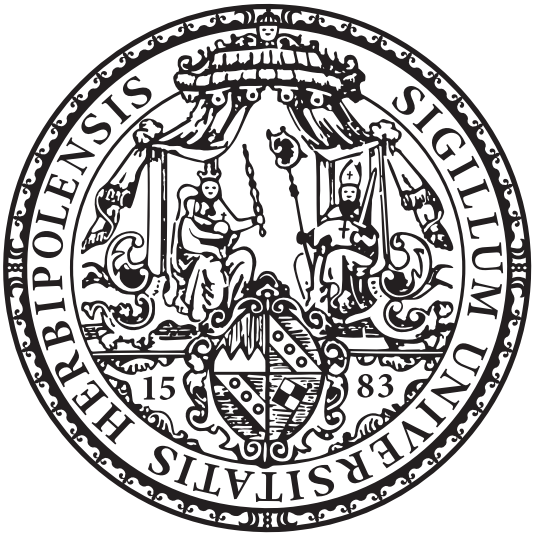
Practical Course Report

Map labelling with phylogenetic tree constraints

Alexander Zaft

Date of Submission: 29. September 2021
Advisors: Prof. Dr. Alexander Wolff
Dr. Jonathan Klawitter



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Abstract

In this report, methods for labelling maps with constraints given by a phylogenetic tree are considered. These phylogenetic trees are used to visualize the evolution of related species. The pyholgenetic tree then is a binary tree, where every internal node equals a differentiation of two species and every leaf is one of the species the analysis considers. In practice, the data for phylogenetic trees is derived with genetic analysis from samples of biological field studies . With the map of sample locations and the phylogenetic data, one can reason about the causes of the diverging evolution.

Our task is the labelling of the map such that we can draw the tree clearly. This is, why our main focus in this work is the rotation of the phylogenetic trees inner nodes. We will use four metrics to judge the quality of a certain rotation of the tree. For three of these metrics, a dynamic program for optimal solving will be implemented. The fourth metric can't be solved with the dynamic program. Instead, we will propose two approximations. All algorithms were also implemented in C/C++ For judging the quality of the algorithms, two methods of instance generation are implemented and additionally, an example from existing publications is judged by its quality. We will consider readability, number of crossings of the leaders and the running time of the implementation. The report will conclude with a discussion on open questions and alternative approaches for the approximation algorithms.

Zusammenfassung

Dieser Praktikumsbericht behandelt das Beschriften von Karten, wobei zusätzliche Einschränkungen in Form eines phylogenetischen Baumes gegeben sind. Mit phylogenetischen Bäumen wird die Entwicklung verschiedener verwandter Arten dargestellt. Dabei ist der phylogenetische Baum ein nicht notwendigerweise vollständiger Binärbaum, bei dem jeder innere Knoten einer Differenzierung zweier Arten entspricht, und die betrachteten Arten jeweils einem Blatt zugeordnet sind. Bei biologischen Feldstudien gesammelte Exemplare können durch Genanalyse Daten gewonnen werden, welche in einen phylogenetischen Baum überführt werden. Durch Karte und phylogenetischen Baum können unter anderem Rückschlüsse über mögliche Gründe von unterschiedlicher Entwicklung gezogen werden.

Aufgabe ist es nun also, den Baum möglichst übersichtlich an die Karte anzuzeichnen. In dieser Arbeit liegt das Hauptaugenmerk auf der richtigen Rotation der inneren Blätter des Baumes. Die Güte der Zeichnung wird hier mit vier verschiedenen Maßen bewertet. Für drei der vier Distanzmaße wird dabei ein Algorithmus implementiert, welcher ein optimales Ergebnis liefert. Für das letzte Distanzmaß werden zwei Approximationsalgorithmen implementiert. Das Programm wurde in in C/C++ geschrieben. Für die Bewertung der Algorithmen werden einerseits Instanzen generiert und andererseits existierende Daten aus biologischen Veröffentlichungen benutzt. Ausgewertet werden dabei Lesbarkeit, Kreuzungen unter den eingezeichneten Leadern und es wird auf die Laufzeit der Implementierung eingegangen. Abschließend wird diskutiert, welche Fragen offen bleiben und es werden Vorschläge für alternative Ansätze für die Approximationsalgorithmen gemacht.

1 Introduction

In biological field studies, often samples will be collected and genetically analysed. Through such a genetic analysis, a phylogenetic tree can be derived. These trees are binary trees, with each leaf representing a collected sample and inner nodes being known or potential relatives. Each inner node therefore stands in for an event after which the two resulting strands diverged evolutionary. This leads to a combination of spacial and genetic data, which may be displayed for easier analysis or visualisation.

