

Solving the Hydrophobic-Polar model with Nested Monte Carlo Search

Milo Roucairol and Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. In this paper we present a new Monte Carlo Search (MCS) algorithm for finding the ground state energy of proteins in the Hydrophobic-Polar-model (HP model). We also compare it to other MCS algorithms not usually used on the HP model as well as to other approaches and provide an overview of the state of the art algorithms used on the HP model.

Keywords: Protein folding · Monte Carlo Search · MCTS · HP model · optimization · pruning.

1 Introduction

Monte Carlo search algorithms have proven to be powerful as game playing agents, with recent successes like AlphaGo [14]. These algorithms have the advantage of only needing an evaluation function for the final state of the space they explore.

Protein folding is crucial to our understanding of biology and designing drugs, however, trying our algorithms directly on accurate models could be counter-productive. In this paper, we use a new MCS algorithm to fold proteins in a simplified lattice based model called the HP model.

First, we will present the protein folding and the HP model, then the different algorithms we used to explore the problem space and finally the results of our experiments.

2 The problem

2.1 Protein folding

With recent developments in ARNm technology, it is now possible to incite cells to produce a specific protein [8], like the spike protein used in COVID-19 vaccines. Unfortunately, deducing the shape a protein will take given the amino acids sequence is not obvious nor trivial. That is a reason protein folding is a very important problem in molecular biology and medicine.

Proteins are chains of amino acids (primary structure), they can fold in many different ways, the secondary structure is the shape the protein will take at a local level (a coil for example), the tertiary structure is the global shape of the

protein with less discernible patterns, finally, the quaternary structure is how a protein can assemble with another. Here we are interested in predicting the ground state energy folding (secondary and tertiary structure) from the primary structure. Many forces drive the folding, which prevents the creation of a very accurate simulator, the main driving force is the hydrophobic one.

One can not approach protein folding without mentioning DeepMind’s AlphaFold [10]. Placing first at the Critical Assessment of Techniques for Protein Structure Prediction in 2018 and 2020, it is the best program for protein structure prediction yet. AlphaFold uses machine learning on a large protein database to train neural networks, in addition to physics based rules in order to predict the folding of a protein.

AlphaFold is the greatest achievement to protein folding prediction in decades, but the research is not over yet. AlphaFold accuracy can still be perfected, and by using neural networks the explainability is low and the model may not be able to predict the structure of proteins never seen before. Our objective here is to provide a better algorithm than the other MCS algorithms for Monte Carlo physics simulation, which may be more explainable than AlphaFold and other MCS algorithms, but should currently be way less accurate than AlphaFold if it was applied on the same problem.

2.2 Hydrophobic-Polar model

The HP model was introduced in 1985 by Ken Dill [4]. The main idea behind the creation of the HP model is that the Hydrophobic-Polar (HP) force is the main force driving the folding of a protein, thus it is the only one used here.

The HP model is a simplified lattice based model for protein folding, it exists in 2D and 3D versions.

In the HP model, proteins are represented as a chain of H and P residues (amino acids), the chain is then folded onto a grid, two residues can not share the same positions. The residue contacts determine the energy of a chain, usually, the reward for an H-H connection is -1, and 0 for H-P and P-P contacts in a context of minimization (since the ground energy state is the state with the least potential energy). Other rewards can be used to obtain different results or guide the search.

2.3 State of the art on HP model

The HP model has seen a number of algorithms trying to solve it. All of the best performing algorithms on the HP model are Monte Carlo based, policy learning using neural networks or reinforcement learning like NRPA [12] led to poor results. In these Monte Carlo algorithms, we can identify two types, the chain growth algorithms and the replica exchange ones.

The chain growth methods add the residues one after the other, next to the previous one, it is similar to a self avoiding walk and it is the method we used in our own algorithm.

The replica exchange methods use pull moves, pulling the chain at one point by rotating a residue around one of its neighbors, symmetrically rotating a part of the chain or pulling from one side of the chain. This means the entirety of the chain is present on the lattice at any given state of the search and is in a physically possible conformation, this method is used in simulated annealing like Monte Carlo algorithms. To see a representation of these moves see Chris Thchuk, Alena Shmygelska, and Holger H Hoos REMC article [16].

Here is a short review of the methods we encountered.

