A step toward a reinforcement learning *de novo* genome assembler

Kleber Padovani\(^1\)*, Roberto Xavier\(^2\)**, Rafael Cabral Borges\(^2\)**, André Carlos Carvalho\(^2\)**, Anna Reali\(^4\)**, Annie Chateau\(^5\)**, Ronnie Alves\(^2,6\)**

\[^1\] Center for Higher Studies of Itacoatiara, University of the State of Amazonas, Itacoatiara, Amazonas, Brazil
\[^2\] Sustainable Development, Vale Technology Institute, Belém, Pará, Brazil
\[^3\] Institute of Mathematics and Computer Sciences, University of São Paulo, São Carlos, São Paulo, Brazil
\[^4\] Polytechnic School, University of São Paulo, São Paulo, São Paulo, Brazil
\[^5\] LIRMM, University of Montpellier, CNRS, Montpellier, France
\[^6\] Institute of Natural Sciences, Federal University of Pará, Belém, Pará Brazil

\(*\) Role(s): Conceptualization, Data Curation, Investigation
\(**\) Role(s): Formal Analysis, Project Administration.
\(**\) Role(s): Methodology
\(**\) Role(s): Validation
\(**\) Writing – Original Draft Preparation
\(*\) Role(s): Writing – Review & Editing
* kleber.padovani@gmail.com (KP)

**Abstract**

*De novo* genome assembly is a relevant but computationally complex task in genomics. Although *de novo* assemblers have been used successfully in several genomics projects, there is still no ‘best assembler’, and the choice and setup of assemblers still rely on bioinformatics experts. Thus, as with other computationally complex problems, machine learning may emerge as an alternative (or complementary) way for developing more accurate and automated assemblers. Reinforcement learning has proven promising for solving complex activities without supervision – such games – and there is a pressing need to understand the limits of this approach to ‘real’ problems, such as the DFA problem. This study aimed to shed light on the application of machine learning, using reinforcement learning (RL), in genome assembly. We expanded upon the sole previous approach found in the literature to solve this problem by carefully exploring the learning aspects of the proposed intelligent agent, which uses the Q-learning algorithm, and we provided insights for the next steps of automated genome assembly development. We improved the reward system and optimized the exploration of the state space based on pruning and in collaboration with evolutionary computing. We tested the new approaches on 23 new larger environments, which are all available on the internet. Our results suggest consistent performance progress; however, we also found limitations, especially concerning the high dimensionality of state and action spaces. Finally, we discuss paths for achieving efficient and automated genome assembly in real scenarios considering successful RL applications — including deep reinforcement learning.
Introduction

The genome of an organism is the sequence of all nucleotides from its DNA molecules. Each isolated nucleotide represents no relevant biological information, and within the organism’s genome there are species genes, that define species traits and behaviors (e.g., eye color) [1]. A single DNA fragment cannot represent the complete information from a gene, and genome assembly is the computational task used to order the sequenced DNA fragments (i.e., reads) and reconstruct the original DNA sequence [2]. The size and number of reads directly impact the complexity of the assembly process, and illuminating this bottleneck problem has become an important bioinformatics problem for producing a fast, automated and reliable solution.

Genome assemblers adopt comparative and/or de novo strategies. A comparative strategy (unlike de novo strategies) requires a reference genome [3]. De novo strategies are particularly important given that only a small number of reference genomes are currently available [4]. However, this approach is a highly complex combinatorial problem that falls into the theoretically intractable class of computational problems (NP-hard) [5]. De novo assemblers (commonly based on heuristics and graphs) can produce acceptable solutions, but require specific bioinformatics knowledge for configuration and parameter setting, and optimal results are not guaranteed [6].

Genome assembly is currently not a fully solved problem. Nevertheless, few approaches have been applied machine learning trying to achieve better solutions for the assembly problem [7, 8]). With current access to increased processing and storage power, machine learning application started to increase and good results have been reported [9]. This increase also allowed the return of reinforcement learning application for these problems [10].

Reinforcement learning (RL) is a basic machine learning paradigm in which intelligent agents take actions in a task environment. Ideally, this agent solves the task when it is able to learn how to maximize the rewards from its actions [11]. Such low use of RL in real-world problems has also been observed in the particular instance of genome assembly, where few studies was found [7, 12–14].

The seminal approach [12] proposes an episodic trained agent (i.e., whose training has been divided into episodes) applying the Q-learning reinforcement learning algorithm, which allows the agent to learn through the consequences (positive or negative rewards) it receives after taking actions. Obtaining intelligent and trained agents by RL using the seminal approach is important because it could eliminate the need for human specialists.

