 sec (4,415,342 nodes) and 178.7 sec (4,555,451 nodes) to handle the same instance. This ensemble corresponds to only 91 unique sequences. For a larger energy interval of 20 $kcal.mol^{-1}$, using DFBB-SCP method restricted to the enumeration of unique sequences, 1,034 sequences were generated in 295.85 sec. (with a number of nodes lower than 50,489). This energy interval is out of reach when all conformations have to be enumerated. SCP-branching is thus an effective solution to handle CPD problems and gives access to the enumeration of unique sequences within larger energy window than what can be achieved using standard branching schemes.

Conclusion

This work reports the introduction of Cost Function Network optimization techniques to solve Computational Protein Design problems. Compared to prior work [20]–[22], it introduces novel search strategies in order to speedup search methods, which were here implemented in the CPD-dedicated software *Osprey*. The presented search algorithms have shown to be more efficient than optimization methods based on the DEE/A^*framework for both GMEC search and suboptimal solutions enumeration (which is often performed in order to account for inaccuracies and approximations made in the modeling of the design problem). In addition, the new SCP-based branching scheme enables to generate unique sequences within a given threshold. The implementation in *Osprey* of existing CFN algorithm together with these new methods enables novel opportunities. Indeed, these algorithms can be used in combination with all the other functionalities in this software, in particular for molecular flexibility modeling. Any macromolecular flexibility method that can be represented as an optimization problem with a single matrix and an upper bound can be performed with the presented CFN-based methods. This specifically includes the recently introduced provable DEEPer modeling [24], [48] that handles designs with discrete and continuous flexibility of side chains and backbone.

Acknowledgements

This work has been funded by the "Agence Nationale de la Recherche", references ANR 10-BLA-0214 and ANR-12-MONU-0015-03. 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. S. Traoré was supported by a grant from the INRA and the Region Midi-Pyrénées. K.E. Roberts and B.R. Donald supported by NIH Grant 2-R01-GM-78031-5. We thank Pablo Gainza for discussion on search algorithms in *Osprey*.

