FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Nikola Kalábová

Evolutionary Algorithms for Multi-Stage Transcriptomic Data Analysis

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	Martin Pilát
Study programme:	Computer Science - Artificial Intelligence

Prague 2024

I would like to thank in the first place my supervisor Martin Pilát, for providing many insightful comments during the development of the algorithm and the thesis writing and being cheerful even when I was stressed. Huge thank you to my consultant Hajk-Georg Drost, for helping me with the biological side of the project, welcoming me in his lab and supporting me endlessly ever since. Next, I would like to thank all my friends in Tübingen, who made this place feel like a second home in such a short time. Thanks for keeping me fed during the stressful times. Last but not lease I would like to thank my family for being always there for me.

Title: Evolutionary Algorithms for Multi-Stage Transcriptomic Data Analysis

Author: Nikola Kalábová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Martin Pilát, Department of Theoretical Computer Science and Mathematical Logic

Abstract: Uncovering genes that are involved in development has proven to be a problem, where experimental and in-silico methods often fall short. In this work, we look at development from the perspective of multi-stage transcriptomic data and genomic phylostratigraphy and try to identify a small set of genes that shape the transcriptome age index (TAI) pattern over the developmental stages. For this purpose, we develop a multi-objective island model genetic algorithm GATAI. By exploring the identified gene sets, we show that our algorithm was indeed able to identify genes involved in development. We further generalize our genetic algorithm, to be able to select a minimal subset of a any set of elements, optimizing user defined set of fitness functions.

Keywords: developmental transcriptomics, genomic phylostratigraphy, evolutionary algorithms, multi-objective optimization

Název práce: Evoluční algoritmy pro analýzu multi-stage transkriptomických dat

Autor: Nikola Kalábová

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Martin Pilát, Katedra teoretické informatiky a matematické logiky

Abstrakt: Identifikace genů, které se podílejí na vývoji jedince, je problém, kde experimentální a in-silico metody často selhávají. V této práci se na vývoj díváme z pohledu multifázových transkriptomických dat a genomické fylogenetické stratigrafie a snažíme se identifikovat malou sadu genů, které formují pattern indexu věku transkriptomu (TAI) napříč vývojovými stádii. Za tímto účelem jsme vyvinuli genetický algoritmus GATAI založený na více kriteriální optimalizaci a modelu ostrovů. Analýzou identifikovaných sad genů ukazujeme, že náš algoritmus skutečně dokázal identifikovat geny zapojené do vývoje. Dále jsme náš genetický algoritmus zobecnili tak, aby byl schopen vybrat minimální podmnožinu libovolné sady prvků a optimalizovat uživatelsky definovanou sadu fitness funkcí.

Klíčová slova: vyvojová transkriptomika, genomická fylogenetická stratigrafie, evoluční algorithmy, vícekriteriální optimalizace

Contents

Introduction

Already 200 years ago, scientists noticed that the embryos look very similar during the mid-developmental stages. However, a more rigid approach was required, because ones perception of morphological features is highly subjective. The progress in the sequencing techniques allowed researchers to capture all RNA transcripts in a tissue (transcriptome) at a given time point. It also allowed for phyostratigraphy (study of the age of the genes) to emerge. All these efforts came together in the definition of the transcriptome age index (TAI) [1]. The TAI is able to estimate the average age of the transcriptome for a given developmental stage. This can give us an idea if the evolutionary older or the evolutionary younger genes are active during the particular stage. Indeed, the mid-developmental stages showed lower TAI values, signifying older genes being more active than young ones, than the early and late developmental stages. The evolutionary older genes are common for many species, because they emerged before their differentiation. This supports the morphological similarity of species during the mid-developmental stages. The fact that TAI is able to predict the transition from uni-cellular to multi-cellular development was shown in [1, 2, 3]. Fascinatingly, similar TAI patterns emerge across kingdoms, from bacteria and fungi to plants and animals.

We can go beyond this idea and ask what are the genes that are driving these patterns. We presume, the genes driving the TAI patterns are those which play a key role in development. Hence uncovering those could have a major impact on the developmental biology field, as finding genes which play a key role in development presents an ongoing challenge for scientists. Standard methods often fail because developmental defects often prove to be lethal or such individuals are infertile. Moreover, for lethal mutations, it cannot be established with certainty, that a studied gene indeed plays a key role in development, as it might be, that the gene is simply very high in the regulatory hierarchy and hence might cause off-target effects. The in-silico methods do not yield satisfying results either. Not many methods are known for completely unknown genes without any homologs (genes with high sequence similarity) with a known function, that could at least identify candidate genes for a further analysis.

We make an assumption that the genes that are creating the TAI pattern are the ones that when removed from the dataset, the entire pattern is destroyed. However, finding just a small set of such genes between tens of thousands of genes in the dataset poses a hard problem. A similar problem was tried to be solved in [4] with very limited success. However, we presume that by choosing a different approach, we might come up with a method strong enough to dissect this problem. Hence, the main goal of this thesis is to explore available methods that can identify a small subset of genes that define the TAI pattern and select and optimize the best preforming one of such methods. After having those genes at hand, we would like to see, if those are conserved across the tree of life or at least for closely-related species.

Having such an optimisation method for gene removal to *destroy* TAI patterns at hand, we want to test, if our method can be applied to other problems, which require to identify key drivers of a pattern created by computing a non-linear summary metric on a set of elements over multiple (time) stages. Furthermore we aim to test if our method can be used for a wide range of set minimization or maximization problems, where the target set is given by set of arbitrary optimization functions.

1 Evo-Devo transcriptomics

To understand the biological methods and techniques we used to uncover the developmentally important genes, we first present a short introductory overview.

1.1 Transcription

In the central dogma of molecular biology, transcription is a key step in the process of transforming segments of DNA (molecular information) into proteins (molecular function). For that reason, transcription can be seen as the expression of an organism's genome into a molecular phenotype. More specifically, transcription is the process of transcribing segments of double-stranded DNA to single-stranded mRNA (messenger RNA). These mRNA transcripts are then further translated to proteins or act as non-coding RNA, which appear to control gene expression in physiology and development [5].

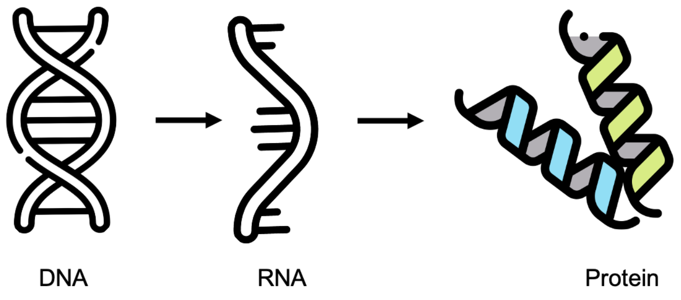


Figure 1.1 Central dogma of molecular biology. DNA gets transcribed to RNA which is translated to proteins.