The Q-learning algorithm requires a Markov decision process definition with established parameters of states and actions, together with a reward system to be achieved by the agent at each action in every state [11]. The problem was then modeled through a state space capable of representing all possible read permutations, with one action for each read in each state [12]. Following these definitions, from graph theory, the proposed state space for n reads can be visualized as a complete n-ary tree, with height equal to n, as the set of states presents one initial state and forms a connected and acyclic graph [15]. The number of existing states in the state space is represented by Eq 1.

\[
\text{number of states} = \frac{n^{n+1} - 1}{n - 1}
\]

The reward system depends on the type of state reached after each action (absorbing or nonabsorbing). An absorbing state does not allow any other state [11,16]. Each state requiring n actions to be reached (with n being the number of reads) is an absorbing state. A small and constant reward (i.e., 0.1) is assigned for actions reaching nonabsorbing states, including actions leading to absorbing states.
achieved by repeated reads. Finally, actions leading to other absorbing states (the final states) produce a reward corresponding to the sum of overlaps between all pairs of consecutive reads used to reach these states. Fig 1 presents a state space example for a set of 2 reads (A and B) with a single initial state, two actions associated with nonabsorbing states and four absorbing states (highlighted in black), achieved after taking two any actions.

Two absorbing states are highlighted (X). These are the final states, as they are reached directly without repeated actions. The Smith-Waterman algorithm (SW) was applied to obtain the overlaps between pairs of reads and added to obtain the rewards of actions leading to the final states. The sum of overlaps when reaching a final state $s$ (Performance Measure - PM), is described in Eq 2, where $read_s$ correspond to the sequence of reads associated with the actions for achieving $s$. In an optimal solution repeated reads overlap completely and pairs reach the maximum PM.

$$PM(s) = \sum_{i=1}^{n-1} sw(read_s[i], read_s[i+1])$$  \hspace{1cm} (2)

With these definitions the seminal approach produced positive results against two sets of 4 and 10 simulated reads less than 10 bp and 8 bp respectively. A scalability analysis was applied to evaluate the performance of this approach against 18 datasets produced following the same simulation methods. The initial set is one of the sets featured in the seminal approach, containing 10 reads with 8 bp extracted from a 25 bp microgenome. Seventeen new datasets were generated from this microgenome and also from a novel 50 bp microgenome (8 from the minor and 9 from the major microgenome) each containing 10, 20 or 30 reads, with 8 bp, 10 bp or 15 bp.

All previous definitions were replicated, but $\alpha$ and $\beta$ were set to 0.8 and 0.9, respectively, and the space of actions was reduced so that actions associated with previously taken reads were removed from the available actions. In the state space depicted in Fig 1, the leftmost and rightmost leaves (i.e., absorbing states) would be removed after this change. Although the number of states is reduced, the size of the state space grows exponentially, as described in Eq 3.

$$\text{number of states} = \sum_{i=0}^{n} \frac{n!}{(n-i)!}$$  \hspace{1cm} (3)

This confirmed positive results from the seminal approach with the first dataset; however, the performance decreased with increasing size, reaching the target microgenome in only 2 out of the 17 major datasets. This may be related to the high
cost required by the agent to explore a vast state space and to the failures in the reward system [13].

To investigate the application of reinforcement learning to genome assembly and address the current challenge of applying RL to real-world problems [18], we analyzed the limits of RL to the Genome Assembly problem, a key problem for scientific development. We corrected previously described issues, explored the performance of an improved reward system and added complementary strategies to be incorporated into the seminal approach to obtain improved and automated genome assemblies via machine learning.

Materials and methods

In this study, 7 experiments were evaluated against the seminal approach. The main goal was to reach an RL trained agent, to correctly identify the order of reads from a sequenced genome. Fig 2 illustrates this proposal, where the environment represents the set of reads to assemble; the agent interacts with the environment by taking actions intended to order the reads; for each action taken, the environment is updated and provides to the agent a corresponding reward; the agent learns from the reward received, and takes a new action, until reaching (ideally) the correct order of reads.

Fig 2. Illustration of the application of reinforcement learning to the genome assembly problem. The set of reads is represented computationally by a reinforcement learning environment. Through successive interactions with the environment, caused by taking actions, the agent ideally learns the correct order of reads — reaching the target genome.

The approaches produced here consider scalability analysis [13], with improvements made to the reward system — especially in Approaches 1 - and to optimize the agent’s exploration — approaches 2 and 3.

