Real-Time Dynamic Programming Applied to *De Novo* Genome Assembly

Srihari Ganesh [1]

Abstract

The DNA fragment assembly problem is an NP-complete problem for which solutions are required in every field of biology. The development of assemblers is an active area of research. Exploratory work has previously been done to attempt to apply reinforcement learning to the fragment assembly problem through Q-learning on an episodic Markov Decision Process (MDP), though issues with scalability were noted. We build upon this work by applying the Real-Time Dynamic Programming (RTDP) algorithm and reformulating the MDP. Simulated results show that RTDP is an improvement over Q-learning, though the MDP formulation has ambiguous results. The experiments also re-emphasize the concerns with using MDP approaches in real-world applications.

1. Introduction

DNA sequencing — finding the order in which the nucleotide bases (each of which is represented by one of the symbols A, C, G, T) that make up the sequence appear in DNA — is vital to biological research and nearly every possible application in medicine and biotechnology. However, current biological sequencing technology does not have the ability to read off entire DNA strands (which can be hundreds of millions of bases). Instead, millions of small fragments (called reads) can be sequenced; these fragments are typically hundreds or thousands of bases long. The reads must then be computationally assembled based on overlaps between reads to recover the true DNA sequences (the genome).

This is the referred to as the fragment assembly problem, and is NP-complete. In addition to the challenges presented by the sheer magnitude of reads and sequence length, biology and experimental techniques introduce other complicating factors. For one, DNA is made up of complementary strands in opposite orientations, so we do not always know the orientation of a read. Other problems include errors in reads, mutations in the sequence, and repeated sequences that occur in multiple locations in the genome.

There are two main classes of DNA fragment assembly problem. In reference-based problems, one uses previously obtained sequences (which are expected to be similar to the sequence being analyzed) to align reads and guide reconstruction. In the *de novo* case, no such reference exists: instead, the genome must reconstructed together entirely from scratch.

1.1. Contributions

This paper explores potential improvements to reinforcement learning (RL) approaches to the *de novo* fragment assembly problem, specifically those using a Markov Decision Process (MDP) model. As will be discussed in Section 2.4, existing work in this area has only applied variants of the ε-greedy Q-learning (εQL) algorithm (Bocicor et al., 2011; Czibula et al., 2013; Padovani et al., 2021; Xavier et al., 2019). We implement a heuristic search algorithm, Real-Time Dynamic Programming (RTDP), with several simple choices of heuristic function (Barto et al., 1995). On small test sets, we show that RTDP dominates εQL. However, concerns with the scalability of the approach (in run-time and memory) reinforce the conclusion that an MDP approach is likely inviable for real-world applications (Xavier et al., 2019).

2. Related Work

Given the complexity of the fragment assembly problem and the amount of reads involved, finding an exact solution is intractable. Instead, heuristic and approximating methods must be used. Given the possibility for errors and discontinuous DNA sequences, any real-world *de novo* approach attempts to assemble the reads into as many large supersequences (contigs) as possible.

2.1. De Bruijn Graphs

Currently, the most popular methods for *de novo* assembly utilize De Bruijn graphs (Pevzner et al., 2001). For each read, this approach finds every subsequence of some prespecified length k and creates a vertex for that k-mer. The k-mers with significant overlap are connected with edges.

[1]Harvard College, Harvard University, Cambridge, MA 02138.

Proceedings of the 2022 STAT 234 Project, Cambridge, Massachusetts, USA, 2022.