To create such a visualisation, the places of collection will be shown on a map, usually colour-coded, and the phylogenetic tree from the analysis will be drawn onto one side of the map. The tree has to be drawn without crossings, but there are many possible embeddings with varying quality in readability. We chose four metrics as approximations for readability. One is the number of *po*-leader crossings the labelling would have. The second and third one are distances between the place where the leafs are drawn and the coordinates of the points in the map. These two differ, since one of them only considers the distance on one coordinate and the other depends on the euclidean distance between both of these points. The last metric looks at the order of the leaves and the topological sorting of the points, and considers the differences between them. For example if a point is the third in topological order, but the leaf which represents the same sample is the leftmost one, we would have to hop two places.

For this practical work, the goal is to label the map under the constraints set by the phylogenetic tree and draw the tree along one side of the map, with the trees leaves being placed in n evenly spaced slots at the maps border. Additionally, we may draw leaders from the slots to the corresponding points. We consider only two types here, which are *s*- and *po*-leaders.

Our algorithm for three of the above variants of the problem is implemented and two approximations for minimizing the crossings between *po*-leaders are implemented and tested.

For our experimental evaluation, generation for two different types of instances is implemented and a few real-world examples are compared to the solutions provided by the algorithm.

2 Problem Description

Now, we define the problem of drawing a phylogenetic tree for a map and additionally define the four metrics that we use throughout this report.

Map labelling with phylogenetic tree constraint Let $I = (M, P, T, D, K)$ be an instance of our problem. Let M be a rectangular area of the plane. This is our map. Let P be a set of n 2-dimensional points $P = \{p_1, \ldots, p_n\}$ in M. Let the phylogenetic tree T be a binary tree, where every leaf corresponds to a point in P. The direction D is given as the side of the rectangle, on which the n evenly spaced slots are designated, and therefore where T will be drawn. Lastly, the kind of leader K, which will be drawn

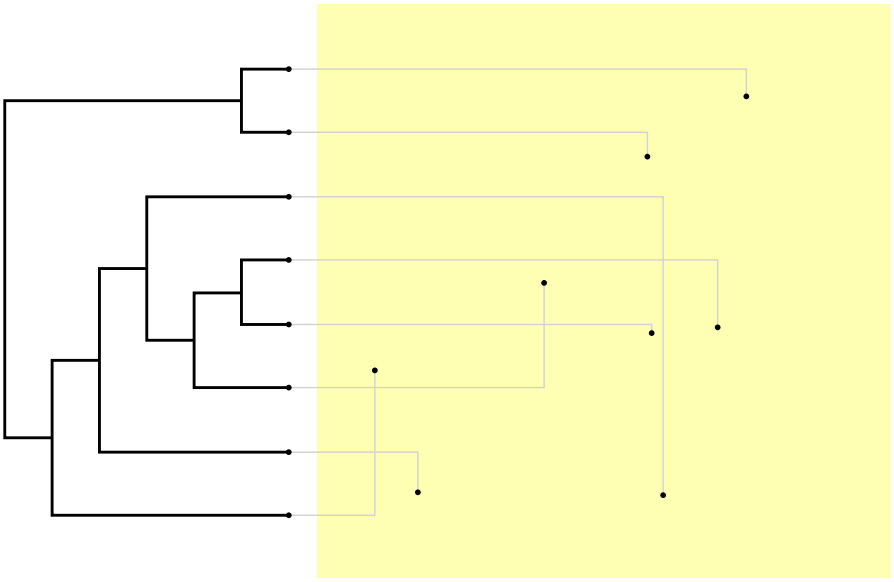


Fig. 1: An example instance with *po*-leaders displayed.

is known as either none, *s*-leaders or *po*-leaders.

An *s*-leader is a straight line from the slot of the leaf to the corresponding point. In contrast, a *po*-leader consists of one horizontal and one vertical line segment.

A labelling consists of the map and the tree. The tree is drawn as an orthogonal dendogram.

We evaluate the quality of a valid labelling of an instance with four measures.

Euclidian Distance The summed L2-distance between all points p_i and the slots s_i of their corresponding leafs. $\sum_{i=1}^{n} \sqrt{(p_i.x - s_i.x) + (p_i.y - s_i.y)}$

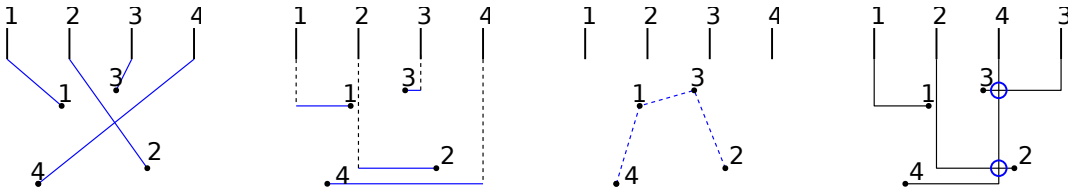
x-Distance The aggregated distance between slot and point in the coordinate parallel to the side on which the slots are placed. $\sum_{i=1}^{n} |p_i.x - s_i.x|$ if the tree is drawn above or below the map, or $\sum_{i=1}^{n} |p_i.y - s_i.y|$, if it is drawn to the left or the right of the map.