References

[1] C.-Y. Chen, I. Georgiev, A. C. Anderson, B. R. Donald, *Proc. Natl. Acad. Sci. U. S. A.*, **2009**, DOI:10.1073/pnas.0900266106.
[2] K. M. Frey, I. Georgiev, B. R. Donald, A. C. Anderson, *Proc. Natl. Acad. Sci. U. S. A.*, **2010**, DOI:10.1073/pnas.1002162107.
[3] N. P. King, J. B. Bale, W. Sheffler, D. E. McNamara, S. Gonen, T. Gonen, T. O. Yeates, D. Baker, *Nature*, **2014**, DOI:10.1038/nature13404.
[4] A. E. Miklos, C. Kluwe, B. S. Der, S. Pai, A. Sircar, R. A. Hughes, M. Berrondo, J. Xu, V. Codrea, P. E. Buckley, A. M. Calm, H. S. Welsh, C. R. Warner, M. A. Zacharko, J. P. Carney, J. J. Gray, G. Georgiou, B. Kuhlman, A. D. Ellington, *Chem. Biol.*, **2012**, DOI:10.1016/j.chembiol.2012.01.018.
[5] J. Karanicolas, B. Kuhlman, *Curr. Opin. Struct. Biol.*, **2009**, DOI:10.1016/j.sbi.2009.07.005.
[6] J. B. Siegel, A. Zanghellini, H. M. Lovick, G. Kiss, A. R. Lambert, J. L. St Clair, J. L. Gallaher, D. Hilvert, M. H. Gelb, B. L. Stoddard, K. N. Houk, F. E. Michael, D. Baker, *Science*, **2010**, DOI:10.1126/science.1190239.
[7] J. M. Palomo, M. Filice, *Biotechnol. Adv.*, **2015**, DOI:10.1016/j.biotechadv.2014.12.010.
[8] N. A. Pierce, E. Winfree, *Protein Eng.*, **2002**, *15*, 779–782.
[9] C. A. Voigt, D. B. Gordon, S. L. Mayo, *J. Mol. Biol.*, **2000**, DOI:10.1006/jmbi.2000.3758.
[10] B. Kuhlman, D. Baker, *Proc. Natl. Acad. Sci. U. S. A.*, **2000**, *97*, 10383–10388.
[11] L. Wernisch, S. Hery, S. J. Wodak, *J. Mol. Biol.*, **2000**, DOI:10.1006/jmbi.2000.3984.
[12] J. Desmet, M. De Maeyer, B. Hazes, I. Lasters, *Nature*, **1992**, *356*, 539–542.
[13] D. Gordon, S. Mayo, *Structure*, **1999**.
[14] E. Althaus, O. Kohlbacher, H.-P. Lenhof, P. Müller, *J. Comput. Biol.*, **2002**, DOI:10.1089/106652702760277336.
[15] A. Leaver-Fay, B. Kuhlman, J. Snoeyink, *Pac. Symp. Biocomput.*, **2005**, DOI:10.1142/9789812702456_0003.
[16] C. L. Kingsford, B. Chazelle, M. Singh, *Bioinformatics*, **2005**, DOI:10.1093/bioinformatics/bti144.
[17] R. F. Goldstein, *Biophys. J.*, **1994**, DOI:10.1016/S0006-3495(94)80923-3.
[18] N. A. Pierce, J. A. Spriet, J. Desmet, S. L. Mayo, *J. Comput. Chem.*, **2000**, *21*, 999–1009.
[19] D. B. Gordon, G. K. Hom, S. L. Mayo, N. a Pierce, *J. Comput. Chem.*, **2003**, DOI:10.1002/jcc.10121.
[20] D. Allouche, S. Traoré, I. André, S. de Givry, G. Katsirelos, S. Barbe, T. Schiex, *Proc. of CP-12*, **2012**.
[21] S. Traoré, D. Allouche, I. André, S. de Givry, G. Katsirelos, T. Schiex, S. Barbe, *Bioinformatics*, **2013**, DOI:10.1093/bioinformatics/btt374.
[22] D. Allouche, I. André, S. Barbe, J. Davies, S. de Givry, G. Katsirelos, B. O'Sullivan, S. Prestwich, T. Schiex, S. Traoré, *Artif. Intell.*, **2014**, DOI:10.1016/j.artint.2014.03.005.
[23] P. Gainza, K. E. Roberts, I. Georgiev, R. H. Lilien, D. A. Keedy, C.-Y. Y. Chen, F. Reza, A. C. Anderson, D. C. Richardson, J. S. Richardson, B. R. Donald, *Methods Enzym.*, **2012**, DOI:10.1016/B978-0-12-394292-0.00005-9.
[24] P. Gainza, K. E. Roberts, B. R. Donald, *PLoS Comput. Biol.*, **2012**, DOI:10.1371/journal.pcbi.1002335.
[25] B. R. Donald, *MIT Press*, **2011**.
[26] R. H. Lilien, B. W. Stevens, A. C. Anderson, B. R. Donald, *J. Comput. Biol.*, **2005**, DOI:10.1089/cmb.2005.12.740.
[27] B. W. B. Stevens, R. H. R. Lilien, I. Georgiev, B. R. Donald, A. C. Anderson, *Biochemistry*, **2006**, DOI:10.1021/bi061788m.
[28] M. J. Gorczynski, J. Grembecka, Y. Zhou, Y. Kong, L. Roudaia, M. G. Douvas, M. Newman, I. Bielnicka, G. Baber, T. Corpora, J. Shi, M. Sridharan, R. Lilien, B. R. Donald, N. A. Speck, M. L. Brown, J. H. Bushweller, *Chem. Biol.*, **2007**, DOI:10.1016/j.chembiol.2007.09.006.
[29] K. E. Roberts, B. R. Donald, P. Boisguerin, P. R. Cushing, D. R. Madden, P. Boisguerin, D. R. Madden, B. R. Donald, *PLoS Comput. Biol.*, **2012**, DOI:10.1371/journal.pcbi.1002477.
[30] I. Georgiev, P. Acharya, S. Schmidt, Y. Li, D. Wycuff, G. Ofek, N. Doria-Rose, T. Luongo, Y. Yang, T. Zhou, B. Donald, J. Mascola, P. Kwong, *Retrovirology*, **2012**, DOI:10.1186/1742-4690-9-S2-P50.
[31] T. Schiex, H. Fargier, G. Verfaillie, *Int. Jt. Conf. Artif. Intell.*, **1995**, *14*, 631–639.
[32] M. Cooper, T. Schiex, *Artif. Intell.*, **2004**, DOI:10.1016/j.artint.2003.09.002.
[33] J. Larrosa, T. Schiex, *Artif. Intell.*, **2004**, DOI:10.1016/j.artint.2004.05.004.
[34] M. Cooper, S. de Givry, T. Schiex, *8th Int. CP-06 Work. Prefer. Soft Constraints*, **2006**, 14p.
[35] C. Lecoutre, O. Roussel, D. E. Dehani, *Princ. Pract. Constraint Program.*, **2012**, 406–421.
[36] S. de Givry, S. Prestwich, B. O'Sullivan, *Princ. Pract. Constraint Program. 2013*, **2013**.
[37] D. Mitchell, *Princ. Pract. Constraint Program. 2003*, **2003**.
[38] E. Hong, S. M. Lippow, B. Tidor, T. Lozano-pérez, **2009**, DOI:10.1002/jcc.
[39] R. M. Haralick, G. L. Elliot, *Artif. Intell.*, **1980**, *14*, 263–313.
[40] C. Bessière, J.-C. Régin, *Proc. Second Int. Conf. Princ. Pract. Constraint Program.*, **1996**, 61–75.
[41] C. Lecoutre, L. Sais, S. Tabary, V. Vidal, *ECAI 2006 17th Eur. Conf. Artif. Intell. August 29-September 1, 2006, Riva del Garda, Italy Incl. Prestig. Appl. Intell. Syst. (PAIS 2006) Proc.*, **2006**, 133.
[42] C. Lecoutre, L. Sa"is, S. Tabary, V. Vidal, *Artif. Intell.*, **2009**, *173*, 1592,1614.
[43] J. Larrosa, T. Schiex, *Int. Jt. Conf. Artif. Intell.*, **2003**, *18*, 239–244.
[44] W. D. Harvey, M. L. Ginsberg, *14th Int. Jt. Conf. Artif. Intell. IJCAI95*, **1995**, *1*, 607–613.
[45] A. R. Leach, A. P. Lemon, *Proteins*, **1998**, *33*, 227–239.
[46] R. Wallace, Directed Arc Consistency Preprocessing, *Selected papers from the ECAI-94 Workshop on Constraint Processing*. Springer, Berlin, 121–137, 1995.
[47] S. De Givry, M. Zytnicki, F. Heras, J. Larrosa, *IJCAI*, **2005**, *19*, 84.
[48] M. a Hallen, D. A. Keedy, B. R. Donald, *Proteins*, **2013**, DOI:10.1002/prot.24150.