1.2 Transcriptomics

As the name hints, transcriptomics studies the transcriptome, which is the set of all RNA transcripts on a given time point. In developmental biology, transcriptomics captures which genes are highly active (expressed) during specific events happening in the tissue and thus might be responsible for those. Its necessary to mention that for protein-coding mRNA, there is not a one to one correspondence between the abundance of transcripts of a gene and the amount of protein produced, because there are other factors, such as the mRNA stability, localization or degradation-inhibiting sequence attachment, which influence how long will the transcript persist in the cell and be transcribed and also how often it will be transcribed. Measuring the gene expression (abundance of transcripts of given genes) is preformed by extracting the RNA from a tissue, filtering for mRNA, amplifying the number of mRNAs for sequencing methods to be able to capture it, and finally sequencing those mRNAs [6].

1.2.1 Bulk and single-cell transcriptomics

For around 35 years, molecular biologists have been able to apply high-throughput technologies to sequence RNA transcripts in bulk [7]. That meant taking all transcripts found in a particular tissue, containing different cell types and sequencing all the transcript that were found. As the technology progressed, around 15 years ago, researchers were able to preform the first single-cell sequencing [8], which enabled looking at the expression on a level of one particular cell of a particular type at a particular time.

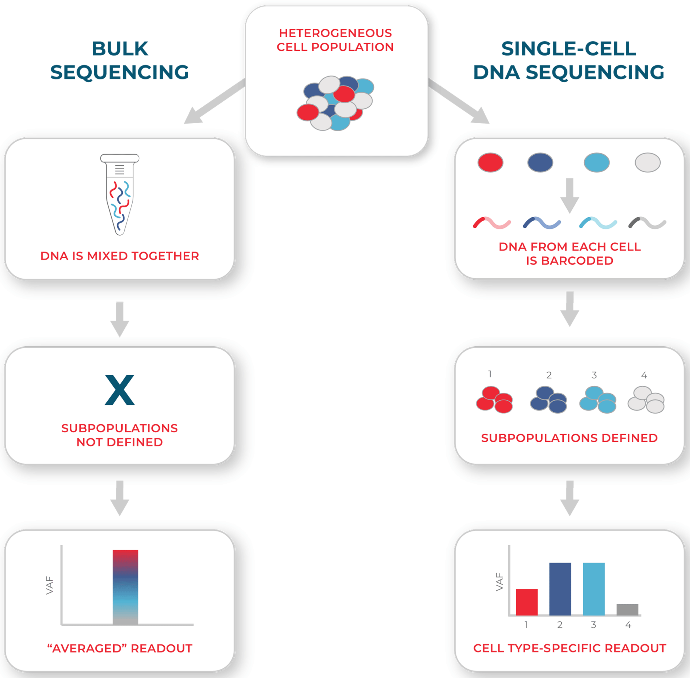

BULK SEQUENCING
HETEROGENEOUS CELL POPULATION
SINGLE-CELL DNA SEQUENCING
DNA IS MIXED TOGETHER
DNA FROM EACH CELL IS BARCODED
X
SUBPOPULATIONS NOT DEFINED
1
2
3
4
SUBPOPULATIONS DEFINED
VAF
"AVERAGED" READOUT
VAF
1
2
3
4
CELL TYPE-SPECIFIC READOUT

Figure 1.2 Single cell vs. bulk sequencing. Source: missionbio.com

1.3 Genomic Phylostratigraphy

Genomic Phylostratigraphy is a computational alignment method based on evolutionary assumptions to trace back genes to their point of origin within the taxonomy of life and assign them a putative origination age (the lowest detectable taxonomic node which is referred to as phylostratum), relative to all other genes [9]. To infer a relative gene age, the protein sequence of the gene is compared against a sequence database to find the most distantly related sequence hit which could potentially be a homolog (gene with shared ancestry, thus similar sequence). Moreover, homologs throughout all the nodes between the studied species and the most distant homolog are taken into account and a gene age is assigned only if we can trace homology across the whole path in the phylogenetic tree from the studied species to the most distant homolog [10].

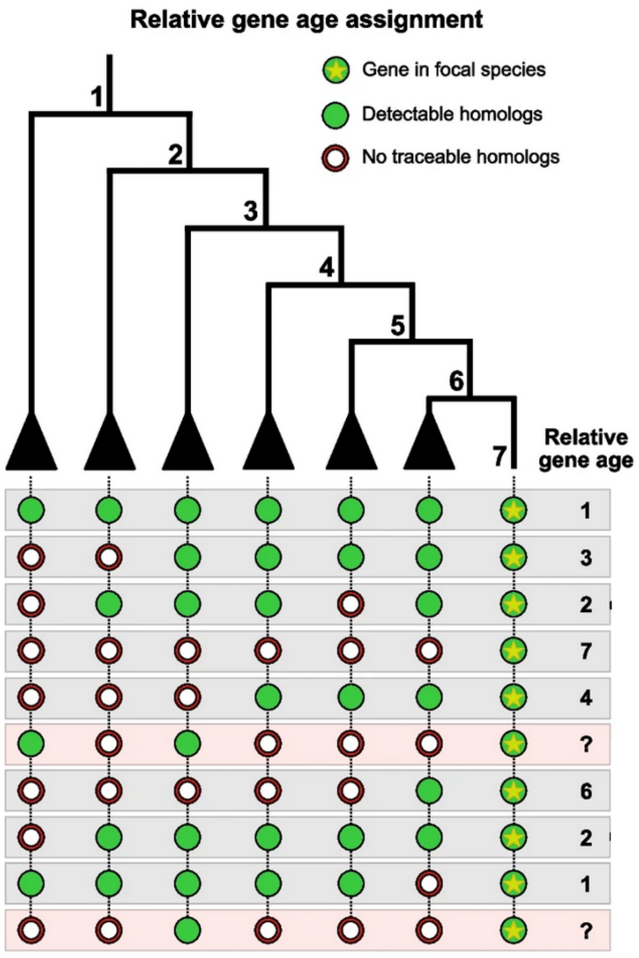


Figure 1.3 Gene age assignment. For each gene, homologs are searched for on every node of the phylogenetic tree. The resulting age is the minimal node value, where homology was detected, if homologs are detected on most of the nodes with higher value. Source: [10]

1.4 Transcriptome age index

Once the expression of the genes and the gene assignment for every gene in a dataset is known, the average age of the transcriptome can be estimated. This approach reveals, whether mainly evolutionary older or evolutionary younger genes are utilized at a given developmental stage. Such a summary metric can be computed as a transcriptome age index (TAI), as proposed in [1], where the TAI is the average age of the genes in the dataset, weighted by their expression.

$$TAI_s = \sum_{i=1}^{|G|} p_i \cdot \frac{e_{is}}{\sum_i^{|G|} e_{is}} \tag{1.1}$$

Where:

TAI_s	is the TAI value at developmental stage s
p_i	is the gene age (phylostratum) of the gene i
e_{is}	is the expression level of gene i during the developmental stage s
G	gene set

When computing TAI over multiple developmental stages, a TAI pattern is obtained. This pattern identifies developmental stages in which the evolutionary younger genes are more transcribed and stages in which the evolutionary older genes are more transcribed. The highly expressed genes potentially have higher influence on the resulting phenotype.