Hop-Distance The hop-distance for a point p is the difference between the position j of p in a topological sorting of P and the position k of the corresponding slot, which depends on the rotation of T.

***po*-leader crossings** When labelling the instance with *po*-leaders, the number of crossings in the labelling can be used as a quality metric.

It is important to note, that the first three metrics are linear, and the value of one of them can be computed if the values of both children are known by summing them.

For example, if we consider hop-distance and a single cherry, the hops needed to place both leaves on the same place in the sequence is just the sum of both individual moves.



(a) Example with euclidean distance measure shown in blue.

(b) Example with x-distance shown in blue.

(c) Example with a hop distance of 6. The topological sorting is displayed in blue.

(d) An example with two *po*-leader crossings. Note that slots 3 and 4 differ from the previous examples.

Fig. 2: Examples for all four metrics. The tree is not displayed.

This independence of every leaf is not given with *po*-leader crossings. As the name suggests, we consider interdependencies between the slot-point-pairs and can therefore not compute the number of crossings by simple addition, as there may be crossings between the children's subtrees, which are naturally not accounted for when only considering them separately.

3 Algorithms

We will look at three different algorithms for aligning the tree on one side of the map. The first is a dynamic program which solves the problem optimally for measurements 1 to 3. The other two are approximations for minimizing *po*-leader crossings (metric 4).

3.1 Dynamic program

In the dynamic program 1, we fill a table D, where each cell is indexed by node and position. For example, $D[k][i]$ will contain the metrics value, where the subtree with root k and size n_k is placed in the slots of the interval $[i, i+n_k]$. As we want to draw the binary tree in a crossing-free manner, there will be no gaps, and this table suffices for all possible positions of the subtrees. In general, we will traverse the tree in a bottom-up manner. The values in D for all leaves can be directly computed for every slot, as all three metrics are only concerned with the relation between assigned slot and the points location. For all inner nodes, we choose the rotation which minimizes the combined values for the left and right subtrees. This can be seen in line 9, where the value is computed for both rotations. The cells which have to be queried are for each child the current position we want to compute and the cell where the right child would start. For exampe, if the left child had a subtree with m leaves and we are currently computing the value of position i, then the leaves of the right subtree would start at position $i+m$ and the required value could be found in cell $D[r][i+m]$.

When the table is fully filled, the value for the whole tree is saved in $D[T.rootj][0]$, as the root has all n leaves in its subtree, leaving only this position possible. For calculating

the optimal rotation, which results in this value, we have to step through the table and try all rotations of the children to match the value in the current cell of D to find the right rotation. Saving the rotation of the two children from which the value derives will yield an increase in performance in practice, but makes no difference in theoretical runtime, as we will see later.

Algorithm 1: Dynamic program for metrics 1-3.

input : M, P, T, metric f
output: Optimal rotation of T according to f

```
1  D ← two-dimensional array of distances;
2  n ← |P|;
3  D ← Table with an entry per position and tree-node;
4  foreach leaf l of T do
5      for i ← 1 to n do
6          D[l][i] ← f(l, p_i);
7  foreach inner node t do
8      for i ← 1to n do
9          D[t][i] ← min {D[t.l][i] + D[t.r][i + t.l.size], D[t.r][i] + D[t.l][i + t.r.size]};
10 Step down through tree with respect to D[T.root][0] ;
11 return T
```

Running time Let n be the number of leaves of the phylogenetic tree. In the first loop and for every leaf, we iterate over all n possible slots and calculate the metric f once per slot. This leads to a quadratic effort, provided the calculation of f is possible in constant time. This is the case for all three metrics we are considering. In the second loop, we go through the tree bottom up and calculate the optimal rotation for both children for each inner node at every possible position. So for each inner node, there are at most n positions that have to be checked. For a single cell of D, this can be done by accessing D four times (once for each possible position of a leaf), calculating both sums and choosing the minimum. This takes constant time. Therefore, the second loop can be completed in $\mathcal{O}(n^2)$ time. As the trees root has exactly n leaves in its subtree, its interval has to start at position 0. With this knowledge, we can now rotate the tree according to table D, which results in the optimally rotated tree. As we have to look at every inner vertex, of which there are no more than n with n possible positions each, this is again a quadratic time effort. Summarizing, we have three steps which require quadratic time each. This leads to an overall running time of $\mathcal{O}(n^2)$.

