Master research Internship

Bibliographic report

Computing Genome Edit Distance

Bioinformatics - Performance

Author:
Ian Jeantet

Supervisor:
Dr Patrik Haslum
Decision Sciences Program
Data61 CSIRO

ENSSAT
LANNION

Abstract: We consider the problem of calculating the edit distance between two genomes based on the same set of genes in terms of inversion, transposition and transersion. As the problem can be formulated as a planning problem, there are various existing methods to solve it. However none of them were able to scale up to solve problems as large as we would like. In this document we will consider the problem as a state search problem. Hence, we will see various modified A* algorithms and pruning techniques coming from the optimal planning that would be applicable to enhance the existing optimal algorithm and would be able to increase the possible size of the genomes.

Contents

1 Introduction

The evolution of genomes through time was intensively studied to determine common ancestor or differences/similarities between a set of genomes. Considering a set of available operations O and a certain cost for each of these operations, the *genome edit distance problem* or *genome rearrangement problem* consists in finding the smallest cost resulting of the sum of the cost of each operation from O needed to obtain a genome G' starting from a genome G.

Here we restrict ourselves to an **ITT evolutionary model**. Two genomes G and G' match our model is they are a type of permutation of the same set $\{g_1, ..., g_n\}$ of genes. Hence they have the same size n. We consider G and G' as circular so every genes has exactly two neighbours. Each gene is also oriented in one of two directions which is indicated by a $\pm$ sign. The set of applicable operations on genomes that modify the order and orientation of the genes is composed of the three following operations :

- **Inversion:** The order of a stretch of genes is reversed. It negates the sign of each gene in the stretch. For example, $G' = (-g_{k-1}, ..., -g_1, g_k, ..., g_n)$ where $1 < k \leq n$ is an inversion of $G = (g_1, ..., g_n)$.
- **Transposition:** A stretch of genes is moved in the genome sequence. For example, $G' = (g_k, ..., g_l, g_1, ..., g_{k-1}, g_{l+1}, ..., g_n)$ where $1 \leq k, l \leq n$ is a transposition of G.
- **Transversion:** A composition of the two previous operations. For example, $G' = (-g_l, ..., -g_k, g_1, ..., g_{k-1}, g_{l+1}, ..., g_n)$ where $1 \leq k, l \leq n$ is a transversion (inverted transposition) of G.

Some other models were used in the past to solve this problem. Two of them are the *inversion only (IO) model* [Kaplan et al., 2000] and the *transposition only model* [Bafna and Pevzner, 1998]. Based on simplified versions of the ITT model they are considered as non-optimal approaches and restrict their available operations respectively to only inversion and to only transposition. Both can be solved in quadratic time ($O(n^2)$).

Also, operations of inversion and transposition often appear having the same cost in genome distance analysis. This is not necessarily true biologically, as transposition and inversion mutations do not naturally occur with the same frequency. [Eriksen et al., 2001] found that the optimal weights introducing least bias are 1 for inversions and 2 for transpositions and transversions. Using this [Eriksen, 2002] gave a polynomial time algorithm able to compute an $(1+\varepsilon)$-approximation of the minimal weighted distance with $\varepsilon > 0$.

The GED problem can be formulated as a *planning problem* which can be solved by several existing methods. The problem formulation and these methods will be developed in the first part of this document and we will focus on the *state search method* that corresponds to the scope of the subject.

The next step in this research project is to examine more advanced optimal search enhancements, such as symmetry reduction, partial expansion, branching strategies and others. The state of the art of these different techniques is presented in this document. These improvements are grouped in two parts. The first one is devoted to the explanations of various modified A* algorithms and the second one is focused on the presentation of different pruning techniques that exist in planning problem and that need to be adapted to the state search approach.

2 Previous works

2.1 Problem formulation

The GED problem was formulated for the first time as a **planning problem** by [Erdem and Tillier, 2005]. One of the genome is represented as the initial state and the second one as the goal state. The planner has to find a sequence of at most k operations that leads the initial state to the goal state. They extend their planning formulation to a model with duplicate genes, involving inversions, transpositions, transversions, insertions and deletions [Uras and Erdem, 2010] but this goes beyond the scope of the subject. Also [Haslum, 2011] phrased the genome edit distance problem as a domain-independent planning problem.