Fast search algorithms for Computational Protein Design

Seydou Traoré[1-3], Kyle E. Roberts[4], David Allouche[5], Bruce R. Donald[4], Isabelle André[1-3], Thomas Schiex[5], Sophie Barbe[1-3*]

[i] Université de Toulouse; INSA,UPS,INP; LISBP, 135 Avenue de Rangueil, F-31077 Toulouse, France
[2] INRA, UMR792, Ingénierie des Systèmes Biologiques et des Procédés, F-31400 Toulouse, France
[3] CNRS, UMR5504, F-31400 Toulouse, France
[4] Department of Biochemistry, Department of Computer Science, Duke University, Durham, NC, USA
[5] Unité de Biométrie et Intelligence Artificielle, UR 875, INRA, F-31320 Castanet Tolosan, France

* Correspondence to: Sophie Barbe, Laboratoire d'Ingénierie des Systèmes Biologiques et des Procédés – INSA; CNRS UMR5504; UMR INRA 792; 135, Avenue de Rangueil; F-31077 Toulouse cedex 4, France. Tel: +33 561 559 963; Fax: +33 561 559 400; E-mail: sophie.barbe@insa-toulouse.fr

Table S-1 Comparison of search strategies for solving the GMEC identification problem. For each method are reported the number of nodes expanded and the number of backtracks (nd (bt)), the CPU-time in second (time) and the speed (nd/min).