In practice, one can store which rotation was chosen with the table D and speed up the last step.

3.2 Local Search

Local search is an approximation method that tries to iteratively approach an optimal solution (the global minimum/maximum of a function) by small steps. While it

has a tendency to reach local minima/maxima, it is easy to implement and there are modifications like simulated annealing and tabu-search. These try to mitigate the shortcoming of the basic search by allowing a small amount of setback in the hope of escaping local extrema. In this work, we only consider the basic local search algorithm.

Algorithm 2: Local search.

input : M, P, T
output: Rotation of T with minimal encountered leader crossings

```
1  repeat
2      k_min ←#po-leader crossings;
3      foreach node n ∈ T do
           // swap left and right child of n
4          n.rotate();
5          k_swapped ←#po-leader crossings;
6          if k_swapped < k_min then
7              k_min ← k_swapped;
8              break;                  // all nodes will have to be rechecked
9          else
               // revert swap
10             n.rotate();
11 until no additional improvements were found;
12 return T;
```

3.3 Top-Down Greedy

This algorithm steps through the tree in a top-down manner. At every node, it calculates the number of *po*-leaders that cross a center line between the childrens subtrees. This is done for both rotations of the children. From both rotations, the one with fewer crossings is chosen. The intention of minimizing these crossings into the siblings subtree is that both subtrees being separated as much as possible should give a low number of overall *po*-leader crossings.

Runtime At each of the n nodes, we have to check all of the *po*-leaders in the subtrees. At the root, this number is exactly n. The check between the centre line and a *po*-leader can be done by checking if both x-coordinates (if we draw the tree above the map) lie on one side of the centre. As this takes constant time, our total running time is $\mathcal{O}(n^2)$.

4 Instance generation

We now describe the method for generating synthetic instances for evaluation. For testing purposes, two generation methods were implemented, namely uniform scattering and a clustered scattering of points in an area. Two examples can be seen in Figure 3.

Algorithm 3: Top-down greedy algorithm.

input : M, P, T
output: Rotation of T minimizing *po*-leaders crossing the line dividing the subtrees.

```
1  Q ← empty queue;
2  Q.add(root);                                   // start at root node
3  while Q ≠ ∅ do
4      c ← Q.dequeue();
5      if c is a leaf then
6          continue;
7      mid ← coordinate of dividing line between c.l and c.r;
8      k_left ← 0;
9      foreach leaf l in the subtree of c do
10         if leader of l crosses mid then
11             kleft ← k_left + 1;
12     c.rotate();                                // swap the children
13     k_right ← 0;
14     foreach leaf l in the subtree of c do
15         if leader of l crosses mid then
16             k_right ← k_right + 1;
17     if k_left < k_right then
18         c.rotate();                  // rotate back, original was better
19     Q.add(c.l);
20     Q.add(c.r);
21 return T;
```

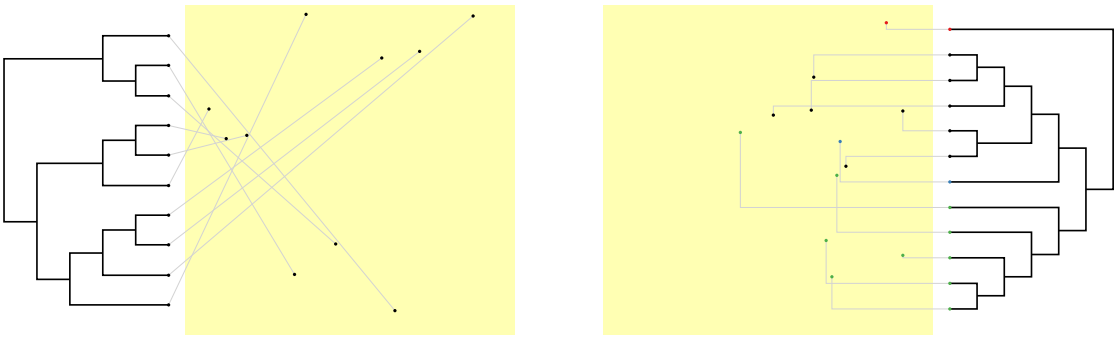
4.1 Uniform

The number of points n and the size of the area are given to the algorithm. The points are scattered uniformly in a rectangular area. The corresponding tree is created by a hierarchical bottom-up clustering of the points.

Optionally, one can introduce errors in the clustering by not merging the points which the algorithm would choose, but merging two random clusters.