There are two main formalisms for planning problems, STRIPS formalism [Fikes and Nilsson, 1971] and SAS$^+$ formalism [Bäckström and Nebel, 1993]. STRIPS formalism is based on propositional logic when SAS$^+$ models states by multi-valued state variables. Note that there is a natural correspondence between STRIPS and SAS$^+$ formalisms [Helmert, 2006].

STRIPS formalism: In the STRIPS (Stanford Research Institute Problem Solver) formalism, a planning task is a tuple $\Pi = < P, O, I, G, C >$ where :

- P is a set of propositions,
- O is a finite set of operators. Each operator $o \in O$ has an associated set of preconditions $pre(o) \subseteq P$ and two sets of effects $add(o) \subseteq P$ (conditions made true by the action) and $del(o) \subseteq P$ (conditions made false by the action),
- I is the initial state,
- $G \subseteq P$ is a set of propositions where a state s is a goal state if $G \subseteq s$,
- $C : O \to \mathbb{R}^+$ is a non-negative cost function defined for each operators.

An operator o is considered applicable to a state s if $pre(o) \subseteq s$. If o is applicable, the resulting state $s[o] = (s \backslash del(o)) \cup add(o)$. The transition graph over the state space contains all edge $< s, s[o]; o >$ from s to $s[o]$ labelled with o for each state s and each operator o applicable to s. A sequence of operators $\pi = < o_1, ..., o_k >$ is applicable to s if it exists a path labelled by π and starting from s into the transition graph. If a such path exists, the end state is noted $s[\pi]$. If $s[\pi]$ is a goal state, π is a plan for s. It costs is the cumulative cost of each operator in the sequence: $C(\pi) = \sum_{i=1}^{k} C(o_i)$. A plan for s is called optimal if it has the minimal cost. The objective of optimal planning is to find an optimal plan for I.

SAS$^+$ formalism: In the SAS$^+$ formalism, a planning problem is defined as a tuple $\Pi = < V, O, S_I, S_G >$ where :

- $V = \{v_1, ..., v_n\}$ is a finite set of state variables. Each have a finite domain D_v. A fact is a couple $< v, d >$ where $v \in V$ and $d \in D_v$. A partial variable assignment is a set of facts, each with a different variable. A state is a partial variable assignment defined for each variables of the set V,

- O is a set of operators specified with $(pre(o), eff(o))$ where both are partial variable assignments. It is the same definition than for the STRIPS formalism but all the effects of o are regrouped in $eff(o)$,
- S_I is the initial state,
- S_G is the goal.

As these two formalism are very close, it is possible to define a state transition graph in the same way as we did for the STRIPS formalism.

Several approaches are proposed for solving planning problems.

2.2 Existing methods

2.2.1 Model Checking

One existing method is based on *model checking* and was introduced by [Cimatti et al., 1997]. The idea is to transform the planning problem into a model checking problem and to use existing model checkers to solve it.

2.2.2 Boolean Satisfiability problem

A similar method consists in transform the planning problem into a sequence of *boolean satisfiability problem* (SAT) [Kautz and Selman, 2006]. A SAT solver will be able to decide if the planning problem can be solved in N steps or not. By increasing the number of steps we will be able to determine the minimal value for N to solve the problem. However SAT problems are NP-complete so it is hard to solve large instances.

2.2.3 Constraint Satisfaction Problem

In his thesis [Nelson, 2013] tried to approach the genome edit distance problem as a *constraint satisfaction problem* (CSP) and to present the advantages and disadvantages of a such approach.

He found that constraint satisfaction is able to solve the problem with the advantage that using a CSP for modelling a problem allows to employ more general solving techniques, regardless of the problem domain. After having implemented two different constraint satisfaction models, he also worked on possible optimisations such as symmetry reducing constraints and genome compression algorithms. Finally he worked on alternative models using inversion-only distance and breakpoint distance. The CSP approach scaled slightly better, but still we are only able to find optimal solutions to problems of about half the size we need.

2.2.4 State Search Problem

Another approach consists to use *state space search* methods to solve planning problems. It corresponds to the approach that we want to improve and that was explored by [Wong, 2012]. It consists in finding a path (the sequence of operations of the planning problem) that links the initial state to a goal state.