1) PERM: Initially used on Self Avoiding Walks (SAW), PERM is a chain growth algorithm and was used on the HP model by Peter Grassberger in 1997 [7]. It stands for Pruned Enriched Rosenbluth Method, the idea is to explore the possible chains uniformly with a bias on the immediate gain, cutting (pruning) branches leading to too few choices and poor performances, and cloning (enriching) branches that lead to great results. PERM has seen many new versions until 2011 [9], mainly proposed by its creator, Peter Grassberger. It still is one of the best algorithms available but has been outperformed by pull-moves based more recent algorithms. We reproduced PERM, but our results with this algorithm did not live up to the expectations, it matched UCT's (see part 4.3) performances.

2) REMC: Introduced in 2007 by Chris Thachuk, Alena Shmygelska and Holger H Hoos [16], the Replica Exchange Monte Carlo algorithm uses pull moves and simulated annealing. That algorithm keeps only a certain number of replicas (it was determined the best number of replicas for the 3D HP model was 2), each with a given temperature. At each step the algorithm mutates each replica with a Monte-Carlo Search using the pull moves, the probabilities of keeping a mutation are decided by the score gain (energy loss) of the mutated state and the temperature. Then, once each state is produced through mutation, the replicas are then again swapped probabilistically according to their score (energy) and temperatures.

3) Wang-Landau sampling: Introduced in 2012 on the HP model, the Wang-Landau sampling (WLS) [17] method seems to be the new best algorithm for solving the HP model. It is a replica-exchange (simulated annealing) algorithm that uses the same pull moves as the REMC, but also uses moves consisting in cutting and joining of the molecule (thus reallocating all the residues according to their position), together they are named the Monte-Carlo trial moves. With these moves, the WLS explores the conformation space to estimate a histogram of the energies of these conformations. With this histogram, the WLS can then direct the exchange of the replicas.

3 Algorithm

3.1 Biased Growth

In a similar way to the PERM algorithm [9], we try to favor the immediate reward when building/folding the molecule. To do this, we use biased playouts, the chances of selecting a move m from M the possible moves from state S follows a softmax distribution with b the bias factor:

$$\frac{\exp(G_m * b)}{\sum_{i \in M_S} \exp(G_i * b)} \quad (1)$$

$M_{c-state}$ denotes the legal moves available from the state *current-state*.
 $G_{M_{c-state}}$ denotes the immediate gains of each legal moves available from the state *current-state*.

Algorithm 1 The biased growth playout algorithm.

```

1: function PLAYOUT(c-state, b)
2:   ply  $\leftarrow$  0
3:   seq  $\leftarrow$  {}
4:   while c-state is not terminal do
5:     gains  $\leftarrow$   $G_{M_{c-state}}$ 
6:     move  $\leftarrow$  softMaxChoice( $M_{c-state}$ , gains * b)
7:     current-state  $\leftarrow$  play(c-state, move)
8:     seq[ply]  $\leftarrow$  move
9:     ply+ = 1
10:  end while
11:  return score(current-state), seq
12: end function

```

3.2 Nested Monte Carlo Search

NMCS [1] is a Monte Carlo Search algorithm that recursively calls lower level NMCS on children states of the current state in order to decide which move to play next, the lowest level of NMCS being a random playout, selecting uniformly the move to execute among the possible moves. A heuristic can be added to the playout move choices, and it is the case here with the biased growth playouts.

Algorithm 2 gives the NMCS algorithm, l is the nesting level and b the playout bias.

3.3 Lazy Nested Monte Carlo Search

The lazy NMCS inherits its main features from the NMCS, but solves an obstacle encountered on this problem. Solving the 3D HP model with the NMCS requires

Algorithm 2 The NMCS algorithm.

```

1: function NMCS(c-state, l, b)
2:   if l = 0 then return playout(c-state, b)
3:   else
4:     best-score  $\leftarrow -\infty$ 
5:     best-sequence  $\leftarrow []$ 
6:     ply  $\leftarrow 0$ 
7:     while c-state is not terminal do
8:       for each move in  $M_{c\text{-state}}$  do
9:         n-st  $\leftarrow \text{play}(c\text{-state}, \text{move})$ 
10:        (score, seq)  $\leftarrow$  NMCS(n-st, l - 1, b)
11:        if score  $\geq$  best-score then
12:          best-score  $\leftarrow$  score
13:          best-sequence[ply..]  $\leftarrow$  move + seq
14:        end if
15:      end for
16:      next-move  $\leftarrow$  best-sequence[ply]
17:      ply  $\leftarrow$  ply + 1
18:      c-state  $\leftarrow$  play(c-state, next-move)
19:    end while
20:    return (best-score, best-sequence)
21:  end if
22: end function