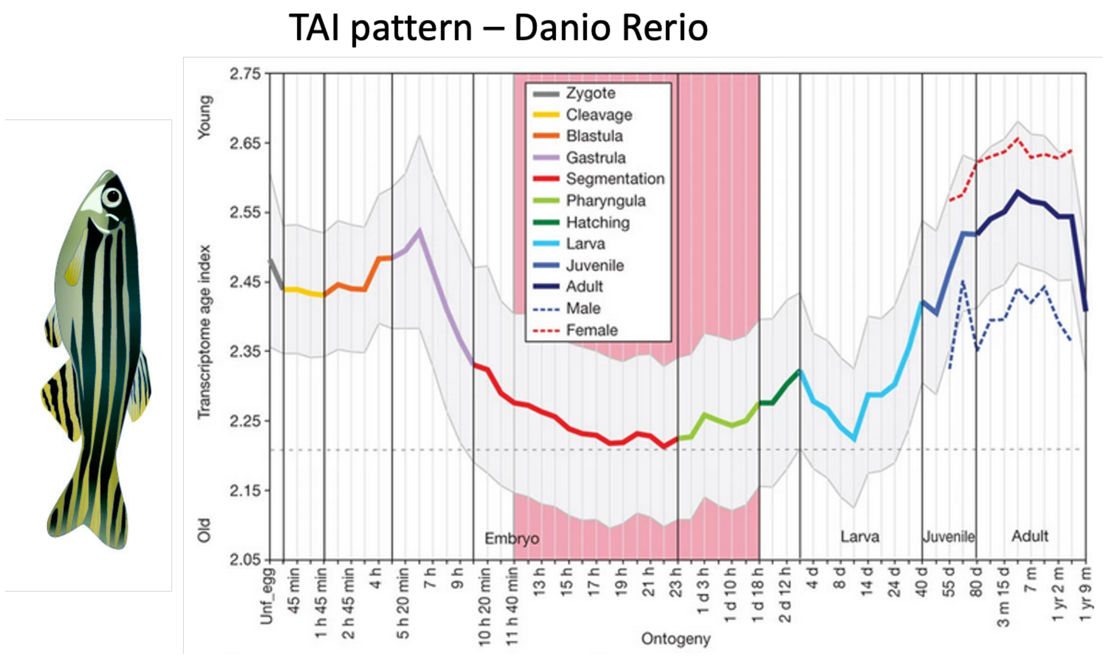


Figure 1.4 Transcriptome age index over the whole life cycle of Danio Rerio. Younger genes are more expressed in the early and late stages puling the TAI up, older genes are more expressed in the middle stages, lowering the TAI. Source: [1]

1.4.1 Significance of the TAI pattern

Before starting to infer anything from this pattern, we need to first test its statistical significance compared to a pattern obtained from a random gene age assignment. For this purpose, the flat-line test was developed [11, 12].

Flat-line test

To compute the flat-line test, first the phylostrata are permuted and randomly assigned to genes. Afterwards, the TAI pattern is recomputed with the newly assigned phylostrata and the variance is measured. This is preformed enough times to get a representative distribution of variances. At the end, the parameters of a gamma distribution are estimated. Having now a distribution at hand, the p-value of the variance of the original TAI pattern being drawn from the gamma distribution can be measured. If the p-value is not significant, the variance was most likely obtained by chance. If it is on the other hand significant, there is a high probability, that the pattern didn't occur by chance and is worth looking into.

After preforming the flat-line test, we can safely say for most of the studied datasets, that the observed pattern is indeed significant and the high variance is not observed purely by chance.

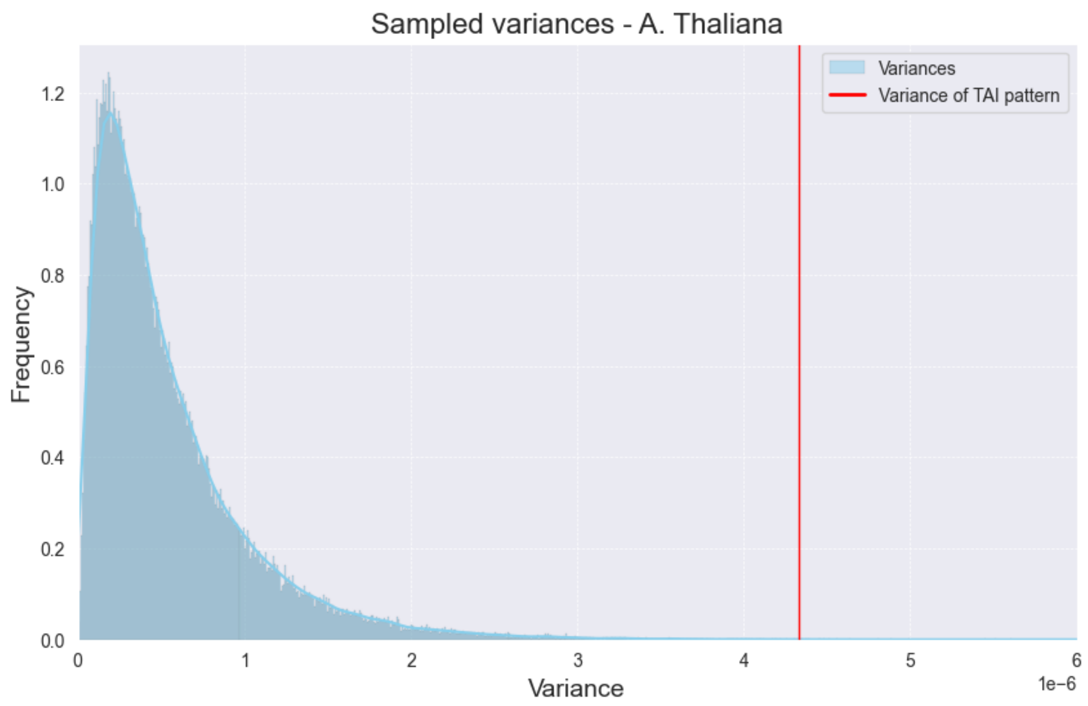


Figure 1.5 Histogram of variances of TAI patterns obtained by permuting the gene ages and recomputing TAI for every developmental stage. Gamma distribution is estimated from the sampled variances and p-value of the variance of the original TAI pattern (red) computed.

2 Identification of development-driving genes

Assigning a function to a gene remains a hard problem for biologists. Finding genes involved in development poses additional challenges mentioned below. Only through tedious experimental designs new function can be revealed which remains a slow and not easy to scale procedure.

2.1 In-vivo methods

In-vivo refers to methods and experiments that are performed or take place within living organisms.