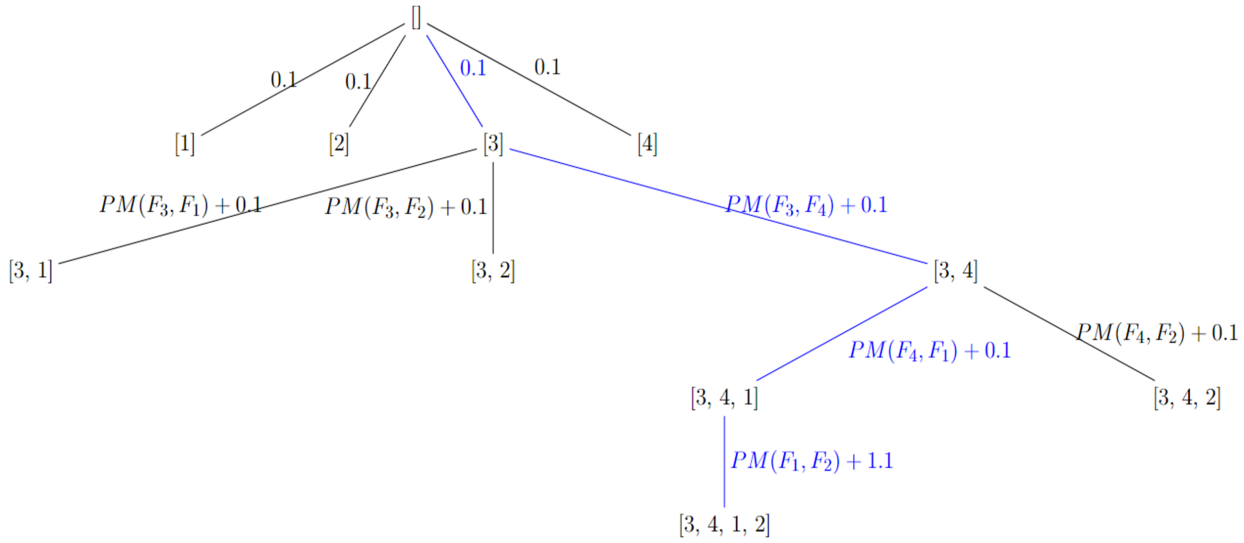

Figure 1. MDP formulation for the fragment assembly problem in the literature (list representation), using an example genome with four reads $[F_1, F_2, F_3, F_4]$ (Bocicor et al., 2011; Padovani et al., 2021).

Then, a solution (solved genome) is given by searching for a Eulerian path through the graph.

2.2. Genetic Algorithms

Genetic algorithms (GA) have also been explored for fragment assembly (Kikuchi & Chakraborty, 2012; Oliviera et al., 2017). These approaches are inspired by natural selection, where generations of solutions are created, with children formed by combining and mutating parents. In each generation, the most fit solutions are selected to reproduce for the next generation. GA builds on the initial greedy approaches in DNA fragment assembly, addressing the issue that greedy algorithms would work towards local maxima, and not the global extremum. These approaches still struggle on large read sets in comparison to De Bruijn graphs, but still may have some utility in smaller genomes.

2.3. Scoring Functions

For GA methods (or generally most iterative algorithms), one must score the quality of a solution without actually knowing the solution beforehand. To define such scoring methods, let one must first define the structure of a solution. As an output, a solution is given as a permutation of reads: that is, given reads $[F_1, F_2, ..., F_n]$, a solution is some permutation of the fragments $[F_{a_1}, F_{a_2}, ..., F_{a_n}]$. One popular class of scoring systems uses pairwise overlaps: for some overlap measure PM, the score S of a permutation is

$$S([F_{a_1}, F_{a_2}, ..., F_{a_n}]) = \sum_{i=1}^{n-1} PM(F_{a_i}, F_{a_{i+1}}). \quad (1)$$

Further on in the paper, we will overload PM to such that $S([F_{a_1}, F_{a_2}, ..., F_{a_n}]) = PM([F_{a_1}, F_{a_2}, ..., F_{a_n}])$. Possible choices for the scoring system include the Smith-Waterman algorithm and the Levenschtein distance (Oliviera et al., 2017; Smith & Waterman, 1981). In an idealized, error-free problem, a simple count of the maximum number of overlapping bases between the tail of the first read and the head of the second can be used.

2.4. Reinforcement Learning

Reinforcement learning (RL) methods have not been widely explored in the DNA fragment assembly, and for good reason — the goal of the fragment assembly problem is optimization (finding the true arrangement of fragments), while a typical RL problem involves some trade-off between exploration and exploitation. Ant colony optimization for fragment assembly was introduced but not explored further (Wetcharaporn et al., 2006).

Formulation of the fragment assembly problem as an episodic Markov Decision Process (MDP) has been briefly explored, and will be expanded upon in this paper (Bocicor et al., 2011; Czibula et al., 2013; Padovani et al., 2021; Xavier et al., 2019). However, the existing literature is very limited and serves as initial exploration: only ε-greedy Q-learning has been studied, with the intention of improving accuracy with more thorough searching. A hybrid Q-learning/GA was the only other method attempted. In addition, testing has only been done on small, simulated genomes and read sets (orders of magnitude smaller than true data), with error-free, correctly-oriented reads.