PDB	Sequence Conformation Space Size	*A**-vanilla			*A**-CFN			DFBB		
		nd	time (s)	nd/min	nd	time (s)	nd/min	nd(bt)	time (s)	nd/min
1MJC	4.36E+26	0	5.1	-	0	4.8	-	0	6.0	-
1CSP	5.02E+30	127	94.6	80	10	38.7	15	82(56)	97.0	50
1BK2	1.18E+32	5,234	285.3	1,100	21	25.6	49	144(90)	9.1	949
1SHG	2.13E+32	558	77.5	432	28	76.8	21	114(75)	29.5	231
1CSK	4.09E+32	113	7.8	869	16	3.7	259	38(15)	3.6	633
1SHF	1.05E+34	21	13.9	90	8	14.5	33	32(13)	14.0	137
1FYN	5.04E+36	58,380	3,355.0	1,044	81	160.7	30	575(425)	159.4	216
1PIN	5.32E+39	24	1,480.4	0	11	1,515.8	0	34(13)	1,465.5	1
1NXB	2.61E+41	45	3.8	710	9	2.0	270	18(0)	3.6	300
1TEN	6.17E+43	243	37.1	392	9	37.9	14	45(18)	37.8	71
1POH	8.02E+43	169	29.8	340	14	29.1	28	82(32)	11.4	431
2DRI	1.16E+47	-	-	-	-	-	-	-	-	-
1FNA	3.02E+47	109,523	638.6	10,290	298	571.8	31	402(287)	61.7	390
1UBI	2.43E+49	193,077	6,895.0	1,680	829	1,913.0	26	764(659)	1,297.9	35
1C9O	3.77E+49	63,974	960.6	3,995	20	245.0	4	275(221)	242.1	68
1CTF	3.95E+51	22,658	2,363.0	575	417	1,949.0	12	708(567)	1,815.1	23
2PCY	2.34E+52	14,929	299.1	2,994	119	50.4	141	114(67)	69.6	98
1DKT	3.94E+58	202,611	1,078.6	11,270	111	855.9	7	174(74)	324.7	32
2TRX	9.02E+59	29,845	123.5	14,499	245	63.0	233	178(83)	128.9	82
1CM1	3.73E+63	-	-	-	-	-	-	-	-	-
1BRS	1.67E+64	-	-	-	-	-	-	-	-	-
1CDL	5.68E+65	-	-	-	-	-	-	-	-	-
1LZ1	1.04E+72	-	-	-	4,899	1,481.2	198	569(376)	494.6	69
1GVP	1.51E+78	-	-	-	-	-	-	-	-	-
1RIS	1.23E+80	-	-	-	-	-	-	-	-	-
2RN2	3.68E+80	15,845	1,276.7	744	104	1,146.3	5	364(258)	1,139.9	19
1CSE	8.35E+82	26	79.0	19	8	187.6	2	22(0)	185.9	7
1HNG	3.70E+88	16	1,739.0	0	4	1,901.5	0	37(14)	1,724.1	1
3CHY	2.36E+92	-	-	-	-	-	-	18,739(16,110)	3,685.2	305
1L63	2.17E+94	19,567	1,409.9	832	74	1,358.7	3	231(138)	1,460.8	9

Table S-2 Assessment of branching schemes for solving the GMEC identification problem. For each branching scheme are reported the number of nodes expanded and the number of backtracks (nd (bt)) and the CPU-time in second (time).

