Solving the HP model with Nested Monte Carlo Search

Anonymous submission

Abstract

In this paper we present a new Monte Carlo Search (MCS) algorithm for finding the ground state energy of proteins in the HP-model. We also compare it briefly to other MCS algorithms not usually used on the HP-model and provide an overview of the algorithms used on HP-model.
The algorithm presented in this paper does not beat state of the art algorithms, see PERM (Hsu and Grassberger 2011), REMC (Thachuk, Shmygelska, and Hoos 2007) or WLRE (Wüst and Landau 2012) for better results.

Introduction

Monte Carlo search algorithm have proven to be powerful as game playing agents, with recent successes like AlphaGo(Silver et al. 2016). 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 counterproductive. 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.

The problem

Protein folding

With recent developments in ARNm technology, it is now possible to incite cells to produce a specific protein (Gros et al. 1961), like the spike protein used in COVID-19 vaccines. Unfortunately, deducing the shape a protein will take based 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 into predicting the ground state energy folding (secondary and tertiary structure) from the primary structure. Many forces drive the folding, which prevent 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 (Jumper et al. 2021). 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 structure of proteins never seen before. Our objective here is to provide a better algorithm for Monte Carlo physics simulation, which may be more explainable than AlphaFold, but is currently way less accurate.

HP model

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 very 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 energy of a chain is determined by the residue contacts, usually, the reward for an H-H connection is -1, and 0 for H-P and P-P contacts in a context of minimisation (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.

State of the art on HP model

The HP model was introduced in 1985 by Ken Dill (Dill 1985), it 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 (Rosin 2011) 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 neighbor, 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 research, 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 check Chris Thchuk, Alena Shmygelska and Holger H Hoos REMC article (Thachuk, Shmygelska, and Hoos 2007).

Here is a short review of the methods we encountered.

1) PERM : Initially used on SAW, PERM is a chain growth algorithm and was used on the HP model by Peter Grassberger in 1997 (Grassberger). It stands for Pruned Enriched Rosenbluth Sampling, 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 (Hsu and Grassberger 2011), 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.

2) REMC : Introduced in 2007 by Chris Thachuk, Alena Shmygelska and Holger H Hoos (Thachuk, Shmygelska, and Hoos 2007), the Replica Exchange Monte Carlo algorithm uses pull moves and simulated annealing. That algorithm keeps only a certain number of replicas (in [REMC], they 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 to keep a mutation are decided by the score gain (energy loss) of the mutated state and the temperature. Then, once each state is produced trough mutation, the replicas are then again swapped probabilisticaly, again according to their score (energy) and temperatures.

3) Wang-Landau sampling : Introduced in 2012 on the HP model, the Wang-Landau sampling (WLS) (Wüst and Landau 2012) 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 an histogram of the energies of these conformations. With this histogram, the WLS can then direct the exchange of the replicas.

Algorithm

Biased Growth

In a similar way to the PERM algorithm (Hsu and Grassberger 2011), 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 with an immediate gain G follows a softmax distribution with b the bias factor:
$\frac{exp(G[m]*b)}{sum([exp(G[i]*b) for\ i\ in\ M])}$.

$M_{current\text{-}state}$ denotes the legal moves available from the state $current\text{-}state$.
$G_{M\,current\text{-}state}$ denotes the immediate gains of each legal moves available from the state $current\text{-}state$.

Algorithm 1: The biased growth playout algorithm.

function PLAYOUT($current\text{-}state$, b)
  $ply \leftarrow 0$
  $seq \leftarrow \{\}$
  while $current\text{-}state$ is not terminal **do**
    $gains \leftarrow G_{M\,current\text{-}state}$
    $move \leftarrow softMaxChoice(M_{current\text{-}state}, gains * b)$
    $current\text{-}state \leftarrow play(current\text{-}state, move)$
    $seq[ply] \leftarrow move$
    $ply += 1$
  end while
  return $score(current\text{-}state), seq$
end function

Nested Monte Carlo Search

Algorithm 2: The NMCS algorithm.