2.1.1 Genetic screens

One option to determine the putative function of genes involved in development is by inducing mutagenesis in individuals of a population. That means forcing a high mutation rate in an organism by genetic engineering, chemicals or radiation [13, 14]. Afterwards, individuals are mated, their offspring sequenced and individuals with developmental defects are compared to the wild type.

2.1.2 Knockout, knockdown or overexpression

To test if a gene is indeed active in development, it can be knocked out [15, 16] or knocked down [17, 16], that is permanently or temporarily inactivated. With the introduction of the CRISPR technology [18, 19], these methods gained on precision and scalability. A developmentally defect phenotype in an individual suggests that there may be a key role of such gene in development. Recent methods made it also possible to knock out a gene in a tissue-specific or time-specific manner, to study its role during particular stages of development [20].

Another way to test a specific gene is to cause its overexpression by inserting more copies of such gene into the organism [21]. Afterwards screening for altered growth, morphology, and timing of developmental stages is performed.

2.1.3 Problems of standard approaches

To create a living system robust to mutation effects, many developmental processes involve redundant genes, where multiple genes can compensate for the loss of one another, making it difficult to identify critical genes through knockout or knockdown screens. Moreover, genes involved in development often have multiple functions (pleiotropy), and disrupting them can cause a range of unrelated phenotypes, complicating the interpretation of results. Third problem is that interactions between genes (epistasis) can obscure the effects of individual mutations, making it challenging to determine the role of a specific gene [22].

Many mutations in developmental genes also result in embryonic lethality, preventing the study of later stages of development or adult phenotypes. Embryonic lethality also causes some developmentally defect phenotypes where the true function can never be observed and thus genes behind such defect are hard to study. There is also a sizable number of mutants, which are able to live but fail to procreate, complicating breeding and further genetic analysis [23]. Genes mutated in those individuals hence remain undetected and unannotated with their respective function in development. If a gene set is enriched in an unknown function, it might be hence development related.

2.2 In-silico methods

2.2.1 Transcriptome analysis

Another way to uncover genes involved in development is to study expression profiles over multiple developmental stages and identify genes that are differentially expressed between stages [2]. Unfortunately, simulating the same environment every time the gene expression is measured is deemed impossible, moreover tested organism are often killed before taking their tissue samples, hence the data do not even come from the same organism for all the developmental stages. For this reason, many genes are differentially expressed because of some external or internal events unrelated to development.

Another option is to study gene co-expression. Co-expressed genes are genes that follow a similar temporal expression profile and thereby might be regulated by the same regulators and thus contribute to relevant developmental and other pathways [24]. With this assumption the genes are clustered by similar expression profiles and claimed to possibly be involved in development if some of the co-clustered ones are experimentally known to be involved in development [7]. The new candidate genes can then be tested by some of the methods introduced above. However, for such approach it is necessary for the expression data to span a large number of developmental stages, for the correlations to be considered significant [25].

2.2.2 Cross-Species Comparison

Given that a closely-related species with more detailed gene annotations exists, candidate genes can be identified by searching for genes in the target species, that are orthologous to genes (sharing common descent) that have been proven to be involved in development in the closely-related species [26].

3 Data

To guide the search for developmentally important genes, we would like to develop a method, that could preselect possible candidate genes given a common and easily accessible omics (genomic, transcriptomic or proteomic) method. We would also like such method to be usable for a broad range of different species across the tree of life.

3.1 Input data

In general, having as much information from different omics experiments would give us stronger tools to identify the genes that play a key role in development. However, having a method that does not require data from multiple potentially expensive and time-consuming experiments broadens the potential applicability of a method. It also enables the usage of already present datasets on which different downstream analyses have been done. This way, previous knowledge can be combined with new findings, to gain deeper insight.

For this reasons we focused on inferring the key regulators of development solely from the gene expression data over multiple developmental stages, alongside the phylostrata (gene ages). The phylostrata can be inferred from the gene sequences, thus no additional information apart from gene expression and protein sequence is required.

The input data are thus a tsv file with first two columns being the phylostrata and the gene ids, followed by columns with the measured expression in the particular developmental stage for the respective genes.

3.2 Collapsing replicates

Many datasets contain replicates for the developmental stages. These need to be collapsed meaningfully for every developmental stage. Following the example of `myTAI` [11], we computed the geometric mean for all replicates belonging to the same developmental stage. The geometric mean is a good way to combine multiple measurements of the same event because it handles well multiplicative effects common in biological data, reduces the impact of outliers and accurately reflects the average for proportional changes. These properties of the geometric mean aggregation hold only for untransformed data, hence the replicates should be collapsed before further transforming the dataset.

3.3 Signal prefiltering

We found that a lot of genes have close to zero expression over the developmental stages. Such genes are very unlikely to be genes that play a key role in development. However, they make the search space in which we will be searching for genes involved in development larger. For this reason we first remove those genes that are lowly expressed in all stages, that is their expression is close to zero in every developmental stage.

3.4 Data transformations

The datasets were obtained from various developmental expression studies. Those studies used different methods to measure the expression over multiple stages. In some studies the microarray, in others the RNASeq technology was used. For that reason, different transformations needed to be used and due to the different data sampling methods, the results might not be comparable between species.
For the RNASeq data, we transformed the raw data to Transcripts Per Million (TPM). TPM is a normalization method for RNA-Seq data that accounts for sequencing depth and gene length. It involves normalizing read counts by gene length (RPK), then scaling these values so that the sum of all TPM values in a sample equals one million, which enables an accurate comparison of gene expression levels between samples.

In gene expression studies, expression data are often first transformed to either log scale or square root scale to achieve variance stabilization. This might however make the identification of pattern driving genes harder as these transformations tend to remove strong signals and make the expression patterns of genes more uniform. Such transformations might also remove some important biological signal. On the other hand, such transformations might be beneficial for very noisy datasets, where just a handful of genes dominate the overall expression and variance of the whole dataset. We will discuss the influence of different transformations in following chapters.

The raw phylostrata values represent an ordinal scale, without capturing the actual phylogenetic distance between species within and across taxonomic nodes. This creates an analysis bias, due to the absence of uniform intervals between values, making arithmetic operations hard to interpret meaningfully. Thus we applied a quantile-rank transformation to the phylostratum assignments of genes. This allows us to convert ordinal phylostratum assignments into a continuous numerical scale, where arithmetic operations are meaningful. Yet, the true phylogenetic distance between species within and across taxonomic nodes is still not captured.

4 Problem setting

To be able to decide for a suitable algorithm to uncover developmentally meaningful genes, we first need to exactly define the requirements, we set on the algorithm and identify possible problems.

4.1 Assumptions

Having not much information at hand, we decided to try to infer developmentally interesting genes from the TAI pattern. Given a TAI over multiple developmental stages, as presented in the previous chapter, our task is to identify genes, that are driving this pattern. Here, we make a bold assumption, that such genes are those, which if removed from the dataset, make the pattern disappear. We presume, this way we might actually be able to extract valuable information because of the statistical significance of the TAI pattern as shown in the previous chapter. One could think that we achieve the pattern destruction simply by removing the most variant genes from the dataset, which would clearly not bring much biological insight. In following chapters we show that this approach would not work if we aim to remove just a small subset of genes.

