LUTE (Local Unpruned Tuple Expansion): Accurate continuously flexible protein design with general energy functions and rigid-rotamer-like efficiency

Supplementary Information

Mark A. Hallen[1], Jonathan D. Jou[1], and Bruce R. Donald[1,2,3,*]

Departments of [1] Computer Science and [2] Chemistry, Duke University, Durham, NC 27708 [3] Department of Biochemistry, Duke University Medical Center, Durham, NC 27710

*Corresponding author, brd+jcb16@cs.duke.edu

The following is supplementary information (SI) that provides additional information to substantiate the claims of the paper. It is available online at

http://www.cs.duke.edu/donaldlab/Supplementary/jcb16/lute/

Section A describes the least-squares method used to calculate the LUTE matrix. Section B describes what terms need to be included in the energy matrix. Each term corresponds to a tuple of RCs at different positions. Section C describes how a *pruning interval*, a parameter used for pruning in continuously flexible designs, can be computed for several types of LUTE calculations. Section D provides further details of our implementation of LUTE combined with the iMinDEE [Gainza *et al.*, 2012] algorithm for sequence and conformational search. Section E analyzes the complexity of LUTE combined with the BWM* algorithm for sequence and conformational search. It includes a proof of Theorem 3.1. Section F presents novel pruning algorithms that we have developed to solve some of the larger continuously flexible designs that LUTE has made possible. Section G provides details of the protein design test cases described in Section 3.

A Calculation of expansion by least squares

The energy (as represented by LUTE) of a conformation $\mathbf{r}$ is a linear function of all the unpruned tuple energies $m(t)$, given by $\sum_{t \in T_{\mathbf{r}}} m(t) = \sum_{t \in T} m(t) I_{t \in T_{\mathbf{r}}}$, where the indicator function I_c is 1 if c is true and 0 otherwise. Thus, once we have decided which tuples to include in T, we can use least squares to obtain the energy $m(t)$ for each tuple $t \in T$.

We draw two sample sets for each fit—one for training and one for cross-validation. For each $\mathbf{r}$ in either of the sample sets, we evaluate the ground-truth $E(\mathbf{r})$ as described in Eq. (2) using local minimization. Each sample set is chosen such that each unpruned tuple appears in at least 10 samples. Furthermore, no sample is

allowed to contain any pruned tuples. To achieve this, we iterate through all the tuples. If the tuple appears in less than 10 samples, then we draw enough samples to cover the difference. To draw a sample containing a tuple t, we first assign the RCs in t to their respective residues. Then we iterate through each remaining position in random order, picking a random RC that does not introduce any pruned tuples into the sample. Because pairs and triples can be pruned, it is possible for this sampling process to reach a point where no RCs are available for a given position. Thus, if the first 5 random sampling attempts are unsuccessful, then we switch to depth-first search (with randomized ordering of RCs) to search for a sample. This way, if there is a sample containing t, we are guaranteed to find it, and if not, then t can be pruned.

The least-squares matrix for large problems is too big to allow efficient direct solution. Hence, we use conjugate gradient [Nocedal and Wright, 2006] on the least-squares normal equations, which is accurate and very efficient, especially since the least-squares matrix is sparse.

B Selection of tuples to include

The limiting behavior of LUTE is favorable. As we expand the set T, we must eventually approach perfect accuracy, because if T is the set of all tuples of RCs at different positions, then m can represent $E(\mathbf{r})$ for each full RC list $\mathbf{r}$ explicitly.

If we assume locality of $E(\mathbf{r})$ (see Section 1), we can expect inaccuracies to diminish fairly quickly with increasing size of T, because the component of $E(\mathbf{r})$ modeling the interactions of a residue i will depend only on the RCs assigned to residues fairly close in space to i. As a result, we expect a relatively compact LUTE expansion for any practical protein design problem.