4.2 Clustered

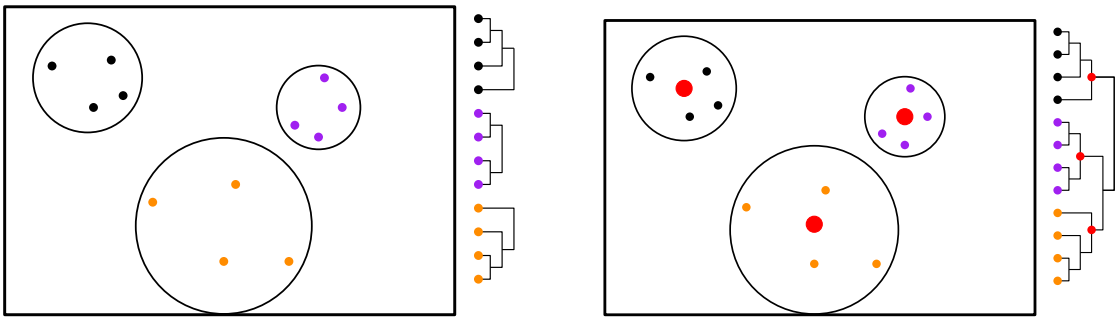
Now, we will create a clustered instance. As before, the number of points n and the size of the area are given. Additionally, the number of clusters c and the median distance from the centre are specified. For clustered instances, we first scatter c cluster centres in the area, just as we scattered the points in Section 4.1. After that, a random number of points is scattered around each centre, such that we have c non-empty clusters. To ensure



(a) Randomly generated uniform instance with s-leaders.

(b) Randomly generated instance with 3 clusters and po-leaders.

Fig. 3: Unsolved examples for both methods of instance generation.



(a) Clustering each group of points separately.

(b) Clustering the centroids of each group.

Fig. 4: The two stages used for determining the tree used in clustered instances.

non-empty clusters, the number of points in a cluster is randomly chosen between one and the number of points which remain to be scattered minus the number of remaining centres. As can be seen in Figure 4, we cluster the generated points hierarchically first on a per cluster basis (Figure 4a). Afterwards the resulting centroids are clustered, as can be seen in Figure 4b. With this, we have created our tree. This approach allows clusters that overlap on the map, but which are separated in the tree, which would be hard with a single clustering with all points, even with the introduction of errors, which is not as controlled.

4.3 Examples

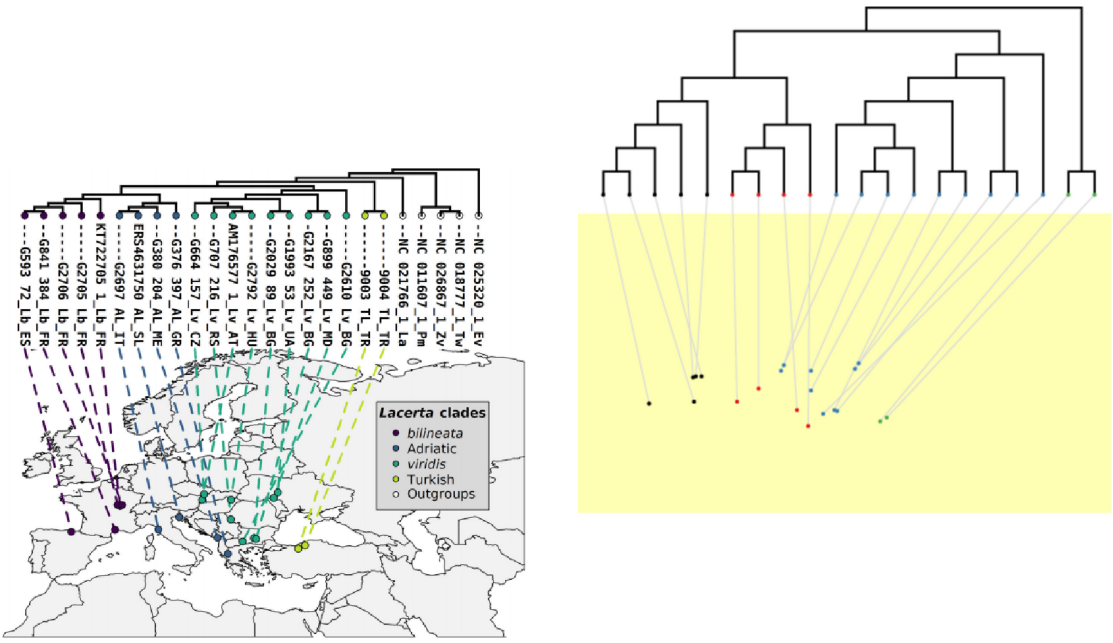
In Figure 3, we can see results of both instance generation methods.