This approach is based on a search algorithm and at least one heuristic function. Several search algorithms and heuristics exists but we will focus on those chosen by Wong. He used a standard A* search, with a state-of-the-art abstraction-based admissible heuristic. This, however, does not scale up to solve problems as large as we would like.

As the problem is very large and shallow, he needed to slightly modify the behaviour of his algorithm and used different techniques to rearrange the order of the nodes in the Open list to avoid to expand too many nodes.

Wong based his heuristic on a *pattern database heuristic* (PDB heuristic). However create a such heuristic requires memory and the size of the PDB grows exponentially with the number of genes. So he needed to add other heuristics to complete the PDB when the number of genes increases such as a *breakpoint heuristic* or a *sign heuristic*.

Breakpoint heuristic: The breakpoint heuristic [Blanchette et al., 1999, Blin et al., 2004] is based on the breakpoint distance. Given two genomes G and G' over the same set of signed genes and a pair of following genes g_1g_2 in G, we call breakpoint the fact that neither g_1 precedes g_2 nor $-g_2$ precedes $-g_1$ in G'. If G contains n genes ($|G| = n$), g_n precedes g_1 as we consider only circular genomes. The breakpoint distance $D_B(G, G')$ is the minimal number of breakpoints between G and G'. One of the ITT operations can fix at most 3 breakpoints (the transposition can) so we can use $\left\lceil \frac{D_B(G,G')}{3} \right\rceil$ as an admissible heuristic.

Sign heuristic: The sign heuristic is based on the number of consecutive gene segments that have a different sign between G and G'. One of the ITT operations can only fix at most one of these segments so we can use this number as an admissible heuristic.

On the experimental side he used sampling techniques to test these heuristics and found some interesting results on the ideal number of samples. If the genome size is to important for the PDB heuristic, a basic sampling technique consists in taking a certain number of samples of this genome that have the maximal size that the PDB allows. For each of these samples we use the PDB to find the optimal cost from it to the goal state. Then we take the maximum of these costs as an admissible heuristic. This heuristic is well admissible because it underestimate the cost between a given state and the goal state. There are a least the cost of the operations used to rearrange the sample plus the cost of the operations to rearrange the rest of the genome.

Another sampling technique was used with the breakpoint heuristic. It is based on the same principle than the previous sampling technique but instead of choosing arbitrary genes, we take the genes that are between two breakpoints and hence at the wrong place. It will give us a better heuristic because we don't take well placed genes to look into the PDB like it could happen in taking random genes.

On a first sight, more samples equals a better heuristic estimation what implies less node expansions but he found a limit around 15 samples where after the gain in node expansions is almost zero. In addition the overall computational time increases after 15 samples. Indeed there is little or no gain in node expansions but there are more computations for the extra samples. In other scenarios the signed breakpoint heuristic shows better results than the other heuristics if the number of samples is small.

More tests need to be done in more different scenarios to confirm or infirm these results and modified algorithms need to be used to improve them.

3 State of the art

3.1 Modified A* algorithms

In this part we consider the genome edit distance problem as a search problem based on a A* algorithm (as Wong did). As a reminder and for further notations, the A* algorithm is a best-first search algorithm which is formulated in terms of weighted graphs. A* is guided by the cost function $f(n) = g(n) + h(n)$ where $g(n)$ is the cost of the path from the start node to n and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal. If h is an admissible heuristic function (non-overestimating), A* is guaranteed to find an optimal solution.

The main issue with this algorithm is that, considering the large branching factor of our problem, it generates many nodes that will never expand. It results in a significant costs in memory and computation times. In this part we will see various modified A* algorithms trying to deal with our scale issue.

3.1.1 Partial Expansion A*

[Yoshizumi et al., 2000] applied a modified A* algorithm to the multiple sequence alignment problem which is close to the genome edit distance problem due to its large branching factor.

Principle: This Partial Expansion A* algorithm (PEA*) aims to reduce the amount of memory required during the A* execution by putting only the promising nodes in the Open list.