4.2 Optimization criteria

We define two optimization criteria, that need to be optimized in parallel and a good trade-off between these criteria needs to be found.

4.2.1 Optimization Objective 1: Pattern significance

Given the fact that TAI patterns can have various shapes and are sometimes defined just over a few developmental stages, it is hard to say what is a non-random pattern and thus hard to define what it means for a pattern to be "destroyed". That is why, instead of some more intricate metrics of an overall pattern shape, we simply consider the variance, which have been shown by the flat-line test to be significant for TAI patterns. Thus, inspired by the flat-line test, we optimize for a low variance of the TAI pattern. Let the average TAI over stages after removing a subset of genes be defined as:

$$E(\mathrm{TAI}_{s,I}) = \frac{\sum_s (\sum_{i \in G \wedge i \notin I} p_i \cdot \frac{e_{is}}{\sum_i e_{is}})}{|S|} \tag{4.1}$$

Where:

I	is the set of indices of the removed genes
$TAI_{s,I}$	is the TAI value in developmental stage s after removing genes on indices in I
p_i	is the gene age value (phylostratum) of the gene i
e_{is}	is the expression level of gene i during the developmental stage s
G	gene set

Then the variance after removing genes with indices $i \in I$ is computed as:

$$\mathrm{var}_I = E(\mathrm{TAI}^2_{s,I}) - E(\mathrm{TAI}_{s,I})^2 \tag{4.2}$$

Such optimization criterion would be easily pushed to 0 simply by removing all the genes. However, this is clearly not the solution we are looking for. In general, larger sets of genes could remove more variance, however might include genes, that are not contributing much to the overall variance decrease. Thus, we introduced a second optimization criterion which is the size of the identified gene set.

Minimal variance

Without removing all genes from the dataset, the variance will never reach 0. Moreover, not even random pattern have close to zero variance. Thus we need a good estimate for the target variance. This will prevent wasting computational power and time on trying to push the variance lower than necessary.

To estimate a target variance, we once again utilize the idea of the flat-line test. Instead of minimizing the variance directly, we developed a metric inspired by p-value.

In the original flat-line test, the phylostrata are permuted, and the variance is computed from the TAI pattern. After a large number of permutations, a distribution of variances of TAI is obtained. For such distribution, the parameters of a gamma distribution are estimated. Afterwards, the p-value of the original TAI pattern variance is computed from that gamma distribution. During our experiments we found that gamma distribution does not approximate the distribution of variances well enough.

For the optimization, not a proper statistical measure is needed. Some optimization methods can work with a function, that just assigns order to the candidate solutions. Other require some measure of how much a solution is better than another. However, usually not a high precision is required.

To make the optimization function simple and easily interpretable, we decided to simply take the ratio of the sampled variances that are smaller than the optimized one to the total number of sampled variances. This creates an approximation of a one-tailed p-value test. We call the resulting metric an empirical p-value. During the optimization, we would like to maximize the empirical p-value, because we want the resulting variance to be in the range or the randomly sampled variances.

$$p(k) = \frac{1}{|V|} \sum_{v \in V} \delta_{v \geq k} \tag{4.3}$$

Where:

k	is the value we want to estimate the p-value for
$\|V\|$	is the total number of variances of the random TAI patterns
V	is the set of variances of the random TAI patterns
$\delta_{v \leq k}$	is the Kronecker delta function

p-value relaxation

Rigorously, we would need to recompute the sampled variances from a dataset created by removing the candidate genes, to test if the resulting TAI pattern is indeed insignificant. However, this would be computationally very demanding. Given the fact, that our goal is to identify just a small subset of genes, we assume that the sampled variances computed from the data after removing the candidate genes, do not differ significantly from the variances sampled from the original dataset. That is why during the optimization, the variances sampled from the original dataset will be used.

However, after the optimization terminates, the variance should be sampled from the dataset with the identified genes removed and the statistical significance should be recomputed to confirm that the resulting pattern has indeed become insignificant also when the sampled variances are computed from the dataset with the identified genes removed.

4.2.2 Optimization Objective 2: Number of identified genes

Removing a higher number of genes generally leads to a lower variance. Thus, to satisfy the first optimization criterion, we could simply remove a large portion of the dataset. Clearly, this would not be very informative. That is why we try to minimize the number of identified genes. This will ensure that only the genes, that influence the TAI pattern significantly are captured. That is why we choose the second optimization objective to be the number of identified genes, which we would like to minimize.

4.3 Optimization limitations

These optimization criteria have a couple of limitations. The first limitation lies in the computation of the pattern itself. The transcriptome age index (TAI) is a very simple metric (summary statistic), hence it might not be able to capture fine relationships between genes. Measuring the significance of the pattern simply by its variance is also a major simplification. Unfortunately, since some of the datasets have as little as 7 developmental stages, we weren't able to come up with a metric that captures the randomness better.

Given the fact that we are trying to find a minimal set of genes that reduce the variance of the overall pattern the most while maintaining biological relevance, we are introducing a bias towards highly variant, thus highly expressed genes. Highly expressed genes are usually located at the lower levels of biological pathways (are regulated by a long cascade of regulators and perform a highly specialized function). However, we would like to capture key regulators across all levels of pathways, ideally all the way up to lowly expressed transcription factors (protein controlling the rate of transcription).

4.4 Additional criteria

Since we already know that the TAI metric is sub-optimal and a different metric might be needed for single-cell data, developing an algorithm that can easily work with a different optimization function seems important. Moreover, the number of genes that should be identified is not known beforehand. This number might as well vary significantly across datasets and so far not a good heuristics is known. For those reasons, having an algorithm that is able to determine the optimal number of identified genes would pose a big advantage.

5 Optimization strategies

A first naive idea would be to construct some iterative greedy algorithm. However, especially for a higher number of stages, the relationship between the expression levels of the genes are highly intertwined. Thus, trying to push the TAI closer to the average TAI for one stage might result in increasing the variance between other stages. Another idea would be to try removing random sets of genes. However, since the datasets tend to have around 30000 genes, randomly stumbling upon a good solutions is improbable. If we assume to extract at most 300 genes, the number of different combinations is:

$$\text{num\_combs} = \sum_{k=1}^{300} \binom{n}{k} \tag{5.1}$$

Where:

n is the number of genes in the dataset
k is the number of genes to select

For this reasons a more sophisticated method needs to be invented.

5.1 Linear or constraint programming

For linearly encoded constraints and equalities we could use a linear optimization solver or a constraint solver. The first problem with this approach is, that our problem, as it is defined now is not linear. While we would possibly be able to redefine the problem to a similar problem with linear constraints, it would be hard to later add arbitrary additional constraints if necessary. Another problem is, that we would need to bound the number of extracted genes before every run, without any knowledge on what the bound should be. Hence, we would need to run the optimization multiple times with different bounds on the number of genes. Another way to extract a reasonable trade-off between the two criteria would be to introduce a hard constraint on the quality of the solution.