function NMCS($current\text{-}state$, l, b)
  if $l = 0$ **then return** $playout(current\text{-}state, b)$
  else
    $best\text{-}score \leftarrow -\infty$
    $best\text{-}sequence \leftarrow []$
    $ply \leftarrow 0$
    while $c\text{-}state$ is not terminal **do**
      for each $move$ in $M_{c\text{-}state}$ **do**
        $n\text{-}st \leftarrow play(current\text{-}state, move)$
        $(score, seq) \leftarrow$ NMCS($n\text{-}st, l-1, b$)
        if $score \geq best\text{-}score$ **then**
          $best\text{-}score \leftarrow score$
          $best\text{-}sequence[ply..] \leftarrow move + seq$
        end if
      end for
      $next\text{-}move \leftarrow best\text{-}sequence[ply]$
      $ply \leftarrow ply + 1$
      $c\text{-}state \leftarrow play(c\text{-}state, next\text{-}move)$
    end while
    return $(best\text{-}score, best\text{-}sequence)$
  end if
end function

NMCS (Cazenave 2009) 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.

Lazy Nested Monte Carlo Search

Algorithm 3: The Lazy NMCS algorithm.

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

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

From line 16 to 18, the threshold is updated with the evaluation.

From line 19 to 23, it is decided with the evaluation, the pruning ratio and the corresponding threshold if the search will be costly or not.

As you can see there is only one FOR loop iterating over the moves in this implementation of the LNMCS, it means the evaluation is incomplete when the algorithm decides wether 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). Nonethless, the experiments were made with this "prototype" version of the algorithm.

Results

Lazy Nested Monte Carlo Search

We conducted experiments on the 10 molecules with 48 mers from the benchmark we can find in Holger's (Thachuk, Shmygelska, and Hoos 2007) and Hsu's (Hsu and Grassberger 2011) work.

ID	molecule	-E*
1	HPHHPPHHHHPHHHPPHHPPHPH HHPHPHHPPHHPPPPHPPPPPPPHH	32
2	HHHHPHHPHHHHHHPPHPPHHPPH PPPPPPHPPHPPPHPPHHPPHHHPH	34
3	PHPHHPHHHHHHPPHPHPPHPHH PHPHPPPHPPHHPPHHPPHPHPPHP	34
4	PHPHHPPHPHHHPPHHPHHPPPH HHHHPPHPHHPHPHPPPPHPPHPHP	33
5	PPHPPPHPHHHHPPHHHHPHHPH HHPPHPHPHPPHPPPPPPHHPHHPH	32
6	HHHPPPHHPHPHHPHHPHHPHPP PPPPPHPHPPHPPPHPPHHHHHHPH	32
7	PHPPPPHPHHHPHPHHHHPHHPH HPPPHPHPPPHHHPPHHPPHHPPPH	32
8	PHHPHHHPHHHHPPHHHPPPPPP HPHHPPHHPHPPPHHPHPHPHHPPP	31
9	PHPHPPPPHPHPHPPHPHHHHHH PPHHHPHPPHPHHPPHPHHHPPPPH	34
10	PHHPPPPPPHHPPPHHHPHPPHP HHPPHPPHPPHHPPHHHHHHHPPHH	33

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 playout 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

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

These results are displayed in minutes and were obtained on a 3.50GHz Intel core i5-6600K CPU.

Our LNMCS performed very poorly on molecule 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 (Hsu and Grassberger 2011), 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.

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 (referred as "the molecule" in this section), the results presented in this section are only to give an idea of these algorithms performances and do not necessarily reflect their potentials on the HP model.

Nested Monte Carlo Search The good performances of the NMCS (Cazenave 2009) compared to the other algorithms presented in this section is what decided us 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.

As you can see on these figures, 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 (about an order of magnitude or two), being able to reach them 4 times as often is a great improvement.