Every node contains the usual f-value and contains also an F-value which is initialized to the f-value. This F-value is used to determines if a child node is promising or not. When a node is expanded, PEA* generates all of its children but puts into the Open list only those that have an f-value less than or equals to the F-value of the parent node plus a predefined and non-negative cut-off value C. The remaining children are discarded and the expanded node is added back to the Open list with an F-value equals to the lowest f-value among all unpromising child nodes.

A* expands nodes in incremental order of f-value while PEA* expands nodes in incremental order of F-value.

Results: If $C = \infty$, this algorithm is identical to A* and if $C = 0$, only the best nodes with the optimal cost are stored. In this case, the search is performed with the same size of memory as the Closed list used by A* but it increases also the number of cumulative expanded nodes due to the possibility of re-expansion of a same node. And this increase of expanded nodes has a direct negative effect on the computational time.

In application to the multiple sequence alignment problem with $C = 0$, they achieved to align seven sequences (the best that A* can do in their configuration) with an average of only 4,7% of the amount of memory required in A* but with more than 5 times of cumulative expanded nodes than A*.

3.1.2 Partial Move A*

[Cazenave, 2010] introduced another variant of A*, the Partial Move A* (PMA*) that aims to reduce not only the amount of memory required for a large branching problem but also the computational time.

Principle: To cut the computational time (i.e not generate all child nodes when a node is expanded) this algorithm splits a move that corresponds to the generation of a child node into incomplete moves and creates in consequence partial nodes. In multi-agent path finding for example, a partial move is moving one agent to one of its neighbour locations and a complete move is moving all the agents to one of their neighbour locations. In the genome edit distance problem, a partial move could consist in choosing or not a gene mutation on the current location of the genome sequence whereas a complete move could consist in choosing a mutation for all the genome sequence. There are two stop conditions and one general case in PMA*:

- During an iteration, the f-value for the current partial node is computed and if this value is greater than the f-value of the current iteration, the path has greater cost and the algorithm stops.
- If not and if the current state completes a move which corresponds to a node then it stops if the goal is reached or it saved the node and starts a new move.
- In the general case, it calls recursively this algorithm for all the possible choices of next partial move.

In fact this algorithm uses a kind of dynamic programming to avoid to compute entirely all the child nodes. If the partial node is unpromising, all the children that can be created from it are not computed.

Results: Cazenave obtained better results than regular A* regarding the number of generated nodes in two well-known problems, the multiple sequence alignment problem and the multi-agent path finding problem. Unfortunately he did not compare PEA* and PMA* in particular on the computational time side.

3.1.3 Enhanced Partial Expansion A*

Another recent modified A* algorithm called Enhanced Partial Expansion A* (EPEA*) was developed to avoid to generate all the surplus nodes. The surplus nodes are all the nodes that not contribute to find the optimal solution. As seen previously they are discarded but they still are generated and it costs on the computational time side. EPEA* was first introduced by [Felner et al., 2012] and re-explained with more details by [Goldenberg et al., 2014]. It seems to be applicable to IDA* with pattern databases that partially corresponds to the previous work done by [Wong, 2012] on the genome edit distance problem. So it could be useful to apply this modified algorithm to our problem.

Principle: The pseudo-code is similar to that of PEA* and the difference is in the use of a domain and heuristic specific Operator Selection Function (OSF). While PEA* generates all the children nodes and discarding all the unpromising, EPEA* uses the OSF to generate only those who are needed and to calculate the lowest F-value of these generated children nodes. Each node is generated at most once during the search process and no child is regenerated when its parent is re-expanded. To improve our IDA* we can augment it with an OSF and work with a EPE-IDA*. To do this we will have to find the right OSF.

There are two types of OSFs:

- Direct Function (OSF_D) which returns exactly the set of operators desired.
- Ordered by Operators (OSF_O) which iterates through all the applicable operators to the current state and decide if it will produce or not the desired f-value.

In the case where a direct-computation OSF is not available for a given domain and heuristics, we can still construct an OSF_D for each node when it is expanded for the first time and use it for every re-expansion. On the other hand storing this function for every node in the Open list may cost a large amount of memory and have the opposite effect of what we are looking for with this novel algorithm.

Application: In their works [Felner et al., 2012, Goldenberg et al., 2014] built several OSFs for different well-known problems. They notably shown better result for the Rubik's Cube problem with EPEIDA* including a PDBs-based OSF.