```

using a level of 4 at least, however, it requires computing many 3 level NMCS, already very costly, one for each possible move the level 4 NMCS can make. The main idea behind the lazy NMCS is that there are moves that lead to low potential states, to do so, we estimate the potential of a state by launching a number of biased growth playouts and calculating the mean of their scores, then we compare that score to a threshold (relative to the number of moves already done) calculated from the previous estimations to decide if we want to expand the search tree from this state, or prune it. To update the pruning threshold tr , it is possible to use a mean, a median, or a max from the previous estimations, here we use the max as it gave the best results on these problems.

In the following pseudocode in algorithm 3, p is the number of playouts used to evaluate a state and r is the ratio to the threshold a state will be pruned on. l is the nesting level and b is the playout bias.

From line 9 to line 14, the state is evaluated with the mean of p playouts.

From line 15 to line 17, the threshold list is extended on the first entry of a new molecule length, this step is not needed if the list is initialized with the right size from the start for problems we know the maximum number of moves that will be played.

From lines 18 to 120, the threshold is updated with the evaluation.

From lines 21 to 25, it is decided with the evaluation, the pruning ratio and the corresponding threshold if the search will be costly or not.

Algorithm 3 The Lazy NMCS algorithm.

```

1:  $tr \leftarrow []$ 
2: function LNMCS( $c-st, l, b, p, r$ )
3:   if  $level = 0$  then return  $PLAYOUT(c-st, b)$ 
4:   else
5:      $best-score \leftarrow -\infty$ 
6:      $best-sq \leftarrow []$ 
7:      $ply \leftarrow 0$ 
8:     while  $c-st$  is not terminal do
9:       for each  $move$  in  $M_{c-state}$  do
10:         $n-st \leftarrow play(c-st, move)$ 
11:        for  $i$  in  $0..p$  do
12:           $(playoutSc, \_) \leftarrow playout(n-st, b)$ 
13:           $es \leftarrow es + playoutSc/p$ 
14:        end for
15:        if  $tr.length() < c-st.nbplay + 1$  then
16:           $tr.push(0.0)$ 
17:        end if
18:        if  $tr[c-state.nbplay] < es$  then
19:           $tr[c-state.nbplay] \leftarrow es$ 
20:        end if
21:        if  $es < ratio * tr[c-st.nbplay]$  then
22:           $(sc, sq) \leftarrow LNMCS(c.1, 0, b, p, r)$ 
23:        else
24:           $(sc, sq) \leftarrow LNMCS(c.1, l - 1, b, p, r)$ 
25:        end if
26:        if  $sc \geq best-score$  then
27:           $best-score \leftarrow sc$ 
28:           $best-sq[ply..] \leftarrow move + sq$ 
29:        end if
30:      end for
31:       $next-move \leftarrow best-sq[ply]$ 
32:       $ply \leftarrow ply + 1$ 
33:       $c-st \leftarrow play(c-st, next-move)$ 
34:    end while
35:    return  $(best-score, best-sq)$ 
36:  end if
37: end function

```

As you can see there is only one FOR loop iterating over the moves in this implementation of the LNMCS, which means the evaluation is incomplete when the algorithm decides whether to prune the first branches or not. This is a minor flaw in this version of the algorithm and it is easily fixed by ulterior versions (along with other shortcomings). Nonetheless, the experiments were made with this "prototype" version of the algorithm.

4 Results

4.1 Lazy Nested Monte Carlo Search

Experimental setup We conducted experiments on the 10 molecules with 48mers from the benchmark we can find in Holger's [16] and Hsu's [9] work, here on table 1. $-E^*$ denotes (the opposite of) the speculated energy of the lowest energy state.

