Review*: A Reinforcement Learning Based Approach to Multiple Sequence Alignment

Anthony Bodenhorst and Aurélien Chassagne

Abstract—This article is intended to reproduce and verify a previous paper called *A Reinforcement Learning Based Approach to Multiple Sequence Alignment* made by I.G. Mircea, I. Bocicor and G. Czibula in 2018. As clearly stated in their title, the goal of their paper is to use a Q-learning algorithm to determine the best order to align DNA sequences using an extended version of the Needleman-Wunsch algorithm. They compare their performance on small datasets over other implementations such as an ant colony optimization approach, concluding that their approach is slightly better and promising. In this review, we will slightly go deeper in their approach and qualify their comments regarding the performance of their approach to solving MSA, which in our opinion does not look particularly promising.

Index Terms—Article review, Bioinformatics, Multiple Sequence Alignment, Machine Learning, Q-Learning

1 Introduction

Multiple sequence alignment (MSA) consists of arranging sequences (e.g. DNA, RNA or proteins) with gaps in order to align as best as possible similar patterns. The multiple sequence alignment problem plays an important role in bioinformatics since it is often necessary to align three or more sequences in order to reveal common information between them that could reveal significant information about the evolutionary, functional or structural relationships that might exist between them.

MSA is generally used for sequences believed to share evolutionary relationships that may originate from common ancestors. Therefore it is for example one of the first steps required to build phylogenetic trees or determining sequence homologies.

The problem can very quickly become complex as soon as the length or the number of sequences increase. In the case of pairwise alignment, the well-known Needleman-Wunsch algorithm is widely used. However, the problem is much more complex when multiple sequences need to be aligned. One way to solve MSA is to iteratively apply the Needleman-Wunsch algorithm on each sequence (progressive alignment). We start by aligning two sequences together to create a *profile*. Then, we align the following sequence with this profile using the resulting Needleman-Wunsch or a new profile. Repeat until all sequences have been aligned. The main problem is that the result depends on the order in which we the pairwise alignment. Therefore, the goal of this article is to find the best alignment order using Q-learning. However, it is worth keeping in mind that MSA is a well-studied problem with many different approaches to solve it like genetic algorithms, ant colony optimization, and many more.

- ***Original article**: Ioan-Gabriel* Mircea, *Iuliana* Bocicor *and Gabriela* Czibula, A Reinforcement Learning Based Approach to Multiple Sequence Alignment, *2018, in Soft Computing Applications, ISBN: 978-3-319-62524-9.*

2 Multiple Sequence Alignment

We adapted the Needleman-Wunsch (NW) to the MSA as mentionned in the introduction. We iteratively aligned a new sequence with the profile (sequences already aligned). This sequence is aligned to the profile, by summing its alignment result to each sequence of the profile using the classical NW algorithm. There is however a difference with the classic NW since the sequences of the profile may already contain gaps. In this case, we consider that the *match/mismatch* with gaps are ignored (i.e. $+0$). Another remark is that in an evolving context, it is more likely to obtain a long gap rather than several gaps. Therefore, we can also differentiate between the creation of a new gap and the extension of an already existing gap in the NW.

To illustrate the NW for MSA, assume we want to align "GCATACA" using a scoring of **+2** for match, **-1** for mismatch and **-2** for gap penalty with the following profile:

$$\left.\begin{array}{c} \text{G–CAACA} \\ \text{GATTACA} \end{array}\right\} \Longrightarrow \left\{\begin{array}{c} \text{G––CAACA} \\ \text{G–ATTACA} \\ \text{GCA–TACA} \end{array}\right. \tag{1}$$

As we can see, even if the sequences of the profile are by definition already aligned with each other, it is possible to add gaps to them to better fit with the new sequences. If we add a gap to the profile, we must add it to all the sequences of the profile in order to keep their relative alignment between them. It is a progressive alignment in which early errors propagate to the final profile (i.e. previous gaps can not be removed). An important remark is that if we change the alignment order from previous example (eq. 1), we could obtain:

$$\left\{\begin{array}{c} \text{GCATACA} \\ \text{GATTACA} \\ \text{GCA–ACA} \end{array}\right. \tag{2}$$

Since the order of alignment matters, we present in the next section the Q-learning method to determine a good order to align sequences.