Finally they show that EPEA* is most effective if :

- The domain possesses a large branching factor.
- There are only few distinct possible f-values for the children of a given node.
- An OSF_D is available what implies to be able to classify the operators to avoid to check all of them when a node is expanded.

3.1.4 PrefPEA*

In the same idea than EPEA* of not generating and evaluating all the successor nodes when a given node is expanded, [Ivankovic et al., 2014] stage the node expansion by the preferableness of successors. Instead of using an OSF, to determine the most promising child nodes, they use the information given by the heuristic when the parent node is generated. When the heuristic value of a node is computed, PrefPEA* stores with the node a set of preferred actions and when this node will be expanded, only preferred successors will be generated. This algorithm also prioritises expansion of node that have still unexplored preferred successors.

They showed that PrefPEA* is more effective if a node has many successors (as it is the case for our problem) and only few preferred ones. In the contrary they noticed that in a small number of cases, less than 2%, PrefPEA* may expand more nodes than A*.

To conclude on this part, we saw four modified A* algorithms that aim to reduce the amount of memory required and the computational time by acting on the number of nodes that are generated

Variant	Nodes generated	Nodes put into OPEN
A*	All	All
PEA*	All	Needed
PMA*	Partially	Needed
EPEA*	Needed	Needed
PrefPEA*	Preferred	Preferred

Table 1: Comparison of different modified A* algorithms

and stored. The Table 1 summarizes the theoretical results for each of them. It seems to be possible to adapt them to the genome edit distance problem but we need to deeply explore pruning techniques to make them more efficient.

As [Wong, 2012] explained, there are several available admissible heuristics for our problem and depending on the number of samples taken, no particular heuristic clearly differs from the others. Another idea would be to consider all of those heuristics at the same time and to use meta reasoning techniques to speed up the time taken by the heuristic computations. [Tolpin et al., 2013, Tolpin et al., 2014] suggested a Rational Lazy A* (resp. Rational Lazy IDA*) algorithm to easily reduce this time in A* (resp. IDA*). They found particularly good results if one heuristic is dominant but more costly than the others.

3.2 Pruning techniques

We saw on the first part some modified A* algorithms including pruning techniques but they may not be good enough. If a pure heuristic search based on A* algorithm is considered as one of the best approach to optimally solve classical planning problems, it is limited in our problem considering that heuristics lead to exponentially large state space. Hence, additional pruning techniques need to be used to efficiently solve the problem when the length of the gene sequence grows. That is why in this part we focus on recent pruning techniques for optimal planning such as *partial order reduction* or *symmetry elimination*. These pruning techniques will have to be adapted to the state space search method.

3.2.1 Partial Order Reduction

Partial order reduction (POR) is a technique for reducing the size of the state space by exploiting the commutativity of concurrently executed operations which result in the same state when executed in different orders. When two executions of actions sequences are sufficiently similar to each other then it is not necessary to investigate both of the executions. It has been extensively studied in model checking and has proven to be an enabling technique for reducing the search space and costs. Regardless of the method used for the POR, it needs to preserve completeness and optimality.

There are various POR methods such as *ample set method*, *stubborn set method* or *sleep set method*.

Ample sets: The ample set method relies on the notion of independence between transitions. The goal is to determine for each generated state the subset of successor states that need to be explored. Using an ample set should lead to a significant smaller state graph but needs to include

enough enabled operators to be sure to have correct results. Also the computation of an ample set should be doable in acceptable time.

Independence: Let be transitions t_1 and t_2 independent of each other such that $s \xrightarrow{t_1} s_1$ and $s \xrightarrow{t_2} s_2$ then it exists s' such that $s_1 \xrightarrow{t_2} s'$ and $s_2 \xrightarrow{t_1} s'$. In other words, the execution of both results in a unique state regardless the order in which they are executed. Transitions that are not independent are dependent.

Invisibility: In using the SAS$^+$ formalism, a transition from a state s is invisible if it has no effect on the partial variable assignment of each variables of s. As we define a state to its variable assignments, a transition t is invisible if $s \xrightarrow{t} s$.