For many designs, a pairwise expansion suffices—i.e., T is the set of RC pairs (i_r, j_s) where $i > j$. The accuracy of this pairwise expansion can be enhanced by first performing triples pruning—in this case, if a triple t is pruned, then we include t in T and set $m(t) = \perp$, but all $t' \in T$ such that $m(t') \neq \perp$ are pairs. However, sometimes higher accuracy is needed; we thus select additional, unpruned triples to add to T. To exploit locality, the triples are chosen at residue positions that have strong pairwise interactions. First, the interaction between each residue pair is quantified by maximizing the absolute value of the minimized pairwise interaction energy over all RC pairs at the residue pair. Then, for each residue position i, the 2 residues j and k interacting most strongly with i are identified (a higher number than 2 can be used if indicated by the least-squares residual). The residue pairs (i, j) and (i, k) are heuristically deemed "strongly interacting" pairs, and any residue triple containing at least two strongly interacting pairs is deemed a strongly interacting residue triple. Finally, all RC triples at strongly interacting residue triples are added to T. Though LUTE can support higher-order tuples than triples, we have not found this to be necessary so far.

C Choosing a pruning interval

If we know which RCs to prune, then the computation by least-squares of the LUTE matrix does not depend on the type of search problem we are solving (it can be single-state or multistate, and GMEC-based or ensemble-based). However, the most efficient pruning algorithm for RCs in a continuous search space, iMinDEE [Gainza *et al.*, 2012], requires as input a *pruning interval* in addition to a pairwise lower-bound energy matrix. The pruning interval is an upper bound on the gap between the optimal conformation and the lowest lower bound on a conformational energy in our search space, computed based on lower bounds on pairwise energies. The pruning is more efficient if this upper bound is relatively tight, and there are a few ways to obtain a valid but relatively tight bound. We will now discuss how this can be done for either single-state or multistate protein designs.

For single-state designs that aim to minimize an energy function with respect to sequence and (continuous) conformation, the method described in [Gainza *et al.*, 2012] to obtain a pruning interval is appropriate.

For multistate designs, two algorithms that are highly compatible with LUTE are K^* [Lilien *et al.*, 2005, Georgiev *et al.*, 2008] and COMETS [Hallen and Donald, 2015], and each of these algorithms offers an effective way to compute a pruning interval. K^*, a partition function-based binding optimization algorithm, provides methods to check if too many RCs have been pruned so that some can be unpruned if needed [Georgiev *et al.*, 2008]. This unpruning can be done by enlarging the pruning interval. COMETS optimizes, with respect to sequence, linear combinations of conformationally optimized energies of several protein states (bound states with different ligands, unbound states, etc.) Pruning is performed separately for each state, using type-dependent dead-end elimination [Yanover *et al.*, 2007]. Generally, for purposes of ensuring stability of each state, we will apply a constraint in COMETS calculations demanding that the optimized conformation energy for our optimized sequence be below some threshold. This threshold is our upper bound on the optimized energy for each state. Likewise, for each state, we can use standard protein design algorithms like DEE/A* [Leach and Lemon, 1998] to compute a lower bound based on pairwise energies for the optimal conformation over all sequences. Putting these bounds together, we obtain a pruning interval for each state.

Using these techniques, we can obtain a valid but relatively tight pruning interval and thus an appropriate LUTE energy matrix for many kinds of single-state and multistate designs (e.g., see Figs. 2, 3, 4, and S3).

D iMinDEE with LUTE

iMinDEE [Gainza *et al.*, 2012] calculates the GMEC for a protein system, and can also enumerate an ensemble of the lowest-energy conformations of a protein in gap-free ascending order of minimized conformational energy. We have implemented iMinDEE to run along with LUTE as part of the OSPREY [Gainza *et al.*, 2013,

Georgiev *et al.*, 2008, Georgiev *et al.*, 2009] open-source protein design software package. As in previous versions of iMinDEE, we estimate a pruning interval, prune based on that interval, calculate the GMEC, and then repeat with a larger interval if needed (at most one repeat is needed; see [Gainza *et al.*, 2012] for a proof). Thus, the protocol consists of the following steps:

1. Precalculate the matrix of energy lower bounds.
2. Prune using iMinDEE [Gainza *et al.*, 2012].
3. Compute an EPIC matrix [Hallen *et al.*, 2015].
4. Prune using EPIC (see Section F.2).
5. Perform LUTE fitting (see Section A).
6. Prune based on the LUTE matrix, using rigid DEE [Desmet *et al.*, 1992], since the LUTE matrix gives an accurate discrete expansion of the energy and thus admits pruning without the iMinDEE pruning interval.
7. Calculate the lower pairwise lower-bound. This is just like enumerating the first conformation from A* in iMinDEE. Though this step is in general NP-hard, very effective algorithms are available for it, as we have shown in [Roberts *et al.*, 2015].
8. Compute the GMEC using rigid A* [Leach and Lemon, 1998].
9. If needed, repeat the process (from step 2 onward) using a higher iMinDEE pruning interval; see [Gainza *et al.*, 2012].

So if we are calculating the GMEC, then the only NP-hard steps here are the pairwise lower-bound calculation and the GMEC calculation on the LUTE matrix, each of which are of similar complexity to enumerating the first conformation from iMinDEE, and comparable to computing a rigid GMEC. Thus, for large designs, the *entire* LUTE-based minimized GMEC (minGMEC) calculation is of similar complexity to enumerating the *first* conformation in traditional iMinDEE, or simply to performing a rigid GMEC calculation.

E BWM* with LUTE

We provide here a proof of Theorem 3.1, which establishes the time complexity of conformational search with LUTE as a function of residue interaction graph branch-width.

High-level intuition. The theoretical guarantees of BWM* [Jou *et al.*, 2015] are unchanged across different discrete formulations of the protein design problem. Given a sparse energy function, the corresponding sparse graph and branch-decomposition can be computed. BWM* computes the GMEC of the corresponding energy function in time exponential merely in w, the branch-width of the branch-decomposition. The time cost is polynomial in the number of residues in the system, n, and the maximum number of RCs per residue, q.

To realistically describe a protein design system, a LUTE energy function need only have nonzero coefficients for tuples whose interactions are nonnegligible, and thus we can compute a sparse residue interaction graph for it (Lemma E.1). As such, when a LUTE energy matrix whose residue interaction graph has branch-width w is provided as input to the protein design problem, BWM* computes the minimized GMEC in time exponential in w, enumerates a gap-free list of conformations in $\mathcal{O}(n \log q)$ additional time, and computes a gap-free ensemble that is guaranteed to contain the minGMEC.

Lemmata and Theorems. For an n-residue protein design with at most q rotamers at each position, let E be a LUTE energy function whose tuples have arity at most k. We can then define the residue interaction graph of the LUTE energy function as follows:

Definition. Let $E(r) = \sum_{r \in T_r} m(r)$ be a LUTE energy function, where $m : T \longrightarrow \mathbb{R} \cup \{\perp\}$ maps RC tuples to real coefficients. The *sparse residue interaction graph* of $E(r)$ is defined to be a hypergraph $G' = (V, T)$, where V is the set of all mutable residues.

Lemma E.1. *For a given LUTE energy function $E(r)$ with finite tuple arity k, G' can be computed in $\mathcal{O}(n^k)$ time.*

Proof. There are at most $\binom{n}{k}$ tuples, corresponding to at most $\binom{n}{k} = \mathcal{O}(n^k)$ edges in G'. □

Lemma E.2 (Given in [Jou *et al.*, 2015])**.** *Given a branch-decomposition with branch-width w, BWM* computes the GMEC of the sparse energy function in $\mathcal{O}(nw^2 q^{\frac{3}{2}w})$ time, and enumerates each additional conformation in order of increasing sparse energy in $\mathcal{O}(n \log q)$ time.*