3 Q-Learning

The Q-learning is a reinforcement learning algorithm. This type of artificial intelligence is mainly based on the fact that an agent evolves in an environment and receives rewards depending on the state reached. Each possibility of action according to a state is subject to a numerical value whose meaning is the quality of the action in this state.

3.1 Learning

The objective of the training is to make the agent learn to select the order of the N given sequences to optimize their alignment. This training will be based on several episodes, episode which represents a reset of the game (selection of the order of the N sequences) without resetting the knowledge of the agent.

3.2 Episode

An episode represents a set of actions (and states) that the agent takes until the end of the game. To reach the end of an episode, two cases are possible:

1) The agent chooses N actions;
2) The agent chooses two identical actions.

3.3 State

A state represents a list composing a succession of actions chosen by the agent during an episode, these lists are of the form $\{1, 2, ..., N\}^k$ where $k \leq N$.
There is 3 types of states:

1) Initial state: beginning of the episode, no action was chosen by the agent;
2) Intermediate state: a state which is not initial of final;
3) Final state: a state representing the end of the episode.

For example the state representation for a game where N = 4 sequences could be:

- {} is the **initial** state;
- {1,2,3} is an **intermediate** state with 3 actions;
- {1,2,3,4} is a **final** state because the agent chose N actions;
- {1,2,1} is also a **final** state because the agent chose two identical actions.

3.4 Formula

The formula that updates the Q-value is the following:
$Q_{new}(s_t, a_t) =$
$Q(s_t, a_t) + \alpha * (R_{t+1} + \gamma * max_a(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t))$
Where :

- $Q_{new}(s_t, a_t)$: the new value of the action;
- $Q(s_t, a_t)$: current Q-value;
- s_t : current state;
- a_t : current action;
- α : the learning rate, value between 0 and 1;
- $max_a(Q(s_{t+1}, a_{t+1}))$: The maximum expected value of future rewards when reach state s_{t+1};
- γ : the discount rate (between 0 and 1). This rate reduces the maximum expected value of future rewards;
- R_{t+1} : the obtained reward.

3.5 Reward

After that an agent has chosen a new action, he receives a reward (or punishment) according to the new state obtained. This reward serves to tell our agent how good or bad this action was.
The reward is calculated as follows:

$$reward = \begin{cases} 0 & \text{if } len(list\_action) < 2 \\ -\infty & \text{elif 2 same actions in } list\_action \\ SP(list\_action) & \text{else} \end{cases}$$

The SP in the reward is the sum of pairs score. It is a MSA metric assessing the quality of the alignment. We compute it as follows:

$$SP = \sum_{i=1}^{N} \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} s(c_i^j, c_i^k)$$

Where:

- n is the number of sequences in the alignment;
- N is the number of columns in the alignment;
- c_i represents column i from the alignment ($\forall i \in \{1, 2, ..., N\}$);
- c_i^j : is the j^{th} character from the i^{th} column ($\forall j \in \{1, 2, ..., n\}$);
- $s(c_i^j, c_i^k)$ is the comparison metric between characters c_i^j and c_i^k:

$$s(c_i^j, c_i^k) = \begin{cases} 0 & \text{if} \quad c_i^j \text{ and } c_i^k \text{ are a } \textit{gap} & \\ -2 & \text{elif} \quad c_i^j \text{ or } c_i^k \text{ is a } \textit{gap} & \text{(i.e. gap)} \\ +2 & \text{elif} \quad c_i^j = c_i^k & \text{(i.e. match)} \\ -1 & \text{else} & \text{(i.e. mismatch)} \end{cases}$$

3.6 Q-table

The Q-table represents all the knowledge of our agent in a 2D-matrix (states and actions). It contains the Q-values which represents the quality of an action in a given state.