Approaches 1: Tackling sparse rewards

Approaches 1.1, 1.2, 1.3, 1.4 aimed to improve the reward system, given by Eq 4. Optimally, the agent achieves the correct order of reads upon learning the set of actions, specifically a permutation of reads, that maximizes the accumulated reward. Thus, the optimal actions – those leading to the anticipated permutation of reads – must always yield the highest cumulative reward. Nevertheless, this proposition may not hold consistently for the reward system proposed in the seminal approach, as it allows some nonoptimal actions to result in maximum accumulated rewards.

\[
r(s, a, s') = \begin{cases} P M(s') & \text{if } s' \text{ is a final state,} \\ 0.1 & \text{otherwise} \end{cases}
\]

(4)

This inconsistency (details in Section 6 of S1 Appendix in supporting information) stems from the sequences alignment with the Smith-Waterman algorithm (SW), which
calculates a score to represent major alignment size (even if partial), but has no constraint on the order between sequences. Thus the overlap score from the SW might induce the agent to find read permutations with high overlap values in pairs of reads without any suffix-prefix alignment. So that, using PM score as a reward for training may be ineffective for some datasets.

Thus, aiming to improve the agent’s performance, we adjusted the reward system through four approaches to explore two aspects: (a) the use of an overlap score that considers the relative order of reads and/or (b) the use of dense rewards. These new reward systems are presented in approaches 1.1, 1.2, 1.3 and 1.4.

As in the seminal approach, approach 1.1 defines that actions leading to the final states produce a bonus reward (of 1.0), added to another numerical overlap score between all subsequent reads used since the initial state. Thus, a reward corresponding to the sum of the normalized overlap score (ranging from 0 to 1) of each pair of reads was produced considering their relative order. Every action leading to nonfinal states produces constant and low rewards (0.1). Eq [5] formalizes the reward system for Approach 1.1, with \( PM_{\text{norm}}(s') \) representing the normalized overlap between the reads used to reach \( s' \) (details in Section 2 of Supporting Information).

\[
r(s, a, s') = \begin{cases} 
0.1 & \text{if } s' \text{ is a final state,} \\
PM_{\text{norm}}(s') + 1.0 & \text{otherwise}
\end{cases}
\]  

(5)

Despite the overlap score considering the order of reads in approach 1.1, it is susceptible to the sparse rewards problem — as in the seminal approach. Although it often produces small, constant and positive rewards, and not a zero-value reward as applied in sparse reward systems, only few and sparse state-action pairs would produce higher rewards. In both systems (Eq 4 and 5) no reward is provided during the learning process (since any read incorporated would produce a reward of 0.1).

Thus, the agent’s learning process depends exclusively on the sparse actions taken during the exploration of this state space, tending to take a long time due to the sparse rewards problem [19]. In approaches 1.2, 1.3 and 1.4, we focused on improving it with higher rewards distributed for each action taken in each episode (previously obtained only at the end of the episode). These approaches focused on reducing or eliminating inconsistencies that allowed permutations of unaligned reads to produce maximum accumulated rewards. Eq [6] and [7] represent the reward systems for approaches 1.2, 1.3 and 1.4, respectively, so that \( ol_{\text{norm}}(s, s') \) represents the normalized overlap between two subsequent reads.

\[
r(s, a, s') = PM_{\text{norm}}(s')
\]  

(6)

\[
r(s, a, s') = \begin{cases} 
PM_{\text{norm}}(s') + 1.0 & \text{if } s' \text{ is a final state,} \\
ol_{\text{norm}}(s, s') & \text{otherwise}
\end{cases}
\]  

(7)

\[
r(s, a, s') = \begin{cases} 
ol_{\text{norm}}(s, s') + 1.0 & \text{if } s' \text{ is a final state,} \\
ol_{\text{norm}}(s, s') & \text{otherwise}
\end{cases}
\]  

(8)

**Approach 2: Pruning-based elimination action**

To reduce the state space from the seminal approach, a heuristic procedure was applied to eliminate fully explored actions where the maximum cumulative reward achieved was smaller than the cumulative reward from taking any other action available. In Fig 3, looking at the changed state space as a tree — removing actions associated with used reads, we see 16 states, 6 are absorbing states and the final states (tree base). Note that 3 out of the 6 final states are highlighted in black, while the remaining states are highlighted in gray and white. The black states correspond to the
explored final states (i.e., visited by the agent). Gray states, such as the one reached by taking action \( a \) in the initial state, represent states where all children were fully visited during the learning process. White states (final or not) are those not yet explored and/or that have unexplored children — e.g., the initial state, where one child is not explored and the other one is partially explored.

When an unexplored final state is reached, such as the rightmost final state in Fig 3, the accumulated rewards are maintained and propagated for its predecessors, maintaining only the highest value propagated for the children. Each reward is represented by integer numbers within the states in the figure. In each nonfinal state the highest accumulated reward achieved during the training process is stored. Thus, it is possible to prune irrelevant actions, that do not produce the maximum accumulated reward (e.g., action \( a \) of the initial state in Fig 3). Note that all possible achievable states after taking this action were explored and the maximum cumulative reward was 6, while the initial state of action \( c \) alone produces a reward equal to 8.