The construction of an ample set revolves around 4 properties [Clarke et al., 1999]:

- **C0:** If a state has at least one successor in the full state space then it has at least one successor in the reduced state space ($\forall s \in S, ample(s) = \emptyset \iff enabled(s) = \emptyset$).
- **C1:** for all states s, a transition t' that is dependent on a transition $t \in ample(s)$ cannot be executed without a transition from $ample(s)$ occurring first. Checking C1 is at least as hard as checking reachability for the full state space so it needs to use an approximating heuristic.
- **C2:** If s is not fully expanded (i.e $enabled(s) \neq ample(s)$) then every transition $t \in ample(s)$ is invisible.
- **C3:** A cycle is not allowed if it contains a state in which some transition t is enabled but never included in $ample(s)$ for any state s on the cycle. A sufficient condition for C3 is that at least one state s along each cycle in the reduced state graph is fully expanded (in this case $enabled(s) = ample(s)$).

Persistent sets: A set T of transitions enabled in a state s is persistent iff, for all non-empty sequences of transitions $(t_1, ..., t_n)$ from $s_1 = s$ such that $\forall i, t_i \notin T$, t_n is independent in s_n with all transitions in T.

Stubborn sets: The stubborn set method [Valmari, 1989, Valmari and Hansen, 2016] is defined through commutativity over sequences of actions. A set of operations $T(s)$ is weakly stubborn at s if :

- **D0:** $\forall a \in T(s)$ and $\forall b_1, ..., b_n \notin T(s)$, if the execution of the sequence $(b_1, ..., b_n, a)$ is possible from s and leads to s', then the execution of the sequence $(a, b_1, ..., b_n)$ is possible and leads also to s'.
- **D1:** Either s is a deadlock or $\exists a \in T(s)$ such that $\forall b_1, ..., b_n \notin T(s)$ the execution of $(b_1, ..., b_n, a)$ is possible from s.

Transitions in a stubborn set T_s in a state s can be either enabled or disabled in s. If we take only the enabled transitions from T_s then the obtained set is a persistent set in s [Godefroid et al., 1996].

[Wehrle and Helmert, 2014] performed a comparison of different strategies for computing stubborn sets, using for instance envelope strategies.

Sleep sets: A sleep set is a set of transitions. The sleep set associated to a state s is a set of transitions that are enabled in s but will be not executed.

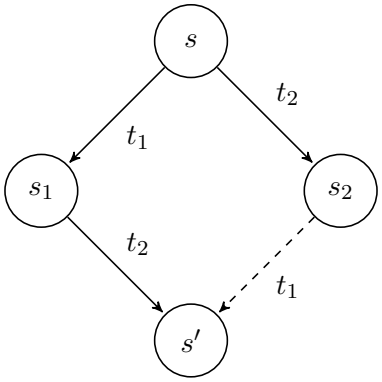

Figure 1: Reduce state space with sleep sets

We can see in Figure 1 that the sleep set for the state s_2 contains t_1 to avoid the exploration of t_1 in s_2 that leads to the same state than the transition t_2 form s_1.

3.2.2 Symmetry Elimination

In contrast to POR, symmetry elimination considers equivalence classes of symmetrical states and uses representative states of each equivalence class. In this case, A* with symmetry elimination prunes some of the resulting successor states and keeps only one per class. The optimal plan is preserved.

Let $\Pi = < P, O, I, G, C >$ be a STRIPS planning task as introduced in the Problem formulation section. A permutation σ of Π is a structural symmetry if:

- $\sigma(P) = P$
- $\sigma(O) = O$ and $\forall o \in O$:
  - $pre(\sigma(o)) = \sigma(pre(o))$
  - $add(\sigma(o)) = \sigma(add(o))$
  - $del(\sigma(o)) = \sigma(del(o))$
  - $C(\sigma(o)) = C(o)$
- $\sigma(G) = G$

This notion of structural symmetry allows directly reasoning on a compact representation of the state space called *orbit space*. The orbit of vertices can be computed in polynomial time and using graph theory, [Pochter et al., 2011] presented a set of algorithms applicable to state based planners without serious harm to the performances. First we need some definitions:

Permutation group: $\Sigma = \{\sigma_1, ..., \sigma_n\}$ is said to generate a finite group G if G is exactly all permutations obtained by repeatedly composing elements of Σ.