PDB	Sequence Conformation Space Size	Binary		N-ary		Dichotomic		SCP	
		nd(bt)	time (s)	nd(bt)	time (s)	nd(bt)	time (s)	nd(bt)	time (s)
1MJC	4.36E+26	21(0)	0.7	21(0)	0.8	21(0)	0.8	21(0)	1.2
1CSP	5.02E+30	215(96)	36.1	5,216(5,105)	81.8	269(123)	26.2	67(22)	14.3
1BK2	1.18E+32	92(35)	8.9	234(183)	8.6	62(20)	8.1	68(23)	8.6
1SHG	2.13E+32	199(87)	7.0	344(245)	11.7	171(73)	6.1	123(49)	3.4
1CSK	4.09E+32	27(1)	1.6	27(1)	1.5	27(1)	2.3	27(1)	2.5
1SHF	1.05E+34	45(9)	1.5	50(14)	1.5	29(1)	2.2	29(1)	2.2
1FYN	5.04E+36	187(83)	212.4	1,916(1,839)	384.2	209(94)	303.5	75(27)	88.6
1PIN	5.32E+39	376(175)	438.8	5,908(5,719)	318.6	0(0)	-	250(112)	141.2
1NXB	2.61E+41	35(1)	1.4	35(1)	1.4	35(1)	1.4	35(1)	1.4
1TEN	6.17E+43	117(41)	4.1	151(81)	2.6	77(21)	8.6	73(19)	2.2
1POH	8.02E+43	69(14)	4.9	94(39)	4.7	61(10)	4.6	57(8)	4.8
2DRI	1.16E+47	4,734(2,350)	6,825.2	0(0)	-	0(0)	-	4,448(2,207)	4,426.2
1FNA	3.02E+47	131(48)	27.6	209(140)	23.9	109(37)	15.2	165(65)	19.6
1UBI	2.43E+49	1,183(572)	309.1	2,901(2,620)	475.5	1,041(501)	198.4	807(384)	192.4
1C9O	3.77E+49	201(82)	131.1	1,627(1,518)	240.1	141(52)	77.5	95(29)	50.2
1CTF	3.95E+51	264(114)	103.3	782(665)	137.8	226(95)	96.4	332(148)	116.2
2PCY	2.34E+52	115(37)	9.9	291(213)	6.1	91(25)	8.1	107(33)	5.8
1DKT	3.94E+58	236(95)	172.4	1,192(1,075)	191.9	186(70)	297.1	134(44)	109.5
2TRX	9.02E+59	204(72)	58.0	567(448)	34.2	190(65)	47.5	174(57)	24.8
1CM1	3.73E+63	537(248)	6,097.2	1,657(1,407)	5,412.0	501(230)	6,070.4	507(233)	5,218.5
1BRS	1.67E+64	0(0)	-	0(0)	-	0(0)	-	0(0)	-
1CDL	5.68E+65	0(0)	-	0(0)	-	0(0)	-	0(0)	-
1LZ1	1.04E+72	1,367(657)	210.9	3,305(2,656)	413.0	1,237(592)	116.9	415(181)	51.3
1GVP	1.51E+78	0(0)	-	0(0)	-	0(0)	-	0(0)	-
1RIS	1.23E+80	0(0)	-	0(0)	-	0(0)	-	0(0)	-
2RN2	3.68E+80	372(155)	54.5	711(530)	46.8	290(114)	38.4	280(109)	26.0
1CSE	8.35E+82	206(62)	10.3	480(337)	17.7	100(9)	9.1	100(9)	4.7
1HNG	3.70E+88	612(266)	287.3	3,476(3,115)	398.3	454(187)	154.5	164(42)	78.5
3CHY	2.36E+92	0(0)	-	0(0)	-	0(0)	-	11,565(5,746)	4,974.4
1L63	2.17E+94	229(75)	187.3	914(766)	179.9	221(71)	124.7	263(92)	98.7

Table S-3 The sequence conformation enumeration problem. For each method are reported the number of nodes expanded (nd) and the CPU-time in second (time).