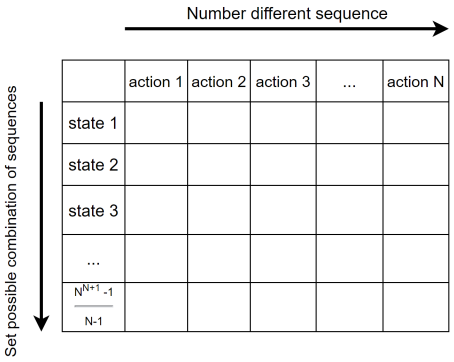


Fig. 1: Q-table representation

3.6.1 Dynamic Q-table

In the framework of the implementation we used the notion of dynamic Q-table. In the classic case of a Q-table we create the table in a static way before launching the game. In the context of a dynamic table, states will be added to the Q-table when they are first encountered. This decision was chosen because of the enormous space taken by Q-table when he is statically created. [1]

1. For example let say we have 12 sequences (N = 12), this give us about 9.7×10^{12} states ($= \frac{n^{n+1}-1}{n-1}$ [1]), i.e. about 100 TB to represent the Q-table (with 8-bit integer by Q-value).

3.7 Exploration & Exploitation

Exploration and exploitation are two important elements for an agent. It allows to balance the decision making of an agent to discover better unknown possibilities.

- The exploitation takes the best known action for a given state.
- The exploration takes random action potentially discovering a better output or leading to a sub-optimal or an unacceptable state (i.e. final state with $-\infty$ score).

In this article we will use the algorithm ϵ-gready, which will be used to balance the decision making. It consists of a threshold ϵ that separates the exploration from the exploitation

Algorithm 1 ϵ-gready

```
input : ε , value between 0 and 1
p ← random()          ▷ random between value 0 and 1
if p ≤ ε then
    explore
else
    exploit
```

An improvement of this algorithm is used in the experimental framework, ϵ-gready decay , its aim is to reduce exploration rates in favor of exploitation. **After each episode**, we reduce the ϵ value by a value ϵ_{decay} (between 0 and 1), and continue this reduction until we reach a value ϵ_{min}

$$\epsilon = max(\epsilon_{min}, \epsilon * \epsilon_{decay})$$

4 Implementation

We tried to reproduce the same implementation as in the original article [1] and we analyzed some variants to study the behavior and the robustness of this method. In the following, we will refer to our article implementation as *Vanilla*.

Although in the original article [1] they explain their implementation as well as the parameters used, some dark areas remains. This forced us to make some assumptions, explaining that our *Vanilla* version does not give the exact same results as those in the original article.

More details about our python implementation and results can be found in our GitHub[2].

4.1 Assumptions

Most of our assumptions concern the extend *Needleman-Wunsch* (NW) for MSA. Although the classical NW algorithm is well documented, this is not the case for its extension to MSA. The only information on their implementation of the NW for MSA is the scoring matrix (i.e. $+2$ for matches, -1 for mismatches and -2 for gaps). As a result, our *Vanilla* implementation differs from the article. We are certain of this since even when Q-learning gives the same answer (i.e. same order of alignment), we do not obtain the same alignment (i.e. different scores). The assumptions that we have done in the NW for MSA are:

- **No extend gap penalty**: Since it is never mentioned in the article, we assume that adding a new gap or extending a gap have the same cost (i.e. -2);
- **Matching with gap is neutral**: When a character of the sequence is match with a gap in the profile, this is neither a match nor a mismatch. Thus we consider that this action is ignored (i.e. $+0$).;
- **Match/mismatch is priority**: When we have an equality, we can either give priority to the diagonal direction (i.e. match/mismatch), or to the vertical or horizontal (i.e. gap). First we apply the diagonal direction and then the horizontal one.

The Q-Learning part is further explained. In the article, the RL agent is trained with the following settings; discount factor is $\gamma = 0.9$; the learning rate is $\alpha = 0.8$; the number of training episodes is 10^4; the exploitation threshold is $\epsilon = 0.2$[3]. We still had to make the following assumptions in our *Vanilla* version:

- **Dynamic Q-table**: They never mention it but we assumed that they are using a dynamic Q-table (not a static one). Otherwise, the Q-table of the hepatitis C dataset (5.1) could not be generated since it is far too huge;
- **Pruning**: In their implementation, they assign a score $-\infty$ when a solution is not acceptable but they do not mention that they pruned the search (i.e. episode) to directly start a new one. However, there is no point in not doing it. So we made the assumption that they applied it implicitly;
- **No epsilon decay**: This is never mentioned in the article so we assumed that they use a constant epsilon (ϵ =exploitation threshold).