Orbit space: The orbit of a vertex s with respect to some subgroup G, denoted $G(s)$, is simply the set of vertices to which elements in G map s. $G(s) = \{\sigma(s) | \sigma \in G\}$.

Point-stabilizer: The point-stabilizer of a vertex s with respect to G, denoted G_s, is a a subgroup of G that contains all the permutations that fix s. $G_s = \{\sigma \in G | \sigma(s) = s\}$.

The goal of a such algorithm is to find a shortest path from S_I to any of the nodes in $G_{S_I}(S_G)$. Once a such path is found to a node $\sigma(S_G)$, the real path from S_I to S_G is simply calculated by applying σ^{-1} on the entire path. Indeed $\sigma \in G_{S_I}$ then $\sigma^{-1}(S_I) = S_I$. However, even if [Shleyfman et al., 2015] proved that many heuristics are invariant under structural symmetry, git is not guaranteed that $h(\sigma(S_G)) = 0$ so the idea is to work with the symmetry group G_{S_I,S_G} that stabilizes both start and goal state. They based their algorithms on the A* algorithm. The steps they added to A*are the following:

1. Before the search, find generator of G_{S_I,S_G}.
2. Whenever generating a state s', search for a previously generated state s such that $s' \in G_{S_I,S_G}(s)$. If s exists, treat s' exactly as if it was s (as s' is in the orbit of s), otherwise s' represents a newly found orbit.
3. Stops the search when a goal state is expanded.

Also [Wehrle et al., 2015] worked on a coupling approach of POR and symmetry elimination. They restricted the orbit space search to states generated by strong stubborn sets and it inherits the strengths of both approach.

4 Conclusion

We saw in this document that to enhance the results of the genome edit distance problem seen as a state search problem, two parts may be improved. On the one hand, various modified A* algorithms exist and seem to be able to improve the basic A* algorithm in terms of amount memory required and computational time. One the other hand some pruning techniques coming from the planning problem theory may be able to reduce considerably the state space. This will be very useful solve the GED as its branching factor is very large. The main part of this internship will be to adapt all of these techniques to the GED problem and to confront them or to use them simultaneously and prove by the experiment if it allows to increase the possible size of the genomes.

References

[Bäckström and Nebel, 1993] Bäckström, C. and Nebel, B. (1993). Complexity Results for SAS^+ Planning. *Computational Intelligence*, 11:625–655.

[Bafna and Pevzner, 1998] Bafna, V. and Pevzner, P. A. (1998). Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240.

[Blanchette et al., 1999] Blanchette, M., Kunisawa, T., and Sankoff, D. (1999). Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2):193–203.

[Blin et al., 2004] Blin, G., Fertin, G., and Chauve, C. (2004). The breakpoint distance for signed sequences. In *1st Conference on Algorithms and Computational Methods for biochemical and Evolutionary Networks (CompBioNets' 04)*, volume 3, pages 3–16. King's College London publications.

[Cazenave, 2010] Cazenave, T. (2010). Partial Move A*. In *International Conference on Tools with Artificial Intelligence*, pages 25–31.

[Cimatti et al., 1997] Cimatti, A., Giunchiglia, E., Giunchiglia, F., and Traverso, P. (1997). Planning via model checking: A decision procedure for AR. In *European Conference on Planning*, pages 130–142. Springer.

[Clarke et al., 1999] Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

[Erdem and Tillier, 2005] Erdem, E. and Tillier, E. (2005). Genome Rearrangement and Planning. In *Association for the Advancement of Artificial Intelligence*, pages 1139–1144.

[Eriksen, 2002] Eriksen, N. (2002). $(1+\varepsilon)$-Approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1):517–529.

[Eriksen et al., 2001] Eriksen, N., Dalevi, D., Andersson, S. G., and Eriksson, K. (2001). Gene order rearrangements with Derange: weights and reliability. *preprint*.

[Felner et al., 2012] Felner, A., Sturtevant, N., Goldenberg, M., Stern, R., Beja, T., Schaeffer, J., and Holte, R. C. (2012). Partial-Expansion A* with Selective Node Generation. In *Association for the Advancement of Artificial Intelligence*.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial intelligence*, 2(3-4):189–208.