An existing GA method was shown to vastly outperform the ε-greedy Q-learning approaches (including the hybrid method) previously proposed, which led to the conclusion that an MDP formulation was unlikely to be the basis of a tractable, real-world approach (Oliviera et al., 2017; Padovani et al., 2021). We will address this thought briefly in the conclusion (Section 6.4); however, this paper will aim to introduce more appropriate MDP approaches in order to make any comparisons more legitimate.

3. MDP Formulation

We will now define the parameters of the episodic Markov Decision Process (MDP) as established in previous works (Bocicor et al., 2011; Padovani et al., 2021). We will first introduce the MDP verbally, then formally notate it. Figure 1 is a graphical representation of the MDP formulation for a sample genome with only four reads.

The MDP formulation supposes that we are given n correctly oriented, error-free reads, and that we are trying to find the optimal permutation of these reads (i.e. find the ordering of the reads that gives the true genome when aligned). At each time-step, we take an action, which is one of the n reads. The state is used for book-keeping: it is a list of actions that we have already taken. With that state representation, we start with the empty list $[]$, and append each action taken to the list, which clearly means that the transitions are known and deterministic. Terminal states occur when all reads have been used once, and to form valid states, no read is allowed to repeat — thus, episodes must strictly consist of a permutation of the action space. Rewards are linear functions of the pairwise metrics as discussed in Section 2.3: for example, if the first two actions taken are F_4, F_1, then the reward for the second action is related to $PM(F_4, F_1)$.

3.1. Formal Definition

Let us now explicitly define the structure of the MDP given n correctly-oriented, error-free reads:

- **State space:** permutations of the reads of up to length n, represented as lists.
  - **Initial state:** the empty list, $[]$.
  - **Terminal states:** lists of length n, where the only reachable states are (non-repeating) permutations of the reads.
- **Action space:** any of the n actions not in the current state
  - **Deterministic transitions:** define the function T as the deterministic transition function, returning the next state given the current state s and action taken a. Then

$$T(s, a) = s + [a], \tag{2}$$

where the $+$ sign indicates the appending of a to the list s.

- **Reward function:** given a pairwise overlap function PM,

$$R(s, a) = \begin{cases} 0.1 & s = [] \\ PM(s[-1], a) + 1.1 & s \text{ is terminal} \\ PM(s[-1], a) + 0.1 & \text{otherwise,} \end{cases} \tag{3}$$

with $s[-1]$ representing the last element in s (and thus the last action taken). In the prior RL works, the Smith-Waterman algorithm and a simple overlap measurement have both been used. Since we are assuming that we have error-free reads, we use the normalized version of the latter and let PM explicitly refer to this measurement from here on. If we let $tail(x, i)$ refer to the last i bases in sequence x and let $head(x, i)$ refer the the first i, then we can define the simple overlap function.

Definition 3.1. Simple overlap function.

$$PM(x, y) \propto \max_i \text{such that } tail(x, i) = head(y, i)$$

3.2. Important Specifics of the MDP Formulation

We will now emphasize some particularities about this formulation of the DNA fragment assembly problem, especially compared to typical RL problems involving MDPs.

- **Goal:** maximizing the sum of undiscounted rewards in any single episode encountered, which translates to visiting the optimal terminal state at some point (since this corresponds to an exact ordered list of actions). This is an optimization problem, which deviates from the common RL goal of maximizing the cumulative reward across all episodes (and the ubiquitous exploration-exploitation trade-off that comes with it). With the optimization goal, visiting the optimal terminal state once is always more desired than visiting a near-optimal state many times without ever reaching the optimum.
- **One-off optimization:** The ordering learned for one set of reads cannot be used to learn how to order other sets of reads. Each instance of the problem (set of reads) only pertains to itself and its own MDP.
- **Fully known and deterministic MDP:** the states, actions, and reward function are all known, deterministic, and observable. There is no randomness and no constraints on observability.