ID	molecule	$-E^*$
1	HPHHPRHNNHPRHNNHPRHHPRH HHRHRHNNHPRHPPHPPPPPPRH	32
2	HHNNHRHNNHNNHPRHPPHNNHPRH PPPPRHPRHPPHPPHNNHPRHNNHPRH	34
3	PHRHHPRHNNHNNHPRHPRHPRHNN PHRHPPPHPRHNNHPRHNNHPRHPRHPRH	34
4	PHRHHPRHPRHNNHPRHNNHPRHPPRH HHNNHPRHNNHPRHPPPPHPRHPRHPRH	33
5	PPHPPRHPRHNNHPRHNNHPRHNNHPRH HHRHRHPRHPPHPPPPHNNHNNHPRH	32
6	HHNNHPRHNNHPRHNNHPRHNNHPRHPP PPPPRHPRHPPHPPHPPHNNHNNHPRH	32
7	PHPPPPRHNNHPRHNNHNNHPRHNNHPRH HPPRHPRHPPHNNHPRHNNHPRHPPRH	32
8	PHNNHNNHPRHNNHPRHNNHPPPPPP HRRHNNHPRHPPHNNHPRHNNHPRHPP	31
9	PHRHPPPRHPRHPRHPRHNNHNNH PRHNNHPRHPRHNNHPRHNNHPRHPPRH	34
10	PHNNPPPPHNNHPRHNNHPRHPRH HHRHPRHPRHNNHPRHNNHNNHPRHNN	33

Table 1: The benchmark's molecules and their speculated lowest energy state

To obtain our results, we used the lazy NMCS with a timeout of 150s, if the algorithm has not found a conformation with the lowest known energy before the end of the timeout then we restart the algorithm until that happens. In our experiments, the payout biased growth gives an immediate gain of 1 to any legal

H-H connection, but also a penalty of -0.2 to the HP connections, this was made in order to incite the biased growth to keep a maximum of H mers open to future connections, we did not experiment on that variable.

Molecule 4 used a lazy NMCS with a threshold based on the mean of the evaluation playouts with the following parameters:

level	4
#eval playouts	10
pruning ratio	0.97
playout bias	20

The other molecules used a lazy NMCS with a threshold based on the best average from a batch of evaluation playouts with the following parameters:

level	5
#eval playouts	20
pruning ratio	0.9
playout bias	20

Results Different methods of evaluation and pruning can greatly change the performance of the algorithm, and some methods can be ineffective on a set of molecules while being capable on another set.

ID	mean time	interquartile
1	5.5	5
2	12.5	15.5
3	10	14
4	20	25
5	10	10
6	+ - 180	-
7	+ - 60	-
8	13.5	12
9	+ - 120	-
10	7	9

Table 2: Mean time and interquartile (in minutes) of LNMCS on the benchmark molecules to reach a state with the lowest state energy

These results are displayed in minutes and were obtained on a 3.50GHz Intel core i5-6600K CPU, the lowest state energy are the -E* featured in table 1.

Our LNMCS performed very poorly on molecules 6 and 7, we were not able to gather enough data to compute the statistics. This was unexpected since only molecules 4 and 9 are difficult for PERM (the state of the art chain growth algorithm) to solve according to Grassberger and Hsu’s latest paper [9], and molecule 4 posed less problems. However, the lazy NMCS could attain the second best energy level very reliably in less than 150s in half of the launches with both molecules.

LNMCS was also able to easily reach the second best level of energy on molecule 9 but could only reach the optimal state every 2 hours approximately, that result was expected since PERM encounters difficulties with that molecule too.

4.2 Comparing LNMCS to Deep Reinforcement Learning in AlphaZero style

Experimental setup Recent results on the HP model were brought to our attention in Deng et al’s work [3]. Experiments were realized on 8 other HP strings in table 3 using a neural network based Monte Carlo Search method, in AlphaZero’s manner. To the best of our knowledge, this is a first for the HP model.

ID	molecule	-E*
S1	HPHPPHHRPHRPPHRRHPPH	11
S2	HHPPHPPHPPHPPHPPHPPHPPH	13
S3	PPHPPHHPPPPHPPPPHPPPPH	9
S4	PPPHHPPHHPPPPPPHHHHHHHPPH PPPHHPPHPP	18
S5	PPHPPHHPPHHPPPPPPHHHHHHHHH HPPPPPPHPPHPPHPPHPPHPPH	31
S6	HHRHRHRHRHHHHRHRPPHPPHPP PPHPPPPHPPHPPHPPHPPHPPHPPH	31
S7	PPHHHPPHHHHHHHPPPPHHHHHHHH HHRHRPPHHHHHHHHHHHHHPPPP HHHHHPPHPPH	52
S8	HHHHHHHHHHHHHPPHPPHPPHPPHPP PHPPHPPHPPHPPHPPHPPHPPHPPH	56

Table 3: The second benchmark’s molecules and their speculated lowest energy state

We launched our LNMCS with a threshold based on the best average from a batch of evaluation playouts with the same parameters as with most previous results:

level	5
#eval playouts	20
pruning ratio	0.9
playout bias	20

Results In table 4 we show that LNMCS is able to outperform this recent method as well.