[Godefroid et al., 1996] Godefroid, P., Van Leeuwen, J., Hartmanis, J., Goos, G., and Wolper, P. (1996). *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer Heidelberg.

[Goldenberg et al., 2012] Goldenberg, M., Felner, A., Stern, R., and Schaeffer, J. (2012). A* Variants for Optimal Multi-Agent Pathfinding. In *Symposium on Combinatorial Search*.

[Goldenberg et al., 2014] Goldenberg, M., Felner, A., Stern, R., Sturtevant, N., Holte, R. C., and Schaeffer, J. (2014). Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research*.

[Haslum, 2011] Haslum, P. (2011). Computing Genome Edit Distances using Domain-Independent Planning. In *International Conference on Automated Planning and Scheduling*.

[Helmert, 2006] Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, pages 191–246.

[Ivankovic et al., 2014] Ivankovic, F., Haslum, P., Thibaux, S., Shivashankar, V., and Nau, D. S. (2014). Optimal Planning with Global Numerical State Constraints. In *International Conference on Automated Planning and Scheduling*.

[Kaplan et al., 2000] Kaplan, H., Shamir, R., and Tarjan, R. E. (2000). A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing*, 29(3):880–892.

[Kautz and Selman, 2006] Kautz, H. and Selman, B. (2006). SATPLAN04: Planning as satisfiability. *Working Notes on the Fifth International Planning Competition (IPC-2006)*, pages 45–46.

[Nelson, 2013] Nelson, J. (2013). *Constraint Satisfaction for the Genome Edit Distance Problem*. PhD thesis, Australian National University.

[Pochter et al., 2011] Pochter, N., Zohar, A., and Rosenschein, J. S. (2011). Exploiting Problem Symmetries in State-Based Planners. In *Association for the Advancement of Artificial Intelligence*.

[Shleyfman et al., 2015] Shleyfman, A., Katz, M., Helmert, M., Sievers, S., and Wehrle, M. (2015). Heuristics and Symmetries in Classical Planning. In *Association for the Advancement of Artificial Intelligence*, pages 3371–3377.

[Tolpin et al., 2013] Tolpin, D., Beja, T., Shimony, S. E., Felner, A., and Karpas, E. (2013). Towards Rational Deployment of Multiple Heuristics in A*. In *International Joint Conference on Artificial Intelligence*, pages 674–680.

[Tolpin et al., 2014] Tolpin, D., Betzalel, O., Felner, A., and Shimony, S. E. (2014). Rational Deployment of Multiple Heuristics in IDA*. In *European Conference on Artificial Intelligence*, pages 1107–1108.

[Uras and Erdem, 2010] Uras, T. and Erdem, E. (2010). Genome Rearrangement and Planning: Revisited. In *International Conference on Automated Planning and Scheduling*, pages 250–253.

[Valmari, 1989] Valmari, A. (1989). Stubborn sets for reduced state space generation. In *International Conference on Application and Theory of Petri Nets*, pages 491–515. Springer.

[Valmari and Hansen, 2016] Valmari, A. and Hansen, H. (2016). Stubborn Set Intuition Explained. In *Proceedings of the International Workshop on Petri Nets and Software Engineering*, pages 213–232.

[Wehrle and Helmert, 2014] Wehrle, M. and Helmert, M. (2014). Efficient Stubborn Sets: Generalized Algorithms and Selection Strategies. In *International Conference on Automated Planning and Scheduling*.

[Wehrle et al., 2015] Wehrle, M., Helmert, M., Shleyfman, A., and Katz, M. (2015). Integrating Partial Order Reduction and Symmetry Elimination for Cost-optimal Classical Planning. In *International Joint Conference on Artificial Intelligence*, pages 1712–1718.

[Wong, 2012] Wong, Z. Y. (2012). Computing optimal genome edit distances under inversion and transposition. *ANU/NICTA*. Summer Scholar Research.

[Yoshizumi et al., 2000] Yoshizumi, T., Miura, T., and Ishida, T. (2000). A* with Partial Expansion for large branching factor problems. In *Association for the Advancement of Artificial Intelligence*, pages 923–929.