When the agent first goes through the sequence of states corresponding to actions \( c, a \) and \( b \), the pruning mechanism propagates the maximum reward value up to the initial state and, at that moment, it cuts the action \( a \) from the initial state. The pseudocode presented in Algorithm 1 presents the procedure for updating the pruning process when the last explored final state (state) is reached obtaining the corresponding accumulated reward achieved (newReward).

![Fig 3. Illustration of the pruning procedure. State space corresponding to the assembly of 3 reads, referred by \( a, b \) and \( c \). The generic pruning procedure is defined in detail by Algorithm 1.](image)

**Algorithm 1** Pruning’s algorithm

```plaintext
1: procedure Prune(state : treeNode, newReward : float)
2: if state ≠ null and (state.unseen or newReward > state.maxReward) then
3:   state.unseen ← false
4:   state.maxReward ← newReward
5:   if state.final then \( \triangleright \) prune children where maxReward < newReward
6:     PruneUselessChildren(state)
7:   end if
8:   Prune(state.parent, newReward)
9: end if
10: end procedure
```

**Approaches 3: Evolutionary-based exploration**

In these approaches, we explore the potential for mutual collaboration between reinforcement learning and evolutionary computing — by applying the elitist selection
of the genetic algorithm [20, 21] — to optimize the exploration of the state space. The individual contributions of the genetic algorithm used in this hybrid proposal were divided into two approaches, 3.1 and 3.2.

**Approach 3.1: Evolutionary-aided reinforcement learning assembly**

Applying the $\epsilon$-greedy to expand the exploration of agents trained by the Q-learning algorithm allows a broader initial exploration, achieving optimal policy once the state space has been sufficiently explored [11]. However, the existing trade-off between exploitation and exploration remains a major problem for RL in high-dimensional environments [22, 23]. Here, for the first time, we introduce the interaction between RL and evolutionary computing into the exploration process based on the operation of the Q-learning algorithm. In each episode, the sequence of actions is stored, and at the end of the episode the sequence is transformed into a chromosome of an initial population, that evolves (see Fig. 4).

![Fig 4. Illustration of the proposed interaction between reinforcement learning (RL) and the genetic algorithm.](image)

At each RL episode, the actions taken by the agent are converted into the chromosome (each action as a gene) of an individual of the initial population of the genetic algorithm, whose size $n$ is predefined. After $n$ episodes ($n$ individuals in the initial population), this population evolves for a predefined number of generations through the genetic algorithm. Then, the most adapted individual of the last generation is obtained. In the end, that individual's chromosomal genes are used as actions in the next RL episode.

New chromosomes are inserted until the number of chromosomes reaches the predefined population size. At this point, agent training is interrupted and $m$ genetic generations are carried out — with $m$ being predefined (see Section 4 of S1 Appendix in supporting information) and applying the normalized sum of overlaps between reads as the adaptive function — the same as that applied in Eq 8 and detailed in Section 2 of S1 Appendix in supporting information.

After $m$ generations (objective function) the most fit individual is used for conducting the next episode in the agent’s RL training, hitherto interrupted. As each gene of the individual’s chromosome corresponds to one possible action, the complete gene sequence will contain distinct successive actions to be taken by the agent in the current episode, producing a mutual collaboration between RL and the genetic algorithm — the initial populations of the genetic algorithm are produced by RL and, as a counterpart, the results from the evolution of the genetic algorithm are introduced in an RL episode (Fig 5).

**Approach 3.2: Evolutionary-based assembly**

To estimate the genetic algorithm contribution in Approach 3.1, its assembling performance was evaluated separately, following the same configurations set for the
previous approach, but adopting as a starting population a set of individuals whose chromosomes were built from random permutations without repetition of reads.

Datasets and analysis

To assess the performance of all the approaches (including the seminal approach), in addition to the 18 datasets from Xavier et al. [13], 5 novel datasets derived from microgenomes extracted in previous studies [12, 13] were created. These are not arbitrary genome fragments, as were the case for previously used microgenomes (which had 25 bp and 50 bp), but represent larger fragments of previously annotated genes from the corresponding organism (i.e., *E. coli*). Given that the datasets are simulated data, no cycles in the genome were considered, which is a limitation of the approach. The experiments were carried out with 23 datasets, detailed in Table 1 — the last 5 lines are gene derived.

An environment for each dataset was created in the OpenAI Gym toolkit [24] to share such RL challenges. These environments are available online (see Section 1 of SI Appendix in supporting information), where the reward system proposed in Approach 1.4 is used. The identification names of each environment are presented in the last column of Table 1. The seminal reward system is also implemented and available — version 1, replacing *v2* with *v1* in the environment name field.

Two experiments were carried out to evaluate the approaches. In each experiment, 20 successive runs of each evaluated approach were performed for all 23 existing datasets (460 runs per approach). Given that each approach has different levels of complexity, the real execution time for each approach was considered for comparison. To reduce the interference of external factors in execution time, all experiments were individually and sequentially performed at the same station (with Ubuntu 16.04 in an AWS EC2 instance of the *r5a.large* type, dual core, 16 GB of RAM and 30 GB of storage).