5 Discussion

We will now discuss the performance of the algorithms from Section 3. For this, we first test an instance from an existing paper and compare the labellings. Secondly, we will

qualitatively evaluate the readability of metrics 1-3 and take a look at the runtime of the implementation. The last subsection concerns the evaluation of both approximation algorithms.

5.1 Real world instance

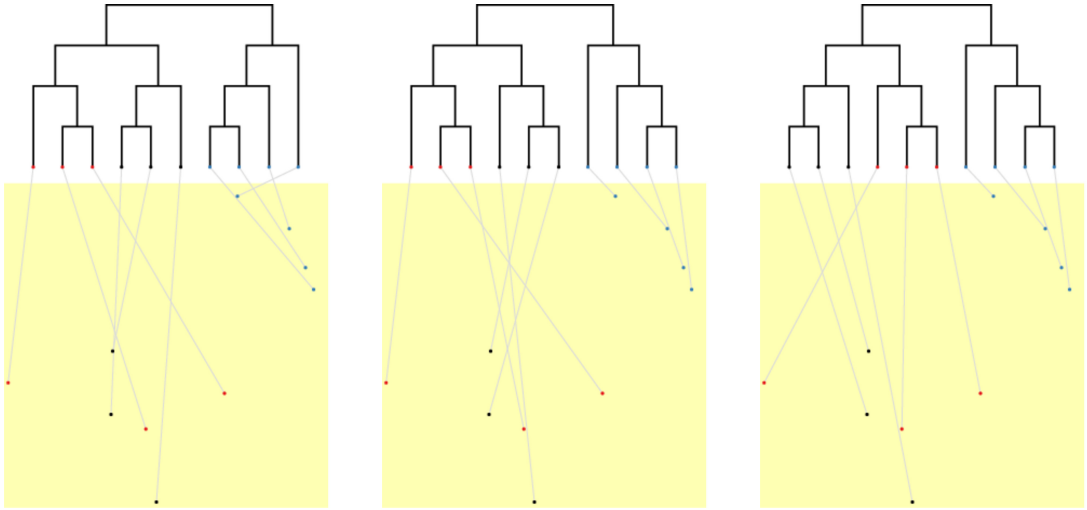


(a) Real instance.

(b) Computed solution with euclidean distance metric.

Fig. 5: An example of a real world instance with a solution from the algorithm.

In this section, we consider an instance from a biological paper [JSK+21]. In Figure 5 you can see the layout from the paper and the solution computed by the dynamic program with the euclidian metric. The algorithm differs in one rotation, which is named "G707 216_Lv_RS", which was rotated to the right of the neighbouring cherry. This eliminates one crossing. Solving the instance with the x-distance metric leads to many crossings in the teal cluster, while the solution with hop-distance introduces one additional crossing in the leftmost cluster. We can see, that overall the solutions seem to make sense, allthough one should consider all metrics, as each has some shortcomings. Manual post-processing could fix many of those errors, with a small amount of work.



(a) Euclidean distance. (b) x-distance. (c) Hop-distance.

Fig. 6: Comparison of tree alignment for a single instance.

5.2 Readability of different metrics

Displaying a map with a phylogenetic tree without leaders can de-clutter the image. Even with leaders, certain orderings make figures easier to read. In Figure 6, one example is displayed with the optimal alignment for each metric.

x-distance The x-distance, seen in Figure 6b, is generally the hardest to read. While it separates the rightmost cluster of the example nicely, this only happens if a clusters y-coordinate is monotonically increasing or decreasing when we increase x. This can be seen with the leftmost (red) subtree in the image: Flipping the cherry would improve readability as the crossing would be eliminated.

Euclidian distance Euclidean distance expectedly results in shorter leaders compared to the other two metrics. This can be especially good if there are clusters which are far apart in y-coordinate, which is not the case in the provided example. The depth which is ignored in with the previous metric. One regularly occurring problem, especially when drawing straight line leaders, is the frequency of very shallow crossing angles. Additionally, points on the edge of clusters may be placed so that they cross most of their neighbours. Both problems can be seen in Figure 6a.

Hop-distance The great advantage of hop-distance can be seen in Figure 6c. In a majority of cases, the points are aligned left-to-right for each cluster. This is especially helpful when not drawing leaders and for example only using color to associate a cluster with the points in the map. When clusters are not overlapping too drastically, this also leads to crossings with a larger angle between both leaders. In some instances,