Theorem 3.1. *For a LUTE energy function whose residue interaction graph has branch-width w, the GMEC can be computed in $\mathcal{O}(nw^2 q^{\frac{3}{2}w} + \beta_t)$ time and $\mathcal{O}(nwq^{\frac{3}{2}w})$ space, and each additional conformation can be enumerated in order of LUTE energy in $\mathcal{O}(n \log q)$ time and $\mathcal{O}(n)$ space.*

Proof. This follows from Lemmas E.1 and E.2. □

F Pruning enhancements

In order to complete some of the larger designs in Section 3, we found it necessary to enhance our pruning capabilities somewhat compared to previous work. Pruning consisted of Goldstein singles and pairs, and triples at selected, highly interacting triples of residues (using the triple selection criteria in Section B). In addition to this triple selection, we introduced two novel pruning enhancements, which build on iMinDEE [Gainza *et al.*, 2012]: a "competitor pruning" technique that speeds up pruning substantially with minimal loss of pruning power, and a "continuous pruning" technique that uses bounds on minimized energies of partial conformations to prune RCs and tuples of RCs that would otherwise require a much tighter iMinDEE pruning interval to prune.

F.1 Competitor pruning

DEE and related pruning algorithms can prune an RC or tuple of RCs by comparing its energies to those of a "competitor", which is another RC at the same residue (or tuple at the same tuple of residues). Previous implementations of these algorithms either allowed any unpruned RC or tuple to act as a competitor, or they only allowed a few "magic bullets," [Gordon and Mayo, 1998] which can reduce pruning power significantly. For some of the larger designs, pruning was the bottleneck, so it was important to have a method to speed up pruning without sacrificing much pruning power. We found that using a reduced set of competitors—the set of RCs that are unpruned using the iMinDEE pruning condition [Gainza *et al.*, 2012] with iMinDEE interval 0—results in such a speedup. To use this set, we begin pruning with a "competitor pruning" procedure, in which we perform pruning at iMinDEE interval 0. However, this pruning does not actually remove RCs and tuples from consideration. Rather, it removes them from the set of competitors that will be used in the the actual pruning step (Fig. S1).

Competitor pruning is fast because pruning with a 0 interval allows a large amount of single-RC pruning, so we spend much less time iterating through pairs. The set of competitors chosen this way is also typically much smaller than the full set of unpruned RC tuples. Hence, the actual pruning step is sped up significantly by using competitor pruning first. We have observed pruning power to be reduced very little when using competitor pruning, and the following analysis suggest why this is the case.

Suppose we are performing Goldstein pruning with a fixed set of witnesses. Then any RC tuple t_1 that can be pruned at all can be pruned using a competitor t_2 such that t_2 cannot be pruned with iMinDEE interval 0. We can show this by taking note of two facts. (1) Pruning with iMinDEE interval 0 is transitive with a fixed set of witnesses. That is, if tuple t_1 can be pruned using competitor t_2, and t_2 can be pruned using competitor t_3, then t_1 can be pruned using competitor t_3. (2) If an RC tuple t_1 can be pruned using competitor t_2 at an iMinDEE interval $I > 0$, then t_1 can be pruned using t_2 at iMinDEE interval 0. Now,