In the first experiment (hereinafter referred to as experiment A) the objective was
Table 1. Public datasets used in the experiments.

| µgen. size | # reads | read size | Gym environment name         |
|------------|---------|-----------|------------------------------|
| 25         | 10      | 8         | GymnomeAssembly.25.10.8-v2   |
| 25         | 10      | 10        | GymnomeAssembly.25.10.10-v2  |
| 25         | 10      | 15        | GymnomeAssembly.25.10.15-v2  |
| 50         | 10      | 8         | GymnomeAssembly.50.10.8-v2   |
| 50         | 10      | 10        | GymnomeAssembly.50.10.10-v2  |
| 50         | 10      | 15        | GymnomeAssembly.50.10.15-v2  |
| 25         | 20      | 8         | GymnomeAssembly.25.20.8-v2   |
| 25         | 20      | 10        | GymnomeAssembly.25.20.10-v2  |
| 25         | 20      | 15        | GymnomeAssembly.25.20.15-v2  |
| 50         | 20      | 8         | GymnomeAssembly.50.20.8-v2   |
| 50         | 20      | 10        | GymnomeAssembly.50.20.10-v2  |
| 50         | 20      | 15        | GymnomeAssembly.50.20.15-v2  |
| 25         | 30      | 8         | GymnomeAssembly.25.30.8-v2   |
| 25         | 30      | 10        | GymnomeAssembly.25.30.10-v2  |
| 25         | 30      | 15        | GymnomeAssembly.25.30.15-v2  |
| 50         | 30      | 8         | GymnomeAssembly.50.30.8-v2   |
| 50         | 30      | 10        | GymnomeAssembly.50.30.10-v2  |
| 50         | 30      | 15        | GymnomeAssembly.50.30.15-v2  |
| 381        | 20      | 75        | GymnomeAssembly.381.20.75-v2 |
| 567        | 30      | 75        | GymnomeAssembly.567.30.75-v2 |
| 726        | 40      | 75        | GymnomeAssembly.726.40.75-v2 |
| 930        | 50      | 75        | GymnomeAssembly.930.50.75-v2 |
| 4224       | 230     | 75        | GymnomeAssembly.4224.230.75-v2|

The first column shows the size (in bp) of the microgenome used to generate the reads of each set; the second column shows the number of reads generated; the third column shows the size of the generated reads; and the fourth column shows the name of the environment built for each set in the OpenAI Gym toolkit.

to verify the impact of progressively including new strategies. For this purpose, the performance of the seminal approach was evaluated (according to [12]) against approaches 1.1, 1.2, 1.3, 1.4 (improved reward system), 2 (pruning dynamic) and 3.1 (genetic algorithm - GA). In the second experiment (or experiment B) the objective was to compare the performance of the new RL-based approaches against the performance of the GA alone. Therefore, in addition to Approaches 1.1, 1.2, 1.3, 1.4, 2 and 3.1, the approach 3.2 (which explores GA alone) was performed in an equivalent time.

For the performance measure in each experiment, two percentage measures were calculated, called the distance-based measure (DM) and reward-based measure (RM). Evaluations of de novo assembly are commonly performed using proper metrics, such as the N50 [25]. These metrics were created because, as previously indicated, de novo assemblies are not supported by a reference genome. In some scenarios, it is not possible to accurately assess the results obtained from the assemblers —- because the optimal output is unknown. Here, although a de novo assembler is evaluated, its assessment environment is restricted, and the target genomes are known; this scenario allows the use of specific (and exact) evaluations, such as DM and RM metrics.

DM considers a successful run when the consensus sequence from the orders of
reads produced is identical to the expected sequence. RM considers any run as a
success when the proposed order of reads represents the sum of \( PM_{\text{norm}} \) higher than
or equal to the sum of \( PM_{\text{norm}} \) from the optimal read sequence (for details, see
Section 3 of SI Appendix in supporting information).

Results

In experiment A the seminal approach consumed the longest running time (23 hours
and 34 minutes) and had the lowest average performance; an optimal response was
obtained in 16.96\% of the runs (i.e., 78 out of the 460 executions) in terms of distance
from the expected genome (DM) and 21.30\% (98 out of the 460) in terms of maximum
reward (RM) (Table 2).

| Experiment A (Approach) | Average DM (%) | Average RM (%) | Total runtime |
|------------------------|----------------|----------------|--------------|
| Seminal                | 16.96          | 21.30          | 23h34m       |
| 1.1                    | 9.57           | 13.70          | 19h38m       |
| 1.2                    | 18.48          | 21.30          | 19h38m       |
| 1.3                    | 20.00          | 24.35          | 19h38m       |
| 1.4                    | 20.43          | 24.78          | 19h38m       |
| 2                      | 20.65          | 25.00          | 18h41m       |
| 3.1                    | 73.91          | 80.87          | 17h03m       |

Table 2. Results achieved in experiment A.