Algorithm 1 RTDP

Input: fragments $[F_1, F_2, ..., F_n]$, updatable heuristic H, reward function R, transition function T, Bellman operator B
repeat
  $s_{history} = ()$
  $s = []$
  while s is not terminal **do**
    $s_{history}$.append(s)
    $a = \arg\max_{a' \notin s}[R(s, a') + H(T(s, a'))]$
    $s = T(s, a)$
  end while
  for s' in $s_{history}[:: -1]$ (in reverse) **do**
    $H(s') = BH(s')$
  end for
until stopping criterion (# of episodes)

- **Massive state space:** The size of the state space is

$$\sum_{i=0}^{n} \binom{n}{i} i! \approx n! \left(e - \frac{1}{n+1}\right) = \Omega(n!)$$

4. Proposed Improvements

4.1. Real-Time Dynamic Programming

Definition 4.1. Bellman update of state s, where H is an optimistic (upper bound) heuristic function for the value of the state, while B is the Bellman operator which

$$BH(s) = \max_{a \notin s}[R(s, a) + H(T(s, a))]$$

As previously noted, ε-greedy Q-learning was the only RL approach tested in the literature. However, Q-learning struggles to propogate state-action values through the state space, which can be problematic given that one of the main challenges of the DNA fragment assembly problem is the size of the state space. Additionally, Q-learning is designed to address the exploration-exploitation trade-off in classical RL problems, but since there is no such trade-off involved here, it is ill-suited for the problem. Instead, we can implement an algorithm from the literature which (in the best case) has the ability to converge without visiting the entire state space, while also aiming for optimization: real-time dynamic programming (RTDP) (Barto et al., 1995).

RTDP is a heuristic search, where an upper bound of the state values is maintained. At each step, actions are selected greedily, the values of the visited states in an episode are backed up in reverse after the episode using the Bellman operator (see Algorithm 1, Definition 4.1). This backing up allows the upper bounds to be fully updated to match our knowledge during each episode, instead of requiring multiple visits and incremental updates like Q-learning. RTDP also avoids unnecessary exploitation - a terminal state s is only revisited if it is known to be the best.

4.1.1. Choice of RTDP Heuristic

Another advantage of RTDP is that it allows us to apply prior knowledge to our search through the heuristic. However, unlike search problems in Euclidean space, there is no obvious heuristic for state values in the DNA fragment assembly problem. As a preliminary study, we propose some simple heuristic functions, with the acknowledgement that these are likely far from optimal and can likely be improved given domain knowledge.

1. **Degenerate heuristic:** based strictly on the number of actions that have been taken, with no consideration for which actions. This is completely uninformative and is meant to serve as the baseline/control.
2. **Naive maximizing heuristic:** at the beginning of the algorithm, the maximum overlap that each read has with any other read is calculated. The heuristic of a state is based on the sum of these maximum overlaps of the remaining possible actions (in addition to the last action taken so far).
3. **Informed maximizing heuristic:** at each step in an episode, the maximum overlap that each of the remaining possible actions has with the rest of the remaining possible actions is calculated. The heuristic of a state is based on the sum of these maximum overlaps of the remaining possible actions remaining (in addition to the last action taken so far)

The heuristics are listed in increasing order of complexity and information, which comes with the trade-off that greater run-time and memory are required to calculate and store more complicated heuristics.

4.2. Tuple State Representation

The existing state representation in the MDP — which we will refer to as the list representation — makes the MDP a tree, since each state can only be reached through the exact order of actions that the state dictates. Intuitively, this seems to be wasteful, since the agent would be unable to re-use learning about suffixes. For example, in an environment with 5 reads (F_1 through F_5), suppose that the agent took actions in the order F_3, F_4, F_1 in one episode, and F_4, F_3, F_1 in a later one. From the prior episode, the agent has already learned something about the value of taking the remaining available actions, F_2 and F_5. However, since they took different paths to get there, they cannot use that knowledge.