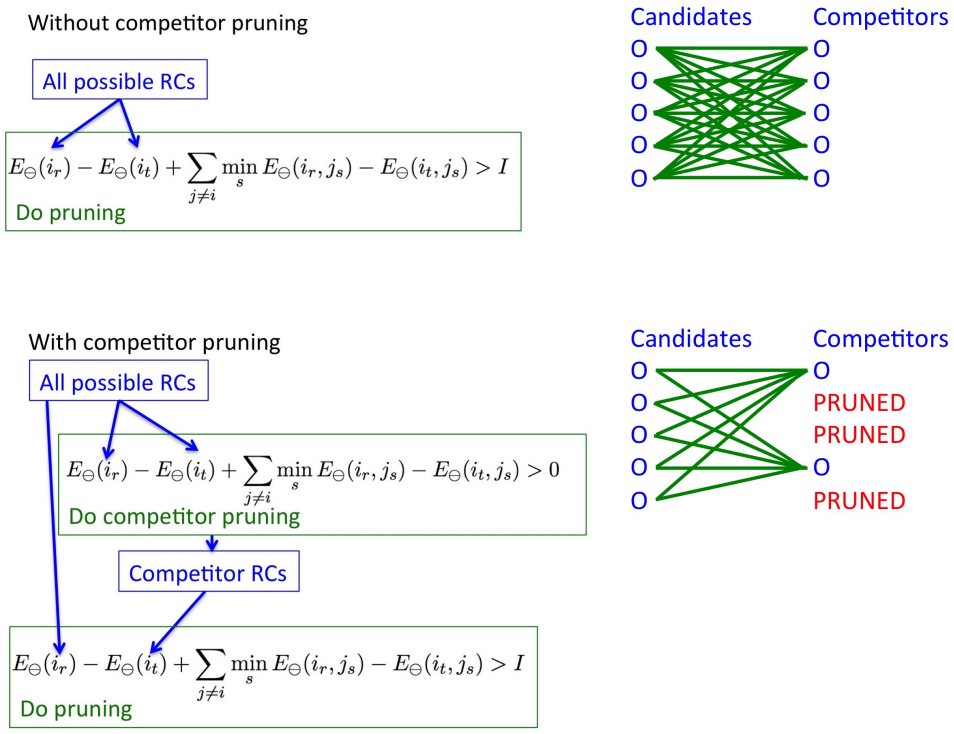

Figure S1: **Competitor pruning reduces the set of RCs we need to consider as competitors during iMinDEE pruning.** In iMinDEE pruning [Gainza *et al.*, 2012], we take a "candidate" RC (or RC pair or triple) i_r and compare it to another RC (or pair or triple) i_t by evaluating the pruning checksum $E_\ominus(i_t) - E_\ominus(i_t) + \sum_{j \neq i} \min_s E_\ominus(i_r, j_s) - E_\ominus(i_t, j_s)$. We compare this checksum to the pruning interval I, where $E_\ominus$ is a lower bound on a one-body or pairwise interaction energy. If the checksum exceeds I, then we can eliminate i_r as a possible constituent of the GMEC. Without competitor pruning (top), all RC tuples (blue) need to be considered as candidates or competitors, and thus the pruning condition (green) must be evaluated for every pair of tuples. But with competitor pruning (bottom), we first perform a modified pruning protocol to quickly eliminate those tuples that will not be effective as competitors. Then, when performing our actual pruning (to eliminate non-GMEC RC tuples), we can do so more quickly because we have less competitors to consider.

suppose that t_1 can be pruned using t_2 at an iMinDEE interval I. There are two cases: t_2 can be pruned at iMinDEE interval 0, or it cannot. If it can be pruned using a tuple t_3 at pruning interval 0, then t_1 can be pruned using t_3 at iMinDEE interval I (using notes 1 and 2). Recursively applying the same argument to t_3, we can be certain that t_1 can be pruned using a competitor that cannot be pruned with iMinDEE interval 0. In practice, a pruning cycle removes witnesses at each iteration, so the condition about a fixed set of witnesses does not strictly hold. But we have still observed pruning power to be reduced very little when using competitor pruning.