The performances of each approach are expressed using distance-based (DM) and
reward-based (RM) metrics (see Methods for details).

Following the updated reward system, DM and MR performances in Approaches
1.2, 1.3, and 1.4 surpassed those of the previous approach, and consumed
approximately 4 hours less (19 hours and 38 minutes) of running time. Approach 3.1
presented the shortest running time, with a DM average of approximately 74\% and an
RM average above 80\%, with the highest performance.

In experiment B, according to Table 3, approach 3.2 presented the shortest running
time, with a DM average of 87\% and an RM average of 95\%. Given the superior
performance of Approach 3.2, Experiment B applied the time taken by the Genetic
Algorithm as a reference to find an optimal solution in terms of the RM for 22 out of
the the 23 datasets used (i.e., 95.65\%), which corresponded to 1 hour and 34 minutes
of running time. Given the dominance of Approach 3.2, we also verified the
performance of this approach on only the dataset with no optimal response (reads
with 4Kbp).

In this experiment the running time for Approach 3.2 was considerably increased
running time, lasting approximately 38 hours (against less than 2 minutes for the
same dataset for approach 3.2 in experiment B). No optimal solution was obtained for
this dataset, however, it is possible to observe a consistent gain in performance, in
terms of both DM (where longer runs had shorter distances than most distances
reached by shorter runs) and RM (which had higher accumulated rewards in all longer
runs) (Fig 6).

All the data generated or analyzed during this study, including reproduction codes,
are also publicly available (see Section 5 of SI Appendix in supporting information).
Table 3. Experimental performances considering similar running times (RT)

| Experiment B (Approach) | Average DM | Average RM | Total runtime |
|-------------------------|------------|------------|---------------|
| 1.4                     | 13.91%     | 17.61%     | 01h36m        |
| 2                       | 12.39%     | 16.30%     | 01h36m        |
| 3.1                     | 14.78%     | 14.78%     | 01h42m        |
| 3.2                     | 87.83%     | 95.65%     | 01h34m        |

Performances were expressed using Distance-based Measure (DM) and Reward-based Measure (RM) (see Methods).

Fig 6. GA performance evaluation on a long run. Violin plots demonstrating the performances obtained by the GA in experiments with short (1 h 34 m) and long (37 h 58 m) execution times in terms of the sums of DM and RM. The gray dots represent the distances/rewards obtained for all runs; the black line in the middle indicates the interquartile range; and the violin curves show the distribution density, where the wider the section, the greater the probability of the observations taking the corresponding value.

Discussion

Genome assembly is among the most complex problems confronted by computer scientists within the context of genomics projects. When applying machine learning to genome assembly this complexity allocates the problem of finding optimal permutations of sequenced reads and reaching the target genome into an NP-hard problem, which comprises the most difficult problems in computer science [26]. This high complexity is particularly expressed in the vast state space required for representing the assembly problem in RL models. To achieve the optimal solution in sets of 30 reads the RL agent should explore a state space of approximately $2^{44}$ states [12] (more than the stars in the universe). In real-world scenarios genomes would be much larger. Applying RL combined with heuristics is a strategy for addressing complex problems, aiming at mapping actions into states that tend to maximize their reward, thus decreasing the computational complexity of the problem.

In this study, we aimed to expand agent learning based on two difficulties observed in the seminal approach for applying RL to the genome assembly problem: (1) the
reward system and (2) the agent’s exploration strategies. We found that both improving the agent’s learning performance and updating the reward systems favored the agent to improve learning. However, this was not yet an optimal solution, as nonoptimal solutions were still present (less frequent) in the reward systems. This is also supported by the fact that RM percentages were higher than DM percentages in some experiments.

The dynamic pruning mechanism showed slight improvement, but the additional processing cost and the benefit from its implementation did not indicate a reasonable net gain from its use as bypass for the problem emerging from the high dimensionality of the state space. Part of the gains were due to the improved agent’s performance where the sum of rewards for the optimal permutation of reads was not maximized in the previous reward system. Despite the gains from the updated reward system, the inconsistencies were not completely resolved. In some of the datasets the agent reached and even surpassed the maximum expected accumulated rewards without obtaining the target genome. A minor improvement is observed in approach 2, requiring approximately one hour less of processing.

The hybrid approach combining RL strategy with GA (Approaches 3), presented better performances. This combination was proven to be advantageous, probably given the curse of dimensionality encountered by the Q-learning algorithm, as a strong GA support was observed for the agent while conducting the RL exploration.