In general, we would miss the opportunity for a post-hoc selection of a suitable trade-off between the number of extracted genes and the quality of the solution.

5.2 Simulated annealing

As presented in [4] another option to consider is to use simulated annealing [27]. This could be done by generating a random solution, slightly modifying it in every iteration and accepting the modification based on the temperature, according to the simulated annealing algorithm. Due to the wide and complex optimization landscape of this problem, such optimization method might easily get stuck in local optima and might not be able to converge. In [4] a similar, yet very much simplified problem was being solved with suboptimal results. The method was suitable only for patterns that can be approximated by a parabola and had many other constrains.

5.3 Graph neural networks

Graph neural networks[28] are neural networks used for regression or classification working on graph structured data. Using GNNs might seem not intuitive at a first sight, because our problem as defined is inherently not a graph problem. However, we can speculate that whether a gene is selected, highly depends on other genes with similar expression pattern, or an opposite expression pattern, because such genes could be co-regulated. From such relationships, we can construct a graph comprising of nodes and edges to learn on. We can then get a classification for every node in the graph, hence for every gene, if it belongs to the development related genes or not. One pitfall might be the contradicting objectives, that need to be combined into one loss function. There are two clear and possibly deep local optima, that the algorithm can easily get stuck in. One being simply selecting all genes to achieve zero variance and the other one being selecting no genes to achieve the minimal possible number of selected genes. Again, the result would be just one solution, hence we cannot post-hoc select the best trade-off between the two objectives.

5.4 Evolutionary algorithms

Evolutionary algorithms[29] were designed to hold many candidate solutions at once and to combine them in a way that can generate better solutions. They also offer a simple encoding and high freedom for the optimization function. We have a simple binary encoding at hand for our problem, which signifies the presence or absence of a particular gene in the solution. We can then optimize the solutions either by a simple genetic algorithm with a fitness function being a weighted combination of the two objectives, or we can utilize a multi-objective optimization. With this approach, the fitness function is easily exchangeable and if we opt for the multi-objective optimization approach, we can select the best trade-off post-hoc. Given all these advantages, we decide to optimize the two objectives presented above using evolutionary algorithms. If we find a better metric than TAI in the future, we will be able to easily swap them. Moreover, with multi-objective optimization, we can decide if we want to identify a possibly larger amount of genes, that make the pattern almost flat, or just a couple of genes that play a key role in the TAI pattern, while removing just part of the overall variance of the TAI pattern.

6 Evolutionary algorithms

Evolutionary algorithms [30] take inspiration in the evolutionary processes in nature. Just as in nature, they keep a pool of individuals (solutions), which are then mated between each other, mutated and their offspring undergo selection and only the strong ones make it to the following generation. An individual usually consist of an array of values, where every element of the array usually corresponds to one variable that we want to optimize.

6.1 Simple genetic algorithm

For a problem that allows for a binary encoding a subclass of evolutionary algorithms called genetic algorithms can be utilized. The simplest genetic algorithm proceeds as follows. First a population of solutions is generated. Then random pairs of solutions are crossed over and the new individuals are mutated. This gives a rise to a new population. For this population a fitness function for every individual is computed. A final population that enters the next generation is selected based on the fitness from either these solutions or also the parental solutions are included [30].

An evolutionary algorithm optimizes a fitness. A fitness defines how good the solution is. The fitness function doesn't have any limitations on the function type, but it needs to be computable in a short time, as it will be computed for many solutions over many generations. Usually the fintess is maximized, however the problem can also be defined in a way that the fitness is minimized.

For classic evolutionary algorithms, two mating operators are defined. First is mutation, which takes a random part of the individual and randomly changes their values. For genetic algorithms, usually a bit-flip mutation is chosen, which takes random indices in the solution array and flips the bit value on those positions.

Second is a crossover. An idea behind a crossover is to select two good solutions and combine them so, that possibly an even better solution is created as consequence. One option is to use a uniform crossover. Just as for bit-flip mutation, random positions are selected and the values on those positions in the two solutions chosen for crossover are exchanged.Another option is to use a one-point crossover, where one index is picked on all position higher than this index, the respective values are exchanged between the two solutions. Similarly, we can define a two point crossover, where the positions between the two crossover points are swapped.

Selection is a procedure where good solutions are retained from either the current generation, or also the parental generation. Not only the fitness but also the overall diversity of the next population can be a criterion. A slightly worse solution can be included to promote diversity.One selection method, that can be used is the roulette-wheel selection, where the probability of a solution to be chosen to appear in the next generation is proportional to its fitness. In this way, a predefined number of solutions can be sampled.Another way to select solutions for the next generation is to use a tournament selection, where in each round two solutions are chosen randomly and the one with higher fitness gets to the next generation.For problems with multiple objectives a multi-objective selection can be used. Here, the selection should ensure that good solutions with different trade-offs between the objectives are selected.

6.2 Multi-objective optimization

For problems with two or more objectives, we can utilize multi-objective evolutionary algorithms (MOEA), which keep a set of solutions with different trade-offs between the objectives. Usually, we are interested in solutions on the Pareto front. A Pareto front consists of so called non-dominated solutions. Those are solutions, that are better than every other solution in at least one objective.

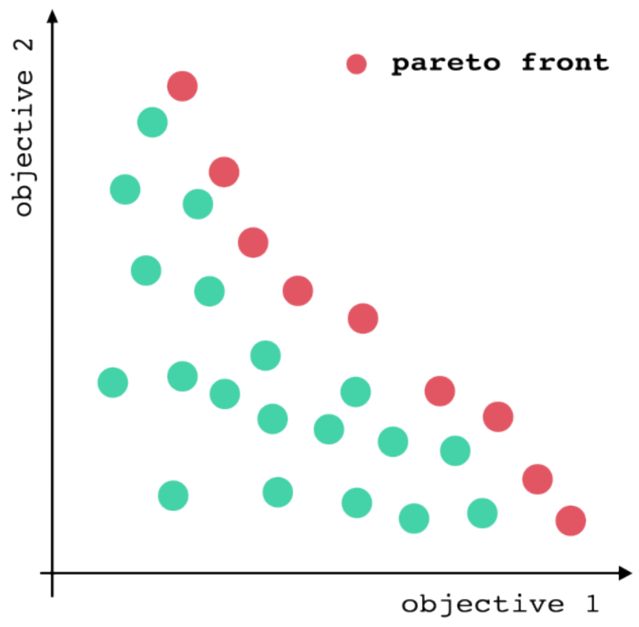


Figure 6.1 Pareto front of non-dominated solutions (red dots), where both of the objectives are maximized. All of these points have higher value in an objective than all other solutions

One of the oldest and most used MOEA algorithms is the Non-dominated Sorting Genetic Algorithm II (`NSGA2`) [31]. This algorithm first sorts the solutions into Pareto fronts, where the solutions in the second Pareto front are those which would be non-dominated after the removal of the first Pareto front. Analogically for the following Pareto fronts. Afterwards, it fills the next population with pareto fronts one by one starting from the first Pareto front. If a whole front doesn't fit into the next population, the solutions are ranked by their crowding distance and just the amount needed to fill the next generation is selected based on increasing crowding distance. The crowding distance is computed by sorting the solutions in all objectives and for every index i computing the distance between solution $i - 1$ and $i + 1$ and taking the average over all objectives.