F.2 Continuous pruning

Another issue that can reduce pruning efficiency in iMinDEE is the fact that the lower bound on a candidate RC tuple $\mathbf{r}$ may not be very tight. We can tighten it by including the minimized partial conformational energy for the partial conformation consisting of the RCs in $\mathbf{r}$, as in the A* step for EPIC [Hallen *et al.*, 2015]. Suppose we have a lower bound for the energy of a partial conformation p based on pairwise energies. As discussed in [Hallen *et al.*, 2015], this bound remains valid if we add to it the minimized sum c of EPIC terms for all intra+shell and pairwise polynomials that represent RCs or pairs in p. iMinDEE is also based on this lower bound (the partial conformation here is the tuple we are trying to prune), so we can add c to the iMinDEE checksum (the left-hand side of the pruning condition) and still have a valid pruning condition with the same pruning interval I. c is always nonnegative, so continuous pruning is always at least as good as regular iMinDEE. $c = 0$ if the partial conformation is just a single RC, but may be quite large if the partial conformation contains RCs that do not pack together well, which is exactly when we want to prune. Effectively, continuous pruning allows us to group together terms in our lower bound on the conformational energy, so that the bound is tighter (Fig. S2).

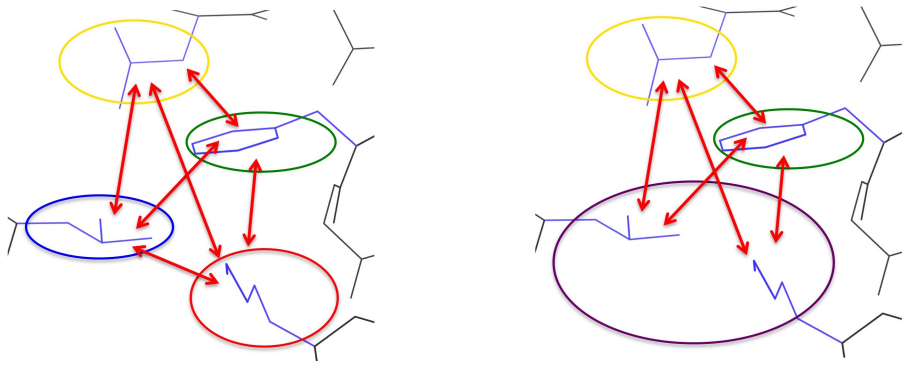

Figure S2: **Continuous pruning increases pruning power by using a tighter lower bound on conformational energy.** iMinDEE pruning is based on lower bounds on conformational energy, which are constructed by adding up lower bounds on pairwise interaction energies (red arrows) and one-body energies (circles). Without continuous pruning (left), each flexible residue and each flexible residue pair has its lower-bound energy computed separately. But with continuous pruning (right), if we add the continuous contribution c for a pair of RCs, then we effectively replace the three lower-bound terms for that pair (the two one-body energies and the pairwise interaction energy) with a single lower bound on the energy of the pair. This tighter bound can significantly increase pruning power, because if a residue pair has an unavoidable clash, the lower bound on the pair's internal energy is likely to show this even if each of the three constituent energies is individually capable of continuously minimizing to a better value.