To address this, we propose a state representation using tu-

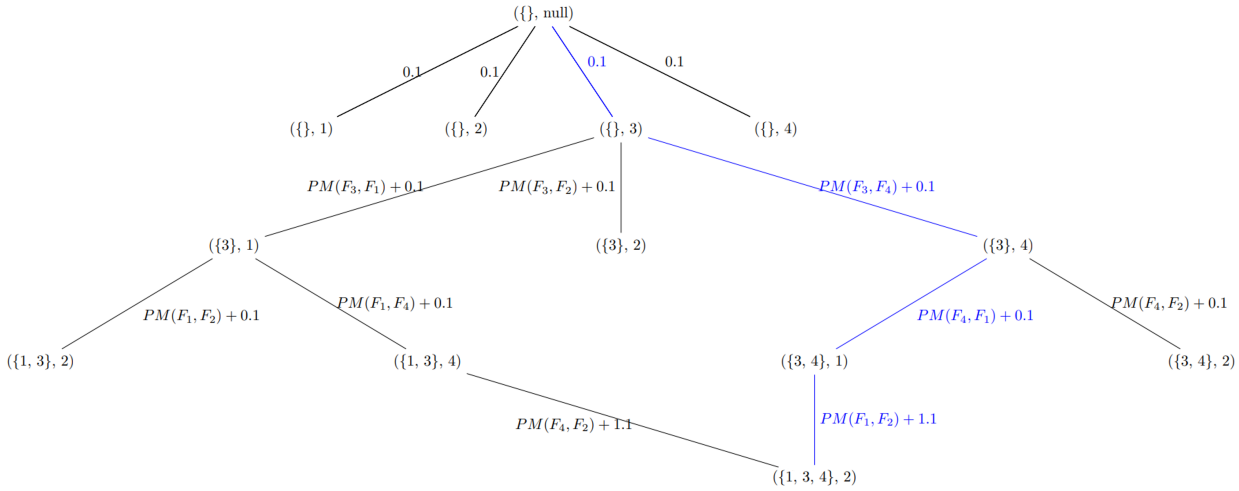

Figure 2. Proposed re-formulation of the fragment assembly MDP with a tuple state representation.

ples, which we will refer to as the tuple representation for reasons soon to be clear. In this tuple, the first entry is a set of all previously taken actions, while the second entry is the latest action taken. For example, the lists $[F_3, F_4, F_1, F_2]$ and $[F_1, F_3, F_4, F_2]$ can both be represented in the tuple representation by $(\{F_1, F_3, F_4\}, F_2)$. We propose that this representation should allow for the re-use of learning about suffixes, which would ideally improve the efficiency of learning.

5. Simulated Experiments

Experiments were run to evaluate the quality of the proposed improvements. The testbeds were the five largest simulated microgenomes used in prior MDP genome-assembly literature (with error-free, correctly-oriented reads), whose specifications are listed in Table 1 (Padovani et al., 2021). Four algorithms were tested, with the acronyms that will be used to refer to them: ε-greedy Q-learning (εQL), with the same hyperparameters used in prior literature, degenerate heuristic RTDP (dRTDP), naive maximizing heuristic RTDP (nmRTDP), and informed maximizing heuristic RTDP (imRTDP) (Padovani et al., 2021). Two MDP formulations were tested: the list representation and the proposed tuple representation. Each combination of algorithm and MDP formulation was tested on each of the five testbeds and averaged across three trials on each testbed. Trials were intended to be thirty minutes long, but the episode occurring at the thirty minute mark was allowed to complete. For these experiments we also define **relative PM**, which is the ratio of a solution's PM to that the PM of the true sequence.

The genetic algorithm GAVGA was used for comparison in prior RL works, where it was shown to dominate ε-greedy Q-learning (Oliviera et al., 2017; Padovani et al., 2021). Use of GAVGA was attempted to serve as a benchmark for this paper; however, for unclear reasons (though likely a combination of our error and improper documentation for user parameters), the results of Padovani et al. were unable to be replicated. Instead, GAVGA sometimes gave poorer results than previously documented, and was often unable to consolidate the genome into one contig. Since the performance of GAVGA was worse than previously reported, rigorous comparisons will not be made. However, we will briefly reference anecdotal observations from the failed trials in the conclusion 6.4.

Table 1. Testbed Specifications (Padovani et al., 2021). All testbeds were simulated with correctly-oriented error-free reads, each of which are 75 bases long. Testbeds will be referred to by their IDs.

TESTBED ID	GENOME LENGTH (BASES)	# OF READS
381_20_75	381	20
567_30_75	567	30
726_40_75	726	40
930_50_75	930	50
4224_230_75	4224	230

6. Results and Discussion

6.1. RTDP Dominates Q-Learning in Less Episodes

RTDP (all heuristics) consistently dominated Q-learning on all microgenomes tested. RTDP was both faster and more accurate: a representative average trace is shown in Figure