Later, its new version `NSGA3`[32] was published. `NSGA3` takes reference points as an argument, where each reference point specifies the importance of one objective over the others. All those points lie on a hyperplane, where the sum of their coordinates is always equal to 1 and the value for each coordinate is greater or equal to 0. For example point (0.5,0.5,0) would try to find a point, that completely disregards the third objective and gives equal importance to the first two.

The main difference between `NSGA2` and `NSGA3` is the selection of the solutions from the first Pareto front that didn't completely fit into the population of the next generation. For `NSGA3`, the solutions already selected to the next generation are associated to the closest reference point. For each reference point the number of associated solutions is computed. Each solution from the last Pareto front (the one that didn't completely fit into the next generation) is then also associated with a reference point. Until the next generation is filled a reference point with the least amount of associated solutions (those already selected to the next generation) is picked and a random solution from the last Pareto front, that is associated with that reference point is added to the next generation. In this way the next generation is populated.

Another widely used algorithms is (`SPEA`)[33]. The idea of Strength Pareto Evolutionary Algorithm (`SPEA`)[33] is having an externally stored all-time Pareto individuals that are updated every generation. Moreover, to not fill-up the whole population just by non-dominated solutions, `SPEA` reduces the number of Pareto individuals by performing a clustering algorithm on those and selecting just one centroid for each cluster. If the next population is not filled with the non-dominated solutions, all current solutions are given a ranking. The Pareto-solutions in the current generation are ranked by how many of the all-time Pareto solutions they cover (are better in all objectives). The dominating solutions are ranked by how many solutions from the all-time Pareto set covers them, where less is better. By this metric the solutions are then selected to the next generation. Looking at the coverage instead of just a pure fitness in all objectives makes the solutions more distributed over the optimization space, thus enhancing diversity.

7 GA for subset minimization

We realized that selecting a minimal subset of genes which are involved in temporal development from a set of genes based on an optimization function can be easily generalized to selecting a subset from a set of elements based on an arbitrary set of optimization functions. To make our tool as general as possible, we developed a general subset minimizer, which minimizes the cardinality of the selected subset as one objective and optimizes a set of further objectives given by the user.

After developing this tool, we simply input the set of genes and the variance minimizing optimization function and we get our target algorithm. Details are described in the following chapter.

Our solution is based on the `DEAP` library [34]. `DEAP` provides a simple framework and a plethora of pre-made crossover, mutation and selection functions alongside full evolutionary algorithms.

7.1 GA architecture

First, we implemented a simple genetic algorithm with uniform crossover and bit-flip mutation. We implemented the fitness to minimize as the sum of the objectives. We tested this approach on our transcriptomic problem. The weighted fitness yielded fairly good and fast results for most of the non-transformed dataset, but for the log and square root transformations this solution was rather slow and sometimes struggled to converge. We also found that the solutions are not very stable between runs. This we tried to solve by utilizing multiple solutions with a good fitness from one run and combining the results. However, we found that those solutions are almost identical, thus don't cover the space of possible good solutions very well. To tackle all above mentioned problems, we decided to use a multi-objective optimization strategy. This allows us to select solutions in a particular section of the Pareto front. Moreover, such approach enables us to easily add additional constrains and optimization objectives in the future.

7.2 Libraries used

We started with the `pyGAD` library [35]. Once we found, that using a multi-objective optimization approach could be advantageous, we switched to `pymoo` [36] which is a Python library optimized for multi-objective optimization (MOO). While having now a wide range of multi-objective selection operators at hand was a big benefit, we otherwise found the library to be rather slow. That is why we finally switched to `DEAP`[34] that had some multi-objective selection operators pre-implemented as well and had proven to be a fast, general and user-friendly interface for our problem. Though we did not find a good enough implementation of an island model and early stopping. Nonetheless, for implementing those most of the `DEAP` code could be reused and only slightly extended.

7.3 Initialization

We kept the encoding simple, with 1 on index i if i-th element of the set belongs to the selected set and 0 otherwise. Since we want to minimize the size of the selected subset, we initialize the solutions to have just a small amount of values set to 1. The fraction of the total number of elements, that should be set to 1 in the first generation is a tunable hyperparameter. The user can also input their custom function that creates an individual. This function is then used to create all individuals.

To define the set of the fitness functions a user defined set of fitness functions is taken and the set size is added as the first parameter. By default all parameters are minimized, but the user can specify which objectives to minimize and which to maximize by passing a tuple of -1 and 1 with length corresponding to the number of objectives as optional argument min_max,, where -1 on index i means the i-th objective will be minimized and 1 on index i means the i-th objective will be maximized. This follows the standard minimization/maximization definition for the DEAP library. SetGA then takes this tuple and adds a -1 on the beginning to signify, that the size of the identified subset should be minimized.

7.4 Island model

We found that in order for the final solution to be robust enough, we need to aggregate solutions from various niches and positions on the Pareto front. While pareto front, novelty[37] or niching[38] based algorithms are partially able to find diverse solutions, we found that we can achieve better results faster with an island model. An island model is a model where multiple independent populations of solutions are optimized at the same time and once every n generations, some solutions are moved from one island to another. This approach enhances diversity of solutions during the optimization as different islands can cover different parts of the optimization space. Moreover, the islands still share information so that a good solutions combining two niches can emerge. While `DEAP` supports some form of island model, it didn't seem general enough for our purposes. That is why we modified the original algorithm that preforms the general GA cycle to suit the island model better. We allow for a custom number of islands. Those are also able to have different mutation types and rates. The information between islands is shared by ring-migration every x generations. Where x is a tunable hyparparameter.

7.5 Mating

For crossover and mutation we allow all functions that would make sense for a binary EA. For mutation, that is bit-flip mutation and inversion. A user can also define their own mutation, or pass a list of different mutation functions or names of mutations functions (bit-flip or inversion) of the same size as the number of islands. This way, different mutations can be defined for different islands.

For crossover we allow the user to choose from uniform, one-point, two-point, partially matched, ordered and uniform partially matched crossover. All these function were taken from the `DEAP` library. We also allow for a custom crossover function to be given.

For selection, we let the user choose from `SPEA2`, `NSGA2`, `NSGA3`. We recommend the user tries the `NSGA3` first, because the `NSGA3` is the fastest of these three. We by default distribute the k reference points uniformly across the space, where k is the number of solutions in an island divided by 10. The reference points can also be passed to the tool as an argument.