Nested Rollout Policy Adaptation The NRPA (Rosin 2011) is similar to the NMCS, the main difference is that

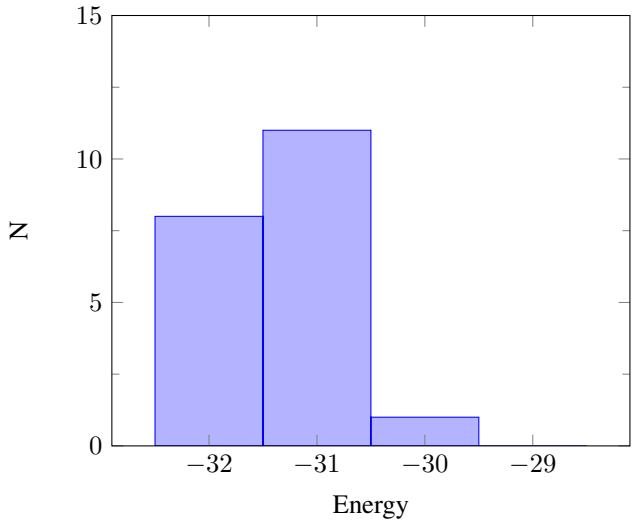

Figure 1: Energy distribution with the Lazy NMCS

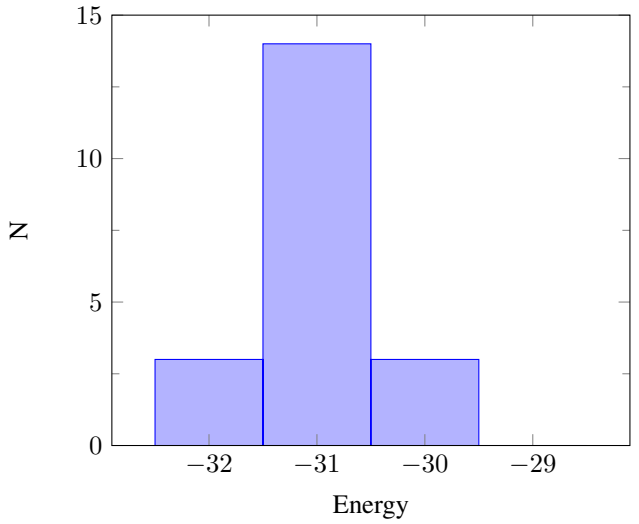

Figure 2: Energy distribution with the NMCS

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 man made policy (like we did here). On many problems the NRPA outperforms the NMCS, however in our case it was not able to do. The performance of the NRPA and GNRPA (Cazenave 2020) are widely dependent on the move representations, here it is the number corresponding to the amount 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 (Doran and Michie 1966) 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 children 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 (Kocsis and Szepesvári 2006) is used on game playing and usually provides good results. It iteratively starts from the initial state and launches playouts, based on the results of each playout, the value used to determine which moves should be taken is 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.

Conclusion

While the Lazy NMCS algorithm likely does not outperform the state of the art algorithms, it has the advantage to be easier to implement and may be applicable to more problems. It is also shown to be an improvement on the NMCS algorithm on this specific problem. In future works we aim to apply it to other problem and find ways to improve its performances.

References

Cazenave, T. 2009. Nested Monte-Carlo Search. In Boutilier, C., ed., *IJCAI*, 456–461.

Cazenave, T. 2020. Generalized Nested Rollout Policy Adaptation. In *Monte Carlo Search at IJCAI*.

Dill, K. A. 1985. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6): 1501–1509.

Doran, J. E.; and Michie, D. 1966. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437): 235–259.

Grassberger, P. ???? Pruned-enriched Rosenbluth method: Simulations of polymers of chain length up to 1 000 000.

Gros, F.; Gilbert, W.; Hiatt, H. H.; Attardi, G.; Spahr, P. F.; and Watson, J. D. 1961. Molecular and Biological Characterization of Messenger RNA. *Cold Spring Harbor Symposia on Quantitative Biology*, 26: 111–132. Publisher: Cold Spring Harbor Laboratory Press.

Hsu, H.-P.; and Grassberger, P. 2011. A review of Monte Carlo simulations of polymers with PERM. *Journal of Statistical Physics*, 144(3): 597–637.

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.; and Hassabis, D. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873): 583–589.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, volume 4212, 282–293. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-45375-8 978-3-540-46056-5. Series Title: Lecture Notes in Computer Science.

Rosin, C. D. 2011. Nested Rollout Policy Adaptation for Monte Carlo Tree Search. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 649–654.

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.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489.

Thachuk, C.; Shmygelska, A.; and Hoos, H. H. 2007. A replica exchange Monte Carlo algorithm for protein folding in the HP model. *BMC Bioinformatics*, 8(1): 342.

Wüst, T.; and Landau, D. P. 2012. 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.