Despite these improvements, the approaches are not suitable for real-world scenarios. This is evident in the experiments performed with the largest dataset. Even the smallest genomes from living organisms is greater than the largest dataset of this study and none of the proposed approaches produced an optimal response for this dataset, even when the best approach (GA) was applied for a longer time.

The superiority of the GA alone allows us to conclude on the infeasibility of applying the Q-learning algorithm to solve the genome assembly problem in search of an optimal read permutation, as proposed in the seminal approach.

Given the absence of approaches in the literature for tackling this problem through RL and considering the optimistic results obtained by RL in other areas (especially when RL is combined to deep learning), further investigations on the applicability of RL, including the use of different modeling approaches and algorithms, are needed.

All the experiments and the RL environments used in this study, are publicly available and open for reuse (for details see Section 5 of Supporting Information), to support future studies.

One of the major challenges in applying RL to real-world problems is the low sample efficiency of the algorithms [27]. Considering the time required by the agent trained by the Q-learning algorithm to reach an optimal solution, it is possible to perceive a high need for numerous interactions with the data. Considering that genome inputs are larger than those experimentally applied here, obtaining a sample efficient algorithm for the problem is in the core for developing a real-world solution. Additionally, the agent sample efficiency must be optimized to explore the state space, which might be achieved by the application of techniques to remove duplicate reads — due to repeats — and the use of an intrinsic motivation to bypass the exploration problem, given the high dimensionality of the proposed state space [27,28].

Although RL faces obstacles for commercial application, as mentioned before, it seems reasonable to consider the recent achievements of deep RL when applied to games, which present an equivalent computational problem to that of genome assembly [29,31]. The transformation of real-world problems into games is also a possibility for applying developed technologies [32]. One of the main benefits of representing the problem as a game is the reduction of the space of actions, which increases with the number of reads in the approaches proposed here.
The use of graph embedding may act as another option for modeling approaches allowing the use of deep RL without requiring the conversion of the problem into an image — the genome assembly problem may be represented through a graph, in the shape of the traveling salesman problem (TSP) [33,34].

Finally, one last aspect to be considered for the adoption of RL into the genome assembly problem is the generalization of the agent’s learning – a major challenge for the use of RL in real-world problems [35]. As designed for the RL environment for the genome assembly problem, the learning acquired by the agent when assembling a set of reads will hardly be applied for the assembly of a new set.

Supporting information

S1 Appendix. Supplementary Material. This is the supplementary material associated to this manuscript, containing further explanations.

S1 Fig. Example of a state space. Image file containing the picture corresponding to Fig 1

S2 Fig. Illustration of the application of reinforcement learning to the genome assembly problem. Image file containing the picture corresponding to Fig 2

S3 Fig. Illustration of the pruning procedure. Image file containing the picture corresponding to Fig 3

S4 Fig. Illustration of the proposed interaction between reinforcement learning (RL) and the genetic algorithm. Image file containing the picture corresponding to Fig 4

S5 Fig. Flowchart representing Approaches 1.1, 1.2, 1.3, 1.4, 2 and 3.1. Image file containing the picture corresponding to Fig 5

S6 Fig. GA performance evaluation on a long run. Image file containing the picture corresponding to Fig 6

References

1. Portin P, Wilkins A. The Evolving Definition of the Term “Gene”. Genetics. 2017;205(4):1353–1364. doi:10.1534/genetics.116.196956.

2. Heather JM, Chain B. The sequence of sequencers: The history of sequencing DNA. Genomics. 2016;107(1):1–8. doi:10.1016/j.ygeno.2015.11.003.

3. Ji P, Zhang Y, Wang J, Zhao F. MetaSort untangles metagenome assembly by reducing microbial community complexity. Nature Communications. 2017;8(1). doi:10.1038/ncomms14306.

4. Wong HL, MacLeod FL, White RA, Visscher PT, Burns BP. Microbial dark matter filling the niche in hypersaline microbial mats. Microbiome. 2020;8(1). doi:10.1186/s40168-020-00910-0.
5. Medvedev P, Georgiou K, Myers G, Brudno M. Computability of Models for Sequence Assembly. In: Lecture Notes in Computer Science; 2007. p. 289–301.

6. Gurevich A, Saveliev V, Vyahhi N, Tesler G. QUAST: quality assessment tool for genome assemblies. Bioinformatics. 2013;29(8):1072–1075. doi:10.1093/bioinformatics/btt086.

7. de Souza KP, Setubal JC, de Leon F de Carvalho ACP, Oliveira G, Chateau A, Alves R. Machine learning meets genome assembly. Brief in Bioinformatics. 2018;doi:10.1093/bib/bby072.

8. Yassine A, Riffi ME. A Review on Machine-Learning and Nature-Inspired Algorithms for Genome Assembly. International Journal of Advanced Computer Science and Applications (IJACSA). 2023;14(7):898. doi:10.14569/issn.2156-5570.