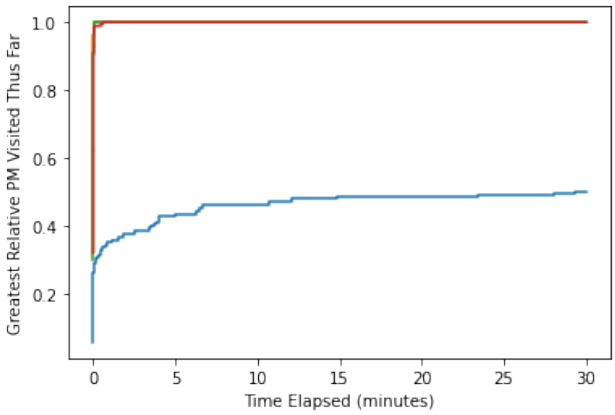

Figure 3. Best relative PM visited by the four algorithms on the list representation of 930_50_75 over time. Traces represent averages across three trials for εQL (blue), dRTDP (orange), nmRTDP (green), and imRTDP (red). The RTDP traces are visually indistinguishable due to their rapid success in comparison to Q-learning on the testbeds.

Table 2. Average Solution PM Achieved on List Representation. Best relative PM achieved by each of the algorithms on each testbed using the list representation, averaged across three trials.

TESTBED	εQL	DRTDP	NMRTDP	IMRTDP
381_20_75	0.793	1.0	1.0	1.0
567_30_75	0.696	1.0	1.0	1.0
726_40_75	0.568	1.0	1.0	1.0
930_50_75	0.500	1.0	1.0	1.0
4224_230_75	0.171	0.981	0.983	0.974

3, while average solution PMs for all testbeds with the list representation are displayed in Table 2. As shown in Figure 4, RTDP runs far less episodes than Q-learning, but the time taken to calculate the heuristics appeared to pay dividends in rapid and reliable solution quality.

However, one should note from Figure 3 that these solutions are found in an extremely low number of episodes. This likely stems from the nature of the testbeds — as idealized microgenomes, there is a high degree of overlap, and thus greedy actions may be highly rewarded. Thus, on top of the other issues with using simulated data, these results should be taken with reservations and verified on more difficult datasets.

6.2. RTDP Heuristics Only Differ in Run-Time

As noted in Figure 3, the choice of heuristic function affected the number of episodes which can be run in a set time span: namely, imRTDP took significantly more time than the other two heuristics. However, with read set sizes, this

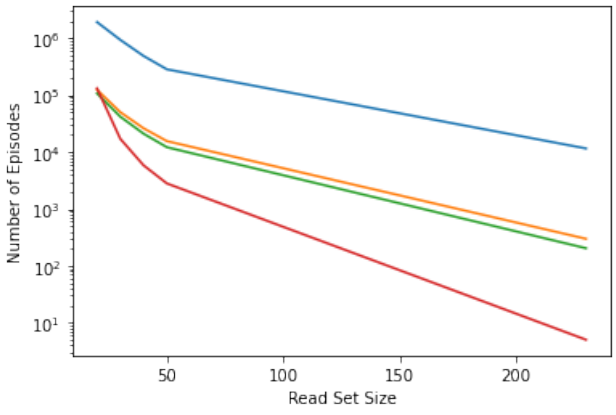

Figure 4. Number of Episodes in 30 minutes as a function of read size. Lines represent averages across three trials for εQL (blue), dRTDP (orange), nmRTDP (green), and imRTDP (red).

small there does not appear to be a significant difference in the number of episodes required for each of the RTDP algorithms to visit the optimal PM for the first time, as can be seen in Figure 5. This implies that the seemingly more informative heuristics are not actually contributing anything to the efficiency of the searching.

Such an effect could stem from the fact that even the more informative heuristics do not actually differentiate between the available actions at a specific state. Suppose we are in state s. Then every possible next state s' (those which can be visited with some valid action a) still has the same set of actions that are considered by each heuristic. Thus, the heuristics can only contribute multiple steps down the line, which blocks information from passing through without a look-ahead algorithm.

Given that intuition, it seems unreasonable to seriously consider the more complicated maximizing heuristics, nmRTDP and imRTDP, with no tangible performance bonuses. This is especially important since heuristic calculations take at least quadratic time, which would scale very poorly to read set sizes in the millions.