G Details of protein design runs

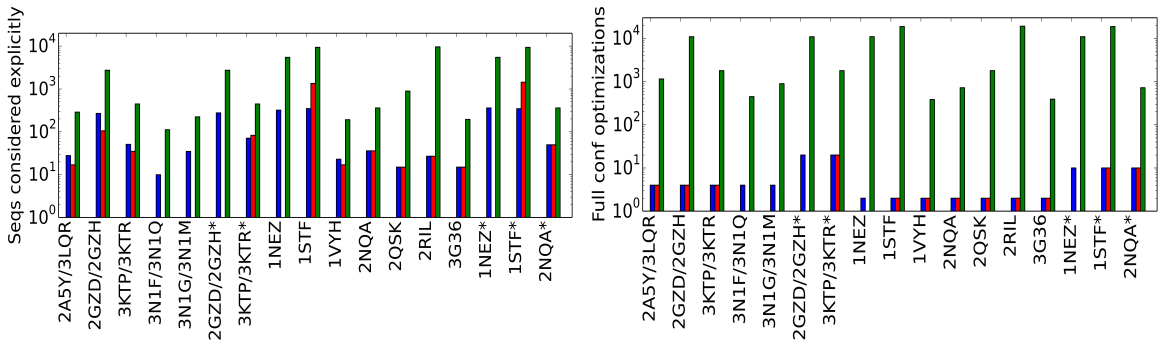

Figure S3: **LUTE brings rigid-rotamer-like efficiency to provably accurate multistate designs with continuous flexibility**. Top: The number of sequences considered explicitly in continuously flexible designs using LUTE with COMETS (blue) is similar to the number considered in discrete COMETS designs on the same system (red) and far less than the total number of sequences in the search (green). Bottom: In the same designs, the number of full conformational optimizations needed is also similar for the continuous (LUTE, blue) and discrete (red) designs. Usually, only the optimal sequence (or top 5 if enumerating 5 sequences) required full conformational optimization, while exhaustive search must fully conformationally optimize every sequence in every state in the multistate design (green). Design test cases taken from [Hallen and Donald, 2015]. * denotes enumeration of the top 5 sequences. Missing discrete calculations mean that discrete search was unable to find a non-clashing conformation for the wild-type protein, or that 5 sequences satisfying the design constraints were not available in the discrete search space.

Table S1: **Single-state protein design test cases**. Each design was run with continuous sidechain flexibility, and some also included continuous backbone flexibility. Designs were performed with and without LUTE, using EPIC [Hallen *et al.*, 2015] in both cases. Designs not finishing without LUTE (after three weeks of computation time) are marked DNF. n_L and n_n denote the number of nodes in the A* tree after enumeration of the GMEC (or of the last conformation if several conformations closely spaced in energy were calculated; see [Hallen *et al.*, 2015]), with and without LUTE respectively. The residual of the pairwise LUTE least-squares fit is denoted r_p. The final least-squares fit residual is denoted r_f. $r_f = r_p$ for designs with $r_p < 0.01$, but otherwise we computed a LUTE matrix with sparse triples and report the residual from the triples fit as r_f. These residuals are reported for the second iteration of iMinDEE, if two iterations were required (iMinDEE always finishes after at most two iterations) [Gainza *et al.*, 2012]. The residual from the first round of iMinDEE is typically much lower because of the greater amount of pruning in the first iteration. Table continues on next page.

Protein name	PDB code	Residue count	Backbone flexibility	n_L	n_n	r_p	r_f
Scorpion toxin	1aho	7	Y	42	751	0.0008	0.0008
Scorpion toxin	1aho	9	Y	49	402	0.002	0.002
Scorpion toxin	1aho	12	Y	46182	2149464	1.03	0.18
Cytochrome c553	1c75	6	Y	3	893	0.0004	0.0004
Atx1 metallochaperone	1cc8	7	Y	141	10816	0.05	0.006
Bucandin	1f94	7	Y	0	167	0.003	0.003
Nonspecific lipid-transfer protein	1fk5	6	Y	0	32	3×10^{-5}	3×10^{-5}
Transcription factor IIF	1i27	7	Y	137	28148	0.07	0.02
Ferredoxin	1iqz	9	Y	21	1550	0.007	0.007
Trp repressor	1jhg	7	Y	221	26079	0.06	0.01
Fructose-6-phosphate aldolase	1l6w	6	Y	0	132	0.0002	0.0002
PA-I lectin	1l7l	6	Y	0	35	5×10^{-10}	5×10^{-10}
Phosphoserine phosphatase	1l7m	7	Y	0	445	0.001	0.001
alpha-D-glucuronidase	1l8n	5	Y	0	1	7×10^{-23}	7×10^{-23}
Granulysin	1l9l	7	Y	0	29	0.0006	0.0006
Ferritin	1lb3	5	Y	0	155	0.0006	0.0006
Cytochrome c	1m1q	8	Y	0	1197	0.0004	0.0004
Hypothetical protein YciI	1mwq	8	Y	0	274	0.003	0.003
Ponsin	2o9s	14	N	205873	162750	4.52	1.93
Scytovirin	2qsk	10	N	0	10	1×10^{-11}	1×10^{-11}