9. LeCun Y. Deep Learning Hardware: Past, Present, and Future. In: 2019 IEEE International Solid-State Circuits Conference - (ISSCC); 2019.

10. Botvinick M, Ritter S, Wang JX, Kurth-Nelson Z, Blundell C, Hassabis D. Reinforcement Learning, Fast and Slow. Trends in Cognitive Sciences. 2019;23(5):408–422. doi:10.1016/j.tics.2019.02.006.

11. Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Cambridge, MA, USA: A Bradford Book; 2018. Available from: http://incompleteideas.net/book/RLbook2020.pdf

12. Bocicor MI, Czibula G, Czibula IG. A Reinforcement Learning Approach for Solving the Fragment Assembly Problem. In: 2011 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing; 2011.

13. Xavier R, de Souza KP, Chateau A, Alves R. Genome Assembly Using Reinforcement Learning. In: Kowada L, de Oliveira D, editors. Advances in Bioinformatics and Computational Biology”. Cham: Springer International Publishing; 2020. p. 16–28.

14. Karami M, Alizadehsani R, Jahanian K, Argha A, Dehzangi I, Alinejad-Rokny H. Revolutionizing Genomics with Reinforcement Learning Techniques; 2023.

15. Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms, Third Edition. Cambridge, MA, USA: The MIT Press; 2009.

16. Grinstead CM, Snell JL. Introduction to Probability; 2012. Available from: https://books.google.com.br/books?id=7ip55DL72wC

17. Smith TF, Waterman MS. Identification of common molecular subsequences. Journal of Molecular Biology. 1981;147(1):195–197. doi:10.1016/0022-2836(81)90087-5.

18. Dulac-Arnold G, Mankowitz DJ, Hester T. Challenges of Real-World Reinforcement Learning. In: ICML 2019 Workshop on Reinforcement Learning for Real Life (RLRL); 2019.

19. Trott A, Zheng S, Xiong C, Socher R. Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards. In: Wallach HM, Larochelle H, Beygelzimer A, d’Alché-Buc F, Fox EB, Garnett R, editors.
20. Baluja S, Caruana R. Removing The Genetics from The Standard Genetic Algorithm. In: In Proceedings of ICML’95. California: Elsevier; 1995. p. 38–46.

21. Konar A. Evolutionary Computing Algorithms. In: Computational Intelligence; 2005. p. 323–351.

22. Gimelfarb M, Sanner S, Lee CG. Epsilon-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning. In: Adams RP, Gogate V, editors. Proceedings of Machine Learning Research. vol. 115. Tel Aviv, Israel: PMLR; 2020. p. 476–485.

23. Peterson EJ, Verstynen TD. A way around the exploration-exploitation dilemma. bioRxiv. 2019;doi:10.1101/671362.

24. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, et al.. OpenAI Gym; 2016.

25. Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, Birol I, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. GigaScience. 2013;2(1). doi:10.1186/2047-217x-2-10.

26. Roughgarden T. Algorithms Illuminated (Part 4): Algorithms for NP-Hard Problems. Algorithms Illuminated; 2020. Available from: https://books.google.com.br/books?id=FlmuzQEACAAJ.

27. Yu Y. Towards Sample Efficient Reinforcement Learning. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. IJCAI’18; 2018. p. 5739–5743.

28. Barto AG. Intrinsic Motivation and Reinforcement Learning. In: Intrinsically Motivated Learning in Natural and Artificial Systems; 2012. p. 17–47.

29. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature. 2019;575(7782):350–354. doi:10.1038/s41586-019-1724-z.

30. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. Mastering the game of Go without human knowledge. Nature. 2017;550(7676):354–359. doi:10.1038/nature24270.

31. Fjelland R. Why general artificial intelligence will not be realized. Humanities and Social Sciences Communications. 2020;7(1). doi:10.1057/s41599-020-0494-4.

32. Reis S, Reis LP, Lau N. Game Adaptation by Using Reinforcement Learning Over Meta Games. Group Decision and Negotiation. 2020;doi:10.1007/s10726-020-09652-8.

33. Cook WJ. In: Pushing the Limits; 2012. p. 211–212. Available from: http://www.jstor.org/stable/j.ctt7t8kc.15.

34. Li Z, Chen Y, Mu D, Yuan J, Shi Y, Zhang H, et al. Comparison of the two major classes of assembly algorithms: overlap-layout-consensus and de-bruijn-graph. Briefings in Functional Genomics. 2011;11(1):25–37. doi:10.1093/bfgp/elr035.
35. Ponsen M, Taylor ME, Tuyls K. Abstraction and Generalization in Reinforcement Learning: A Summary and Framework. In: Adaptive and Learning Agents; 2010. p. 1–32.