ID	our -E*	LNMCS time to previous -E*	previous -E*
S1	11	30s	11
S2	13	25s	13
S3	9	5s	9
S4	18	0.73s	18
S5	31	103s	31
S6	31	143s	31
S7	54	14s	52
S8	58	135s	56

Table 4: Best scores obtained by LNMCS on the second benchmark molecules

Due to a typo on S6 in [3], molecule S6 was retrieved from [13]. The times displayed in table 4 show approximately how much time is necessary to reach the previous lowest energy state for each molecule.

As you can see on table 4, our LNMCS was able to very quickly get to the lowest know state energy and was even able to find new optimums for molecules S7 and S8. However, despite the LNMCS performing better, our own UCT with biased playouts is inferior to their UCT with the priors approach. We think using priors on the LNMCS could lead to even better results than the ones presented in this paper.

4.3 Other MCS algorithms

We tried to solve The HP model with a variety of different Monte Carlo algorithms. We only applied these algorithms to the first molecule from the benchmark, the results presented in this section are only to give an idea of these algorithms' performances and do not necessarily reflect their potential on the HP model.

Nested Monte Carlo Search The good performances of the NMCS [1] compared to the other algorithms presented in this section is what made us decide to try to improve it for this problem into the LNMCS: the NMCS was able to find the optimal value on the molecule in less than 10mn.

In Figure 2 and Figure 1 we compare performances of the level 5 NMCS and the level 5 LNMCS with a ratio of 0.9 on molecule 1, with both a playout bias of 20 over 20 runs with a 150s timeout.

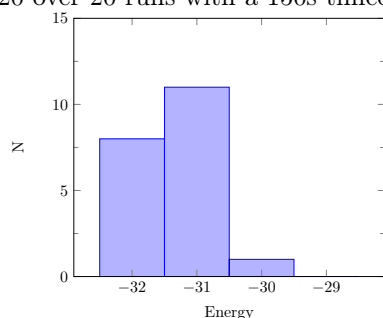


Fig. 1: Energy distribution with the Lazy NMCS

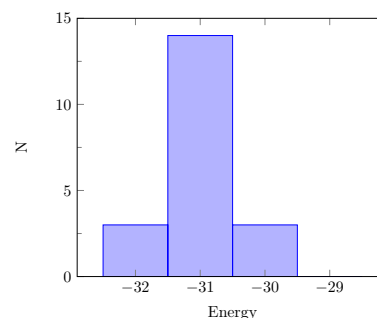


Fig. 2: Energy distribution with the NMCS

As you can see in figures 1 and 2, the LNMCS provides a substantial performance gain over the NMCS on this problem. Lowest energy conformations are way sparser than the second lowest energy conformations, being able to reach them 3 or 4 times as often is a great improvement.

Nested Rollout Policy Adaptation The NRPA [12] is similar to the NMCS, the main difference is that the NRPA learns a policy to decide which move play during playouts, that policy discovery is interesting for many problems which are too complex to implement a handmade policy (like we did here). On many problems the NRPA outperforms the NMCS, however, in this case it was not able to. The performance of the NRPA and GNRPA [2] are widely dependent on the move representations, here it is the number corresponding to the number of residues already placed and its direction and NRPA and GNRPA with a bias of 20 were not able to reach the optimal energies (-28 or -29 for the GNRPA when -32 is the best known). Other moves representations were tried, using the last few previous moves instead of the number of residues already placed for example, but none were able to provide better performances. Our inability to obtain good results with NRPA does not mean it is impossible to solve this problem with it.

Greedy Best First Search with playouts The Greedy BFS [5] is a simple search algorithm that uses a ranked list of the nodes to open according to their scores given by an evaluation function. Iteratively, the Greedy BFS opens the best node from the list and launches the evaluation function on every child of this node to insert them in the ranked list. Here we evaluate the children with their results with one or multiple playouts. This method converges rapidly to a "good enough" solution (a local minimum), -29 when the best known is -32 on molecule 1, but then improves very little, it is due to a large number of good scoring states that do not lead to an optimal solution, making the search too exhaustive. Pruning the search tree could improve the results of this method.