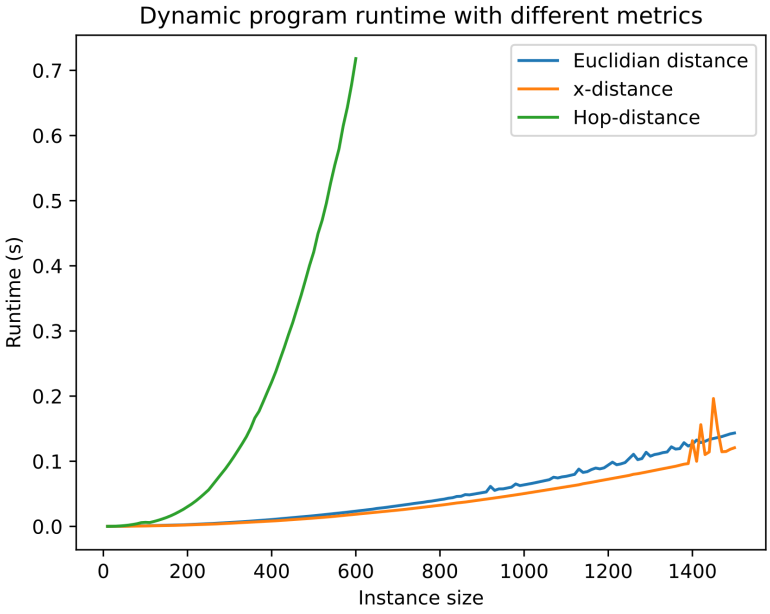


Fig. 7: Comparison of the programs running time for all metrics with instances up to a size of 1500. Mean of 20 runs over randomly generated instances.

with phylogenetic clusters which are dispersed and overlapping, it would be better to consider x-distance as a quality metric, because the hop-distance will vary widely within one cluster. This also leads to the nice property of a sorting within one of these clusters being lost.

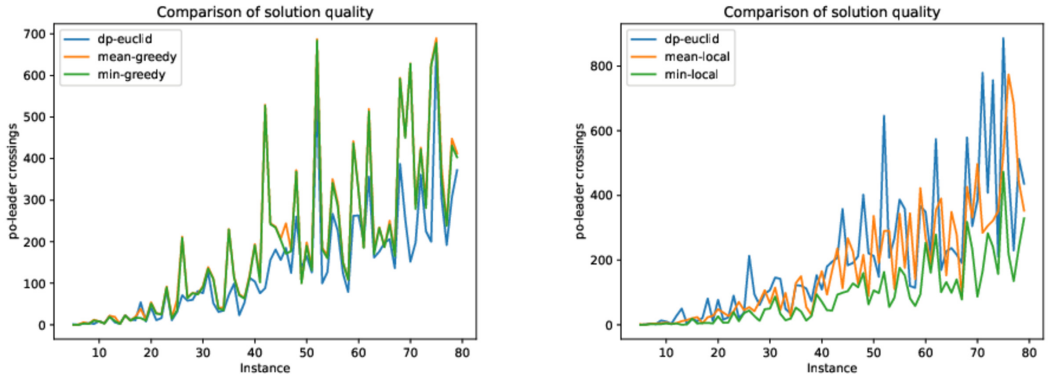
5.3 Running time of the implementation

In Figure 7 we can see the running time of the program on a number of generated instances up to a size of 1500 points. The generation produced clustered instances with between three and 12 clusters, the number increasing as the instances get bigger. The quadratic behaviour, which was discussed in Section 3.1, can be seen quite clearly. The difference in slope can be explained by the time it takes to compute the metrics in the initial seeding of the table. While changing implementation details should improve the running time, even with hop-distance examples with 500 points can be calculated in less than one second, which should suffice for most practical examples.

5.4 Number of *po*-leader crossings

To judge the quality of both approximations, a few pre-generated instances were shuffled and solved afterwards. The dynamic program was used with all three metrics to generate a baseline for comparison. The results can be seen in Figure 8. The greedy algorithm performs worse in most cases, its minimum is also relatively near its mean. The mean

result of the local search varies widely in quality, but the minimum which was found in 50 tries consistently finds the best solution we know of.



(a) Performance of the greedy approximation algorithm.

(b) Performance of the local search.

Fig. 8: Number of *po*-leader crossings for instances of size 5 to 100. For each instance, both approximation algorithms were allotted 50 retries with a randomly rotated tree. The dynamic program with euclidean distance was taken as a baseline.

An interesting effect is the sudden jump in crossings for the solution of all algorithms if the instances size reaches a certain threshold. The effect can be seen in Figure 9. If one varies the size of the map, the number of points on the map and the spread of the points when generating the samples, this jump is always present around the mark when the width of the map divided by the number of points is approximately one. If this is an effect of the generation methods is not known, but intuitively, if the density of the scattered points on the map increases, the possibilities of moving a leaf without it crossing many other leaders in its new place are getting ever slimmer. Qualitatively, the greedy algorithm separates big clusters very well but struggles within clusters and especially when the tree is not an exact clustering of the points. The simple local search does generally well, when one allows enough retries and is especially good in fixing the labelling when there is an error inside a cluster.