Putative monooxygenase	2ril	8	N	71	8125	0.0002	0.0002
dpy-30-like protein	3g36	4	N	0	35	0.0008	0.0008
HIV gp120	3u7y	16	N	125	48935	3×10^{-5}	3×10^{-5}
Atx1 metallochaperone	1cc8	20	N	35670	DNF	0.10	0.003
Atx1 metallochaperone	1cc8	20	N	2	592642	0.006	0.006
Atx1 metallochaperone	1cc8	30	N	77	DNF	0.32	0.01
Atx1 metallochaperone	1cc8	40	N	38201	DNF	1.70	0.30
Scorpion toxin	1aho	17	N	198857	DNF	0.80	0.04

Table S2: **Multistate protein design test cases**. These designs systems were taken from [Hallen and Donald, 2015] and were run using the COMETS multistate design algorithm. Each design was run both with continuous flexibility, using LUTE, and without continuous flexibility. Type is "aff" for designs for affinity, "stab" for designs for stability robust to force field choice, and "multi" for designs to be multispecific to two different complexes (the types of designs, and the COMETS algorithm in general, are described further in [Hallen and Donald, 2015]). In each design, N is the number of sequences in the search space and k is the number of sequences enumerated (we enumerate either the top sequence or the top 5). m_L and m_d are the numbers of sequences explicitly considered using LUTE and in discrete search respectively. Similarly, g_L and g_d are the numbers of full conformational optimizations performed using LUTE and in discrete search respectively. "WC" indicates that the wild-type protein was found to have a clash unavoidable by discrete conformational search, and "NF" indicates that the discrete search provably cannot find five sequences that satisfy the design constraints. Table continues on next page.

Protein re-designed	Mutable residues	PDB id(s)	Type	N	k	m_L	m_d	g_L	g_d
CED-4	1, 5, 227, 229, 259, 265, 279, 282	2a5y/3lqr	multi	288	1	28	17	4	4
Rab-11A	44, 46, 47, 48, 50	2gzd/2gzh	multi	2744	1	269	106	4	4
Poly-adenylate-binding protein 1	564, 571, 580, 582, 584	3ktp/3ktr	multi	448	1	51	35	4	4
CDO	867, 872, 874, 901, 918	3n1f/3n1q	multi	112	1	10	WC	4	WC
Brother of CDO	753, 756, 758, 760, 789, 804	3n1g/3n1m	multi	224	1	35	WC	4	WC
Rab-11A	44, 46, 47, 48, 50	2gzd/2gzh	multi	2744	5	277	NF	20	NF
Poly-adenylate-binding protein 1	564, 571, 580, 582, 584	3ktp/3ktr	multi	448	5	71	83	20	20
Beta-2-micro-globulin	52, 54, 56, 57, 63	1nez	aff	5488	1	321	WC	2	WC
Papain	18, 19, 21, 159, 177, 181	1stf	aff	9408	1	350	1349	2	2
PAF-acetyl-hydrolase	194, 212, 235, 236, 238, 254, 316	1vyh	aff	192	1	23	17	2	2
Leupeptin inhibitor	356, 357	2nqa	aff	361	1	36	36	2	2

Scytovirin	1, 6, 10, 13, 28, 43, 48, 58, 61, 76	2qsk	stab	896	1	15	15	2	2
Putative monooxygenase	5, 13, 21, 55, 57, 59, 61, 70	2ril	stab	9604	1	27	27	2	2
dpy-30-like protein	64, 68, 87, 91	3g36	stab	196	1	15	15	2	2
Beta-2-microglobulin	52, 54, 56, 57, 63	1nez	aff	5488	5	362	NF	10	NF
Papain	18, 19, 21, 159, 177, 181	1stf	aff	9408	5	346	1441	10	10
Leupeptin inhibitor	356, 357	2nqa	aff	361	5	50	50	10	10