Upper Confidence bounds applied to Trees UCT [11] is the most widely used MCTS method, namely in outstanding works like alphazero [15] and AstraZeneca’s AiZynthFinder [6]. It iteratively starts from the initial state, do down the search tree following a formula and launches layouts once it finds an unvisited state, based on the results of each playout, the value used to determine how to go down the search tree are updated. UCT does not work well on the HP model without biased growth (achieving about the same scores as a single biased playout, -18), with biased growth it achieves scores around -28/-29 on molecule 1, like the other non-NMCS algorithms discussed here.

5 Conclusion

In this paper, we proposed a new (prototype) Monte Carlo Search algorithm that has the advantage to be easier to implement than most usual HP model algorithms and is applicable to more problems. It is also shown to be an improvement on the NMCS algorithm for this specific problem and to outperform novel neural network based methods. We already have applied LNMCS to more problems, and it has not been outperformed by any other algorithm on any of these problems yet, we also made slight changes to the algorithm towards a final version (hence the use of the term "prototype" here) in papers soon to be published, hopefully.

In future works, we aim to apply LNMCS to more other problems and find ways to improve its performance. We also aim to use a prior neural network to further improve LNMCS performance on the HP model problem.

References

1. Cazenave, T.: Nested Monte-Carlo Search. In: Boutilier, C. (ed.) IJCAI. pp. 456–461 (2009)
2. Cazenave, T.: Generalized nested rollout policy adaptation. In: Monte Carlo Search at IJCAI (2020)
3. Deng, H., Yuan, X., Tian, Y., Hu, J.: Neural-augmented two-stage monte carlo tree search with over-sampling for protein folding in hp model. *IEEEJ Transactions on Electrical and Electronic Engineering* **17**(5), 685–694 (2022)
4. Dill, K.A.: Theory for the folding and stability of globular proteins. *Biochemistry* **24**(6), 1501–1509 (Mar 1985)
5. Doran, J.E., Michie, D.: Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* **294**(1437), 235–259 (1966)
6. Genheden, S., Thakkar, A., Chadimová, V., Reymond, J.L., Engkvist, O., Bjerrum, E.: AiZynthFinder: a fast, robust and flexible open-source software for retrosynthetic planning. *Journal of Cheminformatics* **12**(1), 70 (Dec 2020). <https://doi.org/10.1186/s13321-020-00472-1>, <https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00472-1>
7. Grassberger, P.: Pruned-enriched Rosenbluth method: Simulations of polymers of chain length up to 1 000 000

8. Gros, F., Gilbert, W., Hiatt, H.H., Attardi, G., Spahr, P.F., Watson, J.D.: Molecular and Biological Characterization of Messenger RNA. Cold Spring Harbor Symposia on Quantitative Biology **26**, 111–132 (Jan 1961), publisher: Cold Spring Harbor Laboratory Press
9. Hsu, H.P., Grassberger, P.: A review of Monte Carlo simulations of polymers with PERM. Journal of Statistical Physics **144**(3), 597–637 (Aug 2011)
10. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S.A.A., Ballard, A.J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A.W., Kavukcuoglu, K., Kohli, P., Hassabis, D.: Highly accurate protein structure prediction with AlphaFold. Nature **596**(7873), 583–589 (Aug 2021)
11. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Machine Learning: ECML 2006, vol. 4212, pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), series Title: Lecture Notes in Computer Science
12. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 649–654 (2011)
13. Santana, R., Larrañaga, P., Lozano, J.A.: Protein folding in simplified models with estimation of distribution algorithms. IEEE transactions on Evolutionary Computation **12**(4), 418–438 (2008)
14. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
15. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815 [cs] (Dec 2017), <http://arxiv.org/abs/1712.01815>, arXiv: 1712.01815
16. Thachuk, C., Shmygelska, A., Hoos, H.H.: A replica exchange Monte Carlo algorithm for protein folding in the HP model. BMC Bioinformatics **8**(1), 342 (Dec 2007)
17. Wüst, T., Landau, D.P.: Optimized Wang-Landau sampling of lattice polymers: Ground state search and folding thermodynamics of HP model proteins. The Journal of Chemical Physics **137**(6), 064903 (Aug 2012)