7.6 Logging

Every 10 generations, statistics for the current generation are stored into the logbook for every island and if the verbose parameter is not set to `False`, printed out. The user gives the names of the objectives as an argument. Given statistics are printed only of one solution from every island, the user defines based on which optimization objective, the current best solution will be selected. The logbook is at the end returned to the user.

7.7 One generation

We took the `eaMuPlusLambda` algorithms as implemented in the `DEAP` library and modified it so that it suits the island model and allows for early stopping, if the model converged. First the next generations population is created by calling the `VarOr` algorithm as its implemented in `DEAP`, only enabling different mutation functions for different islands. Afterwards, solutions are selected from the new population and the parental population with a chosen selection algorithm. If the values of the first user specified optimization objectives did not change for n generations, where n can be specified by the user, the optimization is stopped and the current population returned. Every 5 generations, some solutions are migrated from one island to the next one.

7.8 Solution aggregation

At the end of the optimization, the whole population of all the islands is returned and can be further post-processed. We implemented a function, that will take all the solutions from all the islands, filter out the ones, with a too low fitness according to custom criteria and then take all the elements of the set that have strong enough representation throughout all the good solutions.

7.9 Usage

Using the minimizer is straightforward. The library is available on `pypi` and can be installed by calling `pip3 install setga`. The minimizer is then called from python by `setga.select_subset.run_minimizer()`. The mandatory parameters are only the number of elements in the set, the fitness function to be minimized, the names of the objectives and the number of the objective to pick the best solution according to for logging. The complete documentation can be found in the github repository[1].

[1]https://github.com/lavakin/setga

8 GA for transcriptomics

Having an algorithm, that minimizes a selected subset of a set of elements while optimizing a list of optimization functions, we simply input the gene set as the set of elements and the variance minimization function introduced in chapter three as the fitness function. Afterwards we selected the most fitting genetic operators and the best selection function and optimize the hyper-parameters to return a minimal set of genes in the best possible time. By the definition of the `SetGA` optimization, a 1 in the solution on the index i represents selecting the gene g_i to the set of genes being involved in development.
We name our tool, that is able to extract a small subset of genes, that are driving the TAI pattern `GATAI`.

8.1 Fitness evaluation

To compute the fitness, we first need to compute the variance. We can easily sum the expression over the genes for a particular stage while selecting just the genes, that are not present in the solution by taking the dot product of the expression matrix with the negation of the solution array. In this way we compute the product of sum over the age-weighted expression and the sum over the expression. This way, we efficiently retrieve the TAI value for every stage and compute the variance over the stages.

```python
# Weight expression by phylostrata for every gene
expression_data.age_weighted = exps.mul(expression_data["
   Phylostratum"], axis=0).to_numpy()

def get_distance(solution):
    # Swap 0 and 1 in solution
    solution = np.logical_not(solution).astype(int)
    # Sum expression columnwise only for genes not selected
    age_weighted = solution.dot(expression_data.age_weighted)
    total_exp = solution.dot(expression_data.expressions_n)
    avgs = np.divide(age_weighted, total_exp)
    return np.var(avgs)
```

Figure 8.1 Python code for calculating the distance based on the solution and expression data.

To compute the p-value of the solution, we first need to compute the sampled variances. For that we need to permute the gene ages, recompute that TAI pattern and its variance. After testing different number of permutations, we decided to set the permutation parameter to 100000, because the computation of this number of sampled variances does take under 3 minutes for all of the datasets we tried and the optimization converges without problems. From these sampled variances and the variance of the solution, the p-value is computed as described in the previous sections. To make the p-value easily integrateable with the second part of the fitness, we would like to define the reformulate the p-value to a metric in the range between 0 and 1, where 0 is the best. For this reason, instead of the p-value, we take $1 -$ p-value. To prevent minimizing the $1 -$ p-value to a necessary low value, we set the whole fitness to 0 if the p-value reaches the value 0.8, that is $1 -$ p-value is lower than 0.2.

The above described approach would work, if the starting variance would be close to the variances of permuted TAI patterns, from which is the p-value computed. Unfortunately, that is often not the case and in such cases, there would be nothing to guide search until we reach a value, that is below one of the sampled variances. Furthermore we would possibly need way more permutations to guide the search properly. For this reason, the second part of the fitness is simply the current variance divided by the variance of the original TAI. This metric is roughly between 0 and 1 as well. These two metrics are then added together.

$$\text{fitness}(\text{var}) = \begin{cases} \frac{\text{var}}{\text{tai\_var}} + 1 - \text{p-value}(\text{var}) & \text{if p-value}(\text{var}) < 0.8 \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

Where:

$$\text{var is the variance of the solution}$$
$$\text{tai\_var is the variance of the original TAI pattern}$$

8.2 Mating

For genetic operators we found that regular bit flip mutation and uniform crossover yield satisfactory results, however the optimization is rather slow. That is why we implemented weighted crossover and mutation, where the probability for an index to be selected for a mutation or crossover is proportional to it's square root of variance of the TAI pattern over the stages. First we thought this sped up the process heavily, because the highly variant genes are good candidates for genes that are influencing the TAI pattern heavily. Later we found that the significant speed up for mutation was mostly due to much faster implementation than the default `DEAP` implementation of bit-flip mutation, but for the crossover, we could see a clear difference in the number of generations needed.

At the end we decided for our implementation of the weighted uniform crossover for the crossover operator and for the mutation, we use our implementation of the standard bit-flip mutation for half of the islands and the weighted bit-flip mutation for the other half. Such mutation setting enables fast convergence to the correct highly variant key drivers of the pattern, while making the identification of the less variant genes driving the pattern faster.

For the weighted mutation or weighted crossover, we originally used the `numpy.random.choice` function. However, after running `cProfile` and vizualizing the bottle necks of the optimization, we found out that one of the major bottle necks is the `numpy.random.choice` function. We used the fact that the weights do not change throughout the whole optimization and based on the weights computed the cumulative distribution function of the normalized weights, by creating an array by for every index of the normalized weights array summing the values on all indices smaller than the current one. Notice, that an array created this way is sorted. We compute the CDF array only once and then reuse it every time the mutation function is called. Every call of the mutation (crossover) function, we generate mutation rate times lenght of the solution random numbers between 0 and 1. And for every random number we search for the last index of the CDF array, where the value is smaller than the random number. This search can be done in logarithmic time, thus if we have the CDF array precomputed, the mutation runs in logarithmic time, compared to the DEAP mutation implementation, that scales linearly. Moreover, we implemented the function in numpy, which does the computation internally in C, which brings further speed up. We integrate the weighted mutation and crossover and the optimized version of the bit-flip mutation and uniform crossover also to the general set minimization library.

```python
# Weight expression by phylostrata for every gene
expression_data.age_weighted = exps.mul(expression_data["
   Phylostratum"], axis=0).to_numpy()

    def weighted_random_choice(weights):
        # Compute the cumulative distribution function (CDF)
        