PDB	Sequence Conformation Space Size	*A**-CFN		DFBB		*A**-CFN-SCP		DFBB-SCP	
		nd	time (s)	nd	time (s)	nd	time (s)	nd	time (s)
1MJC	4.36E+26	4,408,541	232.6	4,537,935	140.7	4,415,342	143.7	4,555,451	178.7
1CSP	5.02E+30	244,640	1,474.7	252,247	883.1	252,778	397.4	269,065	115.3
1BK2	1.18E+32	657,753	153.3	666,956	68.0	690,3	425.8	741,604	677.1
1SHG	2.13E+32	1,326,902	418.8	1,410,363	383.6	1,330,026	78.8	1,400,933	85.9
1CSK	4.09E+32	922,038	262.3	948,083	308.1	928,305	44.0	944,657	32.2
1SHF	1.05E+34	66,848	191.3	71,275	6.3	64,703	29.0	68,171	42.4
1FYN	5.04E+36	-		842,041	661.9	941,879	2,418.9	1,042,143	2,406.8
1PIN	5.32E+39	-		-		36,384	7,431.1	39,228	2,354.6
1NXB	2.61E+41	-		97,333,939	5,949.1	-		97,055,693	5,916.9
1TEN	6.17E+43	14,222,924	2,191.7	14,407,481	950.2	13,902,152	1,665.0	14,558,687	2,129.0
1POH	8.02E+43	-		-		-		-	
2DRI	1.16E+47	-		-		-		-	
1FNA	3.02E+47	-		208,370,115	5,621.2	-		-	
1UBI	2.43E+49	-		744,735	8,283.8	-		-	
1C9O	3.77E+49	-		8,030,665	2,157.5	7,293,462	2,448.3	8,153,909	574.0
1CTF	3.95E+51	-		40,391,526	4,202.1	40,064,876	3,428.2	41,465,042	2,501.9
2PCY	2.34E+52	479,406	181.5	481,801	189.9	471,664	106.7	478,295	33.5
1DKT	3.94E+58	-		-		-		-	
2TRX	9.02E+59	-		-		-		-	
1CM1	3.73E+63	-		-		-		-	
1BRS	1.67E+64	-		-		-		-	
1CDL	5.68E+65	-		-		-		-	
1LZ1	1.04E+72	-		21,137,757	4,705.1	20,240,02	2,809.5	21,310,987	4,247.3
1GVP	1.51E+78	-		-		-		-	
1RIS	1.23E+80	-		-		-		-	
2RN2	3.68E+80	-		-		56,012,326	7,570.1	60,037,744	2,790.0
1CSE	8.35E+82	-		-		-		-	
1HNG	3.70E+88	-		-		-		-	
3CHY	2.36E+92	-		-		-		-	
1L63	2.17E+94	-		-		-		-	

Table S-4 The enumeration of unique sequences. For each method are reported the number of nodes expanded (nd) and the CPU-time in second (time).

PDB	Sequence Conformation Space Size	*A**-CFN-SCP-unique		DFBB-SCP-unique	
		nd	time (s)	nd	time (s)
1MJC	4.36E+26	2,187	37.4	181	5.6
1CSP	5.02E+30	9,875	152.7	1,734	74.5
1BK2	1.18E+32	60,462	890.8	23,801	436.6
1SHG	2.13E+32	10,567	26.1	1,357	28.0
1CSK	4.09E+32	3,645	5.5	415	5.0
1SHF	1.05E+34	605	179.5	85	43.3
1FYN	5.04E+36	85,145	846.5	14,530	2,241.2
1PIN	5.32E+39	4,328	2,283.4	2,199	6,229.5
1NXB	2.61E+41	13,477	459.1	1,088	24.1
1TEN	6.17E+43	7,337	136.2	751	113.5
1POH	8.02E+43	7,129	137.7	364	93.9
2DRI	1.16E+47	-		-	
1FNA	3.02E+47	97,585	807.6	8,261	908.0
1UBI	2.43E+49	26,179	7,767.5	20,833	8,627.1
1C9O	3.77E+49	34,781	2,100.6	3,419	1,196.7
1CTF	3.95E+51	463,902	2,243.2	67,465	3,270.8
2PCY	2.34E+52	2,677	28.2	339	42.9
1DKT	3.94E+58	-		75,586	5,080.7
2TRX	9.02E+59	8,536	101.1	728	270.0
1CM1	3.73E+63	-		-	
1BRS	1.67E+64	-		-	
1CDL	5.68E+65	-		-	
1LZ1	1.04E+72	21,311	1,574.1	5,450	3,653.2
1GVP	1.51E+78	-		-	
1RIS	1.23E+80	-		-	
2RN2	3.68E+80	108,496	1,075.5	13,348	1,691.8
1CSE	8.35E+82	3,356	67.7	1,707	1,429.3
1HNG	3.70E+88	47,438	3,006.0	-	
3CHY	2.36E+92	-		-	
1L63	2.17E+94	-		-	