6.3. Tuple Representation Improves ε-Greedy Q-Learning, But Has No Notable Effect on RTDP

The implementation of the tuple representation improved the quality of the best state that Q-learning visits on all testbeds (over the list representation), as can be seen in Table 3. The trace for the testbed in which the list representation came closest, 726_50_75, is shown in Figure 6. These results provide some support for the initial argument for the tuple representation: the Q-values could be re-used, making the learning of suffixes of states more useful than it would have

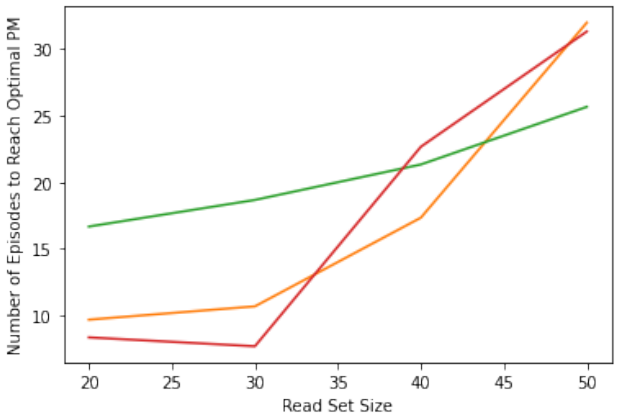

Figure 5. Number of episodes for each RTDP implementation to reach the optimal PM as a function of the size of the read set. 4224_230_75 is excluded since none of the algorithms reached the optimal PM. Traces represent averages across three trials for dRTDP (orange), nmRTDP (green), and imRTDP (red).

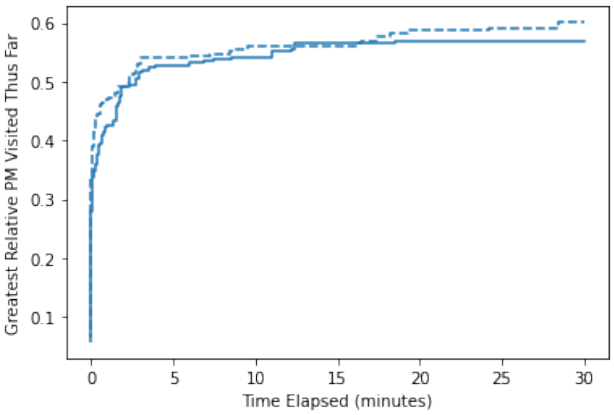

Figure 6. Best relative PM visited by Q-Learning using the list representation (solid) and the tuple representation (dashed) on the 726_40_75 testbed over time. Traces represent averages across three trials. The testbed was chosen since it showed the instance in which the list representation was the closest to defeating the tuple representation.

Table 3. Average Solution PM Achieved by ε-Greedy Q-Learning with respect to State Representation. Solution PMs are averaged over three trials.

TESTBED	LIST	TUPLE
381_20_75	0.793	0.895
567_30_75	0.696	0.744
726_40_75	0.568	0.601
930_50_75	0.500	0.530
4224_230_75	0.171	0.176

been in a tree.

However, no such consistent difference was brought about by the change in state representation when running any of the RTDP implementations. Several possible explanations exist. For one, RTDP explores the state space thoroughly, meaning that the states it visits may have a minimal amount of shared subsequences, especially in comparison to the amount of repetition that is involved in ε-greedy Q-learning. Additionally, the nature in which updates occur in RTDP could also play a factor: only the visited states in each episode are updated, so in a tuple representation, states whose Bellman-updated values have theoretically changed may not be getting updated to reflect those changes, thus nullifying a possible improvement that the tuple representation may have been able to offer. In a way, this update rule may still be mimicking the behavior of a tree, like in the list representation.

6.4. Conclusion

To summarize, RTDP outperformed Q-learning in both speed and accuracy on all testbeds used. Of the three tested, the choice of heuristic in RTDP did not make a significant difference when measuring speed by the number of episodes, but the run-time to calculate heuristics led to the most complicated heuristic, imRTDP, being less favorable. Additionally, while the tuple representation was more favorable for Q-learning, no notable difference was observed in the performance of any form of RTDP across state representations on any of the given testbeds.

Given that the attempted choices heuristic did not actively improve learning — as evidenced by the degenerate heuristic keeping pace with the more intelligently designed versions — domain knowledge may unlock performance gains through an improved heuristic. However, the scalability of any such heuristic remains in question, since heuristics may have to sweep through all of the reads multiple times.

Scalability concerns naturally arise given the testbeds used, as well. As noted in the introduction (Section 1), the size of true biological genomes and read sets are orders of magnitude larger than the testbeds used here. It is also worth knowing that the reads were error-free, when this is a large obstacle in real-world science. However, in this MDP approach, the overlap measure PM can be defined without affecting the formulation of the problem; thus, any results about algorithms and state representations are still fairly general with respect to the PM function.