6 Conclusion and Future Work

In this work we defined the problem of labelling a map under constraints given by a phylogenetic tree. Additionally, four different metrics for evaluating the quality of solutions were chosen. Afterwards, an dynamic program for calculating an optimal solution for three of the four metrics was implemented. For the fourth metric, we chose two approximation algorithms. The dynamic program was evaluated in runtime and aesthetic quality and the approximation algorithms were tested for the quality of the approximation.

The largest open question is the difficulty of calculating a *po*-leader labelling with minimum-crossings. Neither a proof of NP-hardness nor a fast algorithm is known to the author. Due to the combination of tree and point layout, mapping other NP-problems to the labelling problem was unintuitive, as both the points and the tree have to be chosen correctly. One possible approach this question could be an implementation of the problem as a linear program. This would also yield a possibility to compare the approximation algorithms to the real best solution.

Apart from the optimal solution for the minimization problem, one could improve the approximation algorithms. As we have seen, the greedy algorithm is good at separating larger clusters, especially if these clusters of points correspond to a subtree. One possible improvement would be to switch to a local search after traversing down the tree far enough with the greedy algorithm. This should provide the search with a good starting state. The decision of switching could be done with several heuristics, for example traversing the whole tree or switching after a threshold is reached in the sum of distances of the subtrees points.

For local search, simulated annealing could be used to escape local minima. Tabu-search, where recently travelled states (rotations of the binary tree) are disallowed may also improve the approximation. An idea for another approximation would be drawing the leaders without the constraints of the tree. This can be done crossing-free in polynomial time [BKSW07]. Afterwards, one could rearrange only those points which violate the trees constraints.

Looking further, more general problem which may be worth considering would be placing the slots on a line, which may cross the rectangular map, or more complicated versions, e.g. placing the slots on a circle around the map[PSD$^+$21].

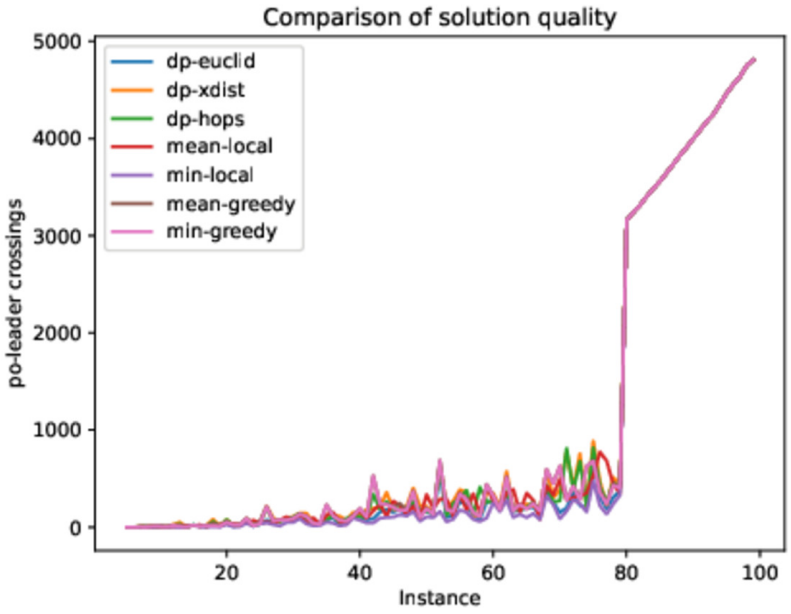


Fig. 9: The observed threshold for the number of crossings with a map width of 80.

References

[BKSW07] Michael A Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff: Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry*, 36(3):215–236, 2007.

[JSK+21] Robin Tobias Jauss, Nadiné Solf, Sree Rohit Raj Kolora, Stefan Schaffer, Ronny Wolf, Klaus Henle, Uwe Fritz, and Martin Schlegel: Mitogenome evolution in the lacerta viridis complex (lacertidae, squamata) reveals phylogeny of diverging clades. *Systematics and Biodiversity*, 0(0):1–12, 2021, 10.1080/14772000.2021.1912205. `https://doi.org/10.1080/14772000.2021.1912205`.

[PSD+21] Da Pan, Boyang Shi, Shiyu Du, Tianyu Gu, Ruxiao Wang, Yuhui Xing, Zhan Zhang, Jiajia Chen, Neil Cumberlidge, and Hongying Sun: Mitogenome phylogeny reveals indochina peninsula origin and spatiotemporal diversification of freshwater crabs (potamidae: Potamiscinae) in china. *Cladistics*, n/a(n/a), 2021, https://doi.org/10.1111/cla.12475. `https://onlinelibrary.wiley.com/doi/abs/10.1111/cla.12475`.