4.2 Variants

In addition to the *Vanilla* implementation we just explained. We created five variants, changing only one parameter from the *Vanilla* version to study their impact:

- **Epsilon decay**: Use a dynamic epsilon starting to $\epsilon = 1$ and applied an of $\epsilon_{decay} = 0.95$ at each episode until $\epsilon_{min} = 0.01$
- **Without pruning**: In the original article, we do not have our second option as an ending of episode (i.e. "The agent chooses two identical actions"). This variant reproduces identically the information understood from the original article;
- **MSA extended gap**: Extending a gap in the NW has a score of -1 (instead of -2) because we prefer a large gap rather than multiple small gaps;
- **MSA scoring**: Modifies the scoring matrix used by the NW ($+5$ for matches, -3 for mismatches and -4 for gaps, as in another of their article [2]);
- **MSA priority**: Reverse priority order in case of equality (first vertical, then horizontal, and finally diagonal).

We have also implemented a **brute force** algorithm, which simply calculates all possible different permutations of the alignments and scores them.

2. https://github.com/abodenho/advance_bio_info

3. In the original article, they inverted the notion of exploration and exploitation compared with the current literature, and thus gave an ϵ of 0.8

5 RESULTS

We perform the experiments of our *Vanilla* version and our 5 variants on five of the seven DNA sequence datasets used in the original article since we did not have access to the last two datasets. The information of these datasets is summarized in table 1.

Dataset	# seqs	Avg. length	Max.	Min.
Hepatitis C [4]	10	211.9	212	211
Papio anubis [4]	5	1093	1098	1088
Dataset 1 [3]	3	39.33	40	39
Lemur, gorilla, mouse [3]	3	93	94	92
Rat, lemur, opossum [3]	3	129	129	129

TABLE 1: Detailed information of the data sets used for our experiments.

All experiments are averages over 20 runs of 10^3 episodes on a processor Intel Core i7-7700HQ at 2.8 GHz with 8 GB of RAM (instead of 5 runs of 10^4 episodes on an i3 processor at 2.4 GH with 3 GB of RAM in original article). The resulted alignments are evaluated on several measures: the average sum of pairs score (Score), the alignment length (AL), the number of perfectly matched columns (EM), the column score (CS=$\frac{EM}{AL}$), the average number of episode (Episode) and time in seconds (Time) to reach the optimal solution found.

5.1 Hepatitis C

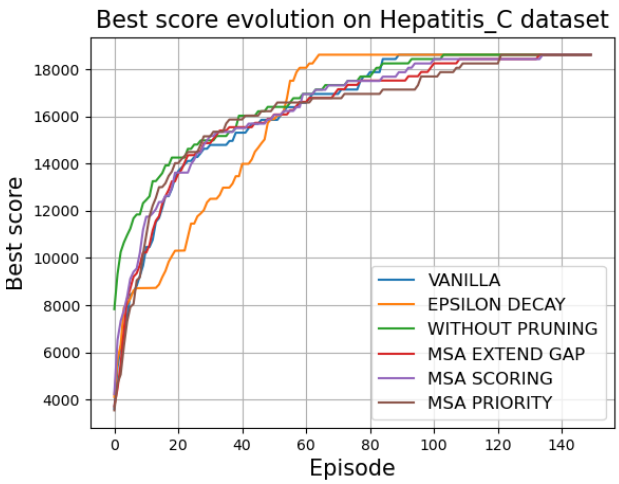
	Score	AL	EM	CS	Episode	Time
Article	**18627**	**212**	**198**	**0.93**	**<100**	**<2**
Vanilla	18627	212	198	0.93	60	4.2
Epsilon Decay	18627	212	198	0.93	52	4.1
Without Pruning	18627	212	198	0.93	59	7.7
MSA Extend gap	18627	212	198	0.93	64	4.3
MSA Scoring	18627	212	198	0.93	63	4.2
MSA Priority	18627	212	198	0.93	69	4.5

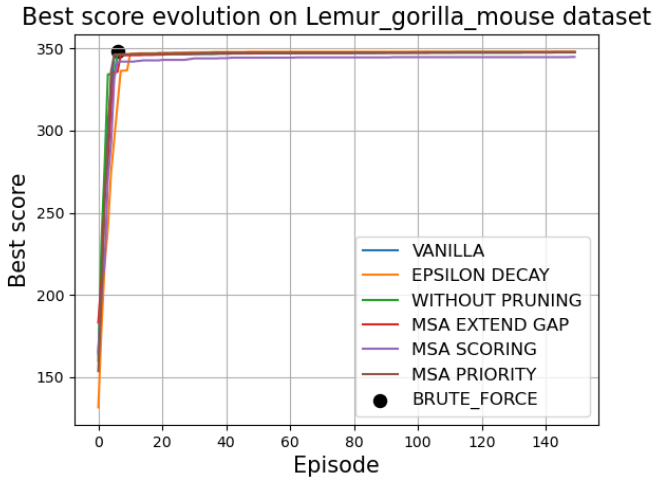
TABLE 2: Performance comparison on the *Hepatitis C* dataset.

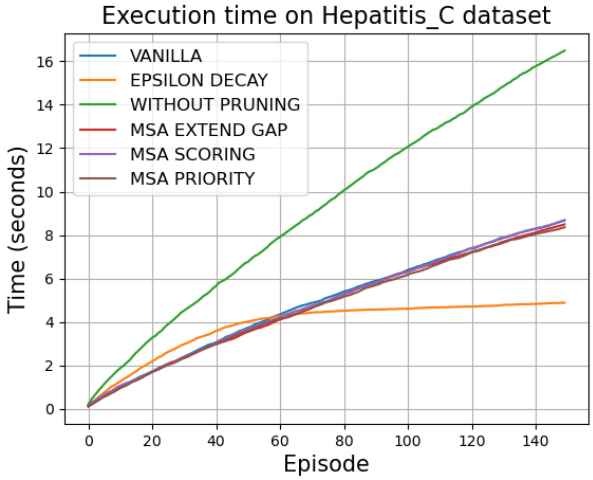
5.2 Lemur, gorilla, mouse

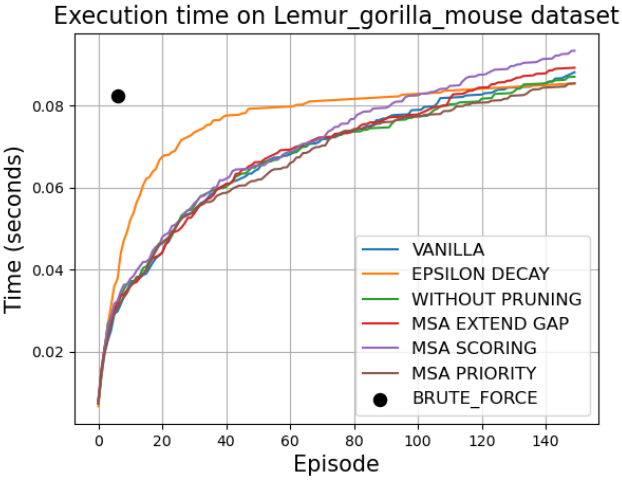
	Score	AL	EM	CS	Episode	Time
Article	**345**	**99**	**66**	**0.67**	**<100**	**<1**
Vanilla	345	99	64	0.65	50	< 0.1
Epsilon Decay	345	99	64	0.65	12	< 0.1
Without Pruning	345	99	64	0.65	39	< 0.1
MSA Extend gap	345	99	64	0.65	14	< 0.1
MSA Scoring	345	101	66	0.65	73	< 0.1
MSA Priority	345	99	64	0.65	51	< 0.1
Brute Force	348	94	60	0.64	6	< 0.1

TABLE 3: Performance comparison on the *Lemur, gorilla, mouse* dataset.









5.3 Papio anubis

	Score	AL	EM	CS	Episode	Time
Article	**18719**	**1110**	**918**	**0.83**	**160**	**190**
Vanilla	18881	1102	911	0.83	270	75
Epsilon Decay	18881	1102	911	0.83	169	38
Without Pruning	18878	1101	909	0.83	269	89
MSA Extend gap	18881	1102	911	0.83	272	83
MSA Scoring	18812	1112	916	0.82	382	106
MSA Priority	18872	1102	910	0.83	403	97
Brute Force	18881	1102	911	0.83	120	510

TABLE 4: Performance comparison on the *Papio anubis* dataset.

5.4 Dataset 1

	Score	AL	EM	CS	Episode	Time
Article	**167**	**42**	**31**	**0.74**	**<100**	**<1**
Vanilla	167	42	31	0.74	4	< 0.1
Epsilon Decay	167	42	31	0.74	4	< 0.1
Without Pruning	167	42	31	0.74	4	< 0.1
MSA Extend gap	167	42	31	0.74	3	< 0.1
MSA Scoring	167	42	31	0.74	3	< 0.1
MSA Priority	167	42	31	0.74	3	< 0.1
Brute Force	167	42	31	0.74	6	< 0.1

TABLE 5: Performance comparison on the *Dataset 1*.

5.5 Rat, lemur, opossum

	Score	AL	EM	CS	Episode	Time
Article	**486**	**133**	**87**	**0.65**	**<200**	**<1**
Vanilla	486	132	86	0.65	32	0.1
Epsilon Decay	486	132	86	0.65	52	0.1
Without Pruning	486	132	86	0.65	46	0.1
MSA Extend gap	486	132	86	0.65	52	0.1
MSA Scoring	477	135	89	0.66	36	0.1
MSA Priority	486	132	86	0.65	80	0.1
Brute Force	486	132	86	0.65	6	0.2

TABLE 6: Performance comparison on the *Rat, lemur, opossum* dataset.

6 Analyze & Discussion

First thing to mention is the variance of the results. By running the same tests several times, it is not uncommon to obtain slightly different results. Consequently, the results are a bit tricky to compare and must be taken with precaution, but we can still notice few points.

For the Papio anubis [5.3] dataset, our alignments are shorter, even when we get the same alignment order. Which proves that our implementation of the extend NW for MSA is not identical.

In general terms, the program execution is faster with our implementation, except for the Hepatitis C [5.1] which can be explained by the "poor" optimization of our NW implementation. The efficiency advantage can be explained by two things. First, We do not use the same computer to run the experiment (e.g. different processors). The second is our data structure used inside the implementation performs better than their data structure in term of rapidity.

The impact of pruning is great on large datasets as the Papio anubis [5.3] and Hepatitis C [5.1] datasets. This is explain due to the use of the NW algorithm (inside our implementation) which is time consuming. Therefore the pruning avoid to compute unnecessarily NW alignments.

The variants do not seem to bring much change, except the "Epsilon Decay" that stands out from the rest. It is faster (about 2x faster than the other variants on the Papio anubis [5.3] dataset). We can observe that the epsilon decay is slower at the beginning (exploration part) and becomes faster at the end (exploitation part). Although this may seem counterintuitive, it is normal since the exploration part has a higher chance of computing a new NW while the exploitation uses an already known solution (i.e. alignment). A final note on epsilon decay is that it is particularly slower on small datasets (e.g. [5.2], [5.4] and [5.5]) since the exploration does not take advantage of the pruning effect compared to datasets with many sequences (the exploitation cannot by definition achieve a not-accept state, and therefore cannot be pruned).

A last remark and not least is the comparison with brute force. We can see that on the three small datasets (e.g. 3 sequences, so only 6 permutations) the brute force is almost as fast as the Q-learning. It can therefore be considered as more optimal since it ensures to find the best order quickly and "faster" since Q-learning must continue to run even after reaching an optimal solution (i.e. the stopping criterion is often a number of episodes). However, brute force quickly becomes unusable when the dataset grows (e.g. brute force is not reasonable on the Hepatitis C dataset).

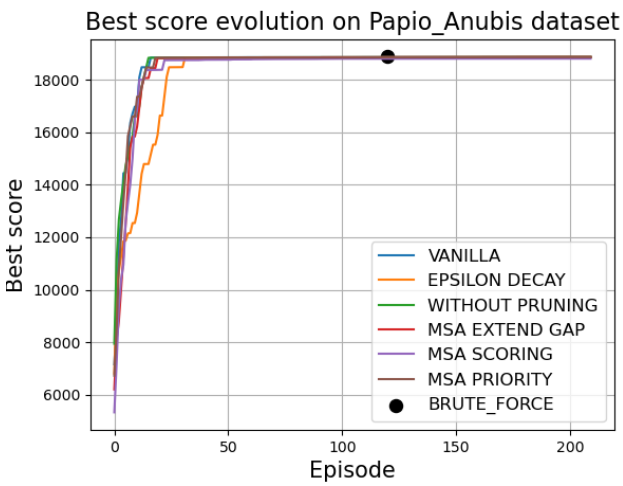
7 Conclusion

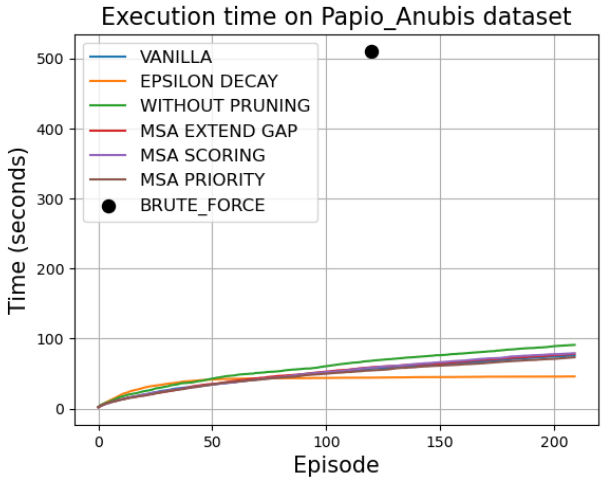
To conclude, we can say that for large datasets, the solution provided by a reinforcement learning model can be a good approach compared to a brute force approach. However, we have not had enough tests with large datasets to be able to say with any certainty whether this technique may be good in general. Indeed, four of the five datasets are small enough to be outperformed by a simple brute forcing algorithm. In addition, we have managed to show that a proposed variant ("Epsilon decay") may be a good improvement on the version presented in the original article by improving the result while reducing the time consumption. As final word, after reproducing the article, we believe that the use of Q-learning is not suited for MSA ordering problem, mainly due to the fact that the Q-table increases exponentially with the number of sequences. Not to mention that their implementation of Q-learning struggles to find a correct answer even after a large number of episodes when the number of sequences is too large.

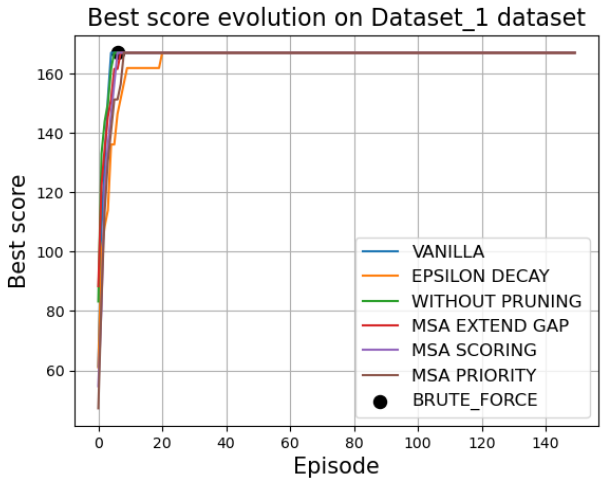
References

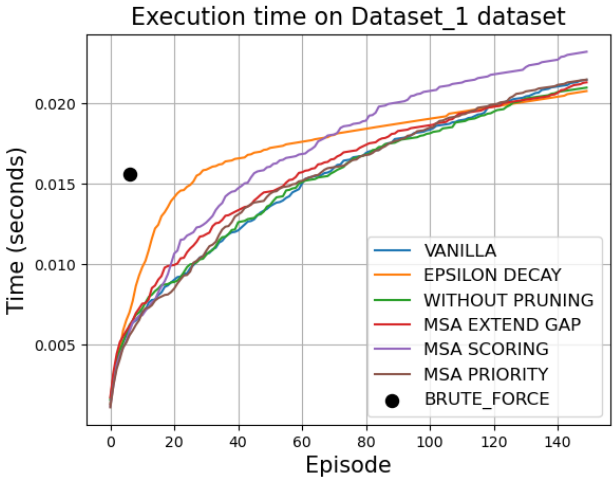
[1] Ioan-Gabriel Mircea, Iuliana Bocicor and Gabriela Czibula, *A Reinforcement Learning Based Approach to Multiple Sequence Alignment*, 2018, in *Soft Computing Applications*, ISBN: 978-3-319-62524-9.
[2] Ioan-Gabriel Mircea, Gabriela Czibula, Maria-Iuliana Bocicor, *A Q-learning Approach for Aligning Protein Sequences*, 2015.
[3] Xuyu Xiang, Dafan Zhang, Jiaohua Qin, Fu Yuanyuan, *Ant colony with genetic algorithm based on planar graph for multiple sequence alignment*, 2010.
[4] EMBL-EBI, The european bioinformatics institute. http://www.ebi.ac.uk/about

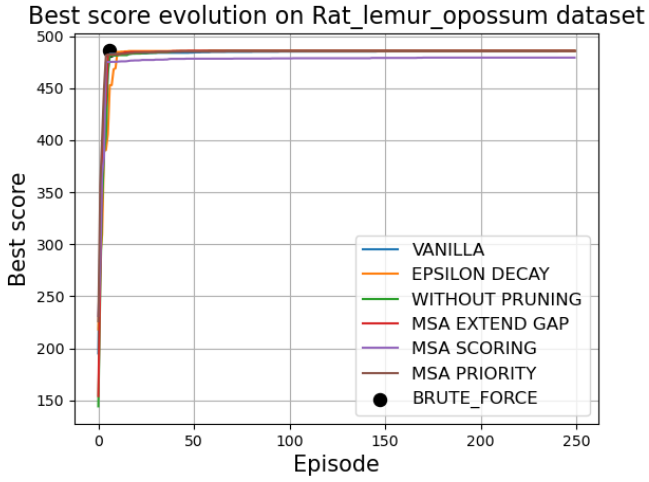
APPENDIX A

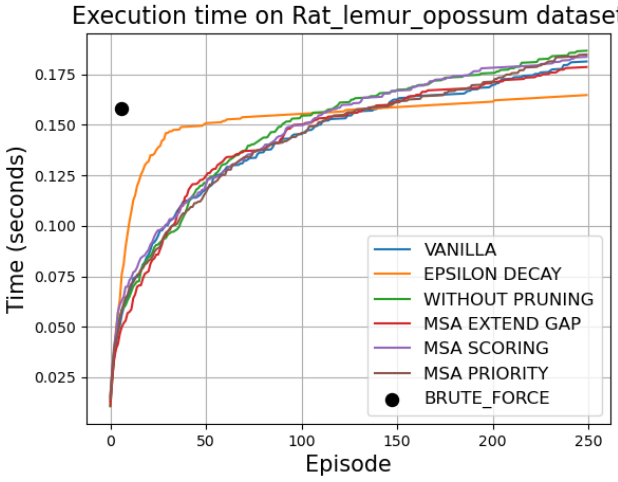

Best score evolution on Papio_Anubis dataset
Best score
Episode
VANILLA
EPSILON DECAY
WITHOUT PRUNING
MSA EXTEND GAP
MSA SCORING
MSA PRIORITY
BRUTE_FORCE


Execution time on Papio_Anubis dataset
Time (seconds)
Episode
VANILLA
EPSILON DECAY
WITHOUT PRUNING
MSA EXTEND GAP
MSA SCORING
MSA PRIORITY
BRUTE_FORCE


Best score evolution on Dataset_1 dataset
Best score
Episode
VANILLA
EPSILON DECAY
WITHOUT PRUNING
MSA EXTEND GAP
MSA SCORING
MSA PRIORITY
BRUTE_FORCE


Execution time on Dataset_1 dataset
Time (seconds)
Episode
VANILLA
EPSILON DECAY
WITHOUT PRUNING
MSA EXTEND GAP
MSA SCORING
MSA PRIORITY
BRUTE_FORCE


Best score evolution on Rat_lemur_opossum dataset
Best score
Episode
VANILLA
EPSILON DECAY
WITHOUT PRUNING
MSA EXTEND GAP
MSA SCORING
MSA PRIORITY
BRUTE_FORCE


Execution time on Rat_lemur_opossum dataset
Time (seconds)
Episode
VANILLA
EPSILON DECAY
WITHOUT PRUNING
MSA EXTEND GAP
MSA SCORING
MSA PRIORITY
BRUTE_FORCE