Another issue arises from the fact that the reads are ideal-

ized in an additional way: they are spread relatively evenly across the genome, with high amounts of overlap with their neighbors. This could serve as another explanation as to why RTDP perhaps performs unrealistically well on these testbeds — greedy actions may quite often be optimal. Because of this, it would still be useful to continue exploring other algorithms. RTDP is designed to be robust in stochastic and unknown environments, so finding algorithms that can take better advantage of fully known, deterministic MDPs — perhaps from the operations research literature — could also be a future direction of research if one wished to continue with the MDP formulation of fragment assembly.

Obviously, comparisons to GAVGA would have been ideal for consistency with prior work, but from the data it is unclear where they stand (Oliviera et al., 2017; Padovani et al., 2021). Anecdotally, GAVGA seemed to be terminating in seconds, which in could be similar to the speed at which dRTDP and nmRTDP reached some optima, but nothing greater can be drawn from these observations without testing.

There may be an inherent advantage to genetic algorithms (GAs) for the fragment assembly problem. In a massive state space, they anchor on good solutions in search of better ones, while episodic MDP formulations force a restart. On that line, in GAs, the mutations to solutions can occur anywhere, giving a greater flexibility in what is considered a neighbor; this stands in direct contrast to the top-down nature of episodic MDPs. It's possible that genetic algorithms could be a generalization of this MDP formulation, with edges connecting what we considered terminal states. The episodic MDP formulation introduces an extra restriction when there may not need to be one.

In general, it was already known that the MDP formulation was heavily flawed. With a massive amount of reads, one is forced to confront the curse of dimensionality given the ever-expanding state representation (Sutton & Barto, 2018); issues with thoroughly searching and memory would likely making scaling to real-world genomes difficult. Anecdotally, we observed over 10 gigabytes of memory being used to run RTDP these microgenomes, which is a troubling sign when biological data are orders of magnitude larger.

And thus, even with an marked improvement in searching the episodic MDP representation of the fragment assembly problem, it is unclear how much these contributions could scale. There is definitely work that can still be done to improve the MDP approaches that we have explored so far. However, the restricted nature of the episodic MDP formulation, and the dimensionality that accompanies it, may make this infeasible in practice.

Acknowledgements

Thank you to Eura Shin and Professor Susan Murphy for their invaluable feedback during the course of this project. Thank you as well to Dr. Raaz Dwivedi for giving feedback and revisions on both the poster and paper, including providing the calculation in Section 3.2. Additionally, thank you to all three for teaching STAT 234 in Spring 2022.

References

Barto, A., Bradtke, S., and Singh, S. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1–2):81–138, 1995.

Bocicor, M., Czibula, G., and Czibula, I. A Reinforcement Learning Approach for Solving the Fragment Assembly Problem. In *2011 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 191–198, 2011.

Czibula, G., Czibula, I., and Bocicor, M. A Comparison of Reinforcement Learning Based Models for the DNA Fragment Assembly Problem. *Studia Universitatis Babes–Bolyai, Informatica*, 58(2):90–102, 2013.

Kikuchi, S. and Chakraborty, G. An Efficient Genome Fragment Assembling Using GA with Neighborhood Aware Fitness Function. *Applied Computational Intelligence and Soft Computing*, 2012.

Oliviera, R., Damasceno, F., Souza, R., Santos, R., Lima, M., Kawasaki, R., and Sales, C. GAVGA A Genetic Algorithm for Viral Genome Assembly. In *EPIA Conference on Artificial Intelligence 2017*, pp. 395–407, 2017.

Padovani, K., Xavier, R., Carvalho, A., Reali, A., Chateau, A., and Alves, R. A step towards a reinforcement learning de novo genome assembler. *arXiv*, 2021.

Pevzner, P., Tang, H., and Waterman, M. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748–9753, 2001.

Smith, T. and Waterman, M. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

Sutton, R. and Barto, A. (eds.). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

Wetcharaporn, W., Chaiyaratana, N., and Tongsima, S. DNA Fragment Assembly: An Ant Colony System Approach. In *EvoWorkshops 2006: Applications of Evolutionary Computing*, pp. 231–242, 2006.

Xavier, R., Padovani, K., Chateau, A., and Alves, R. Genome Assembly Using Reinforcement Learning. In *Brazilian Symposium on Bioinformatics 2019: Advances in Bioinformatics and Computational Biology*, pp. 16–28. Springer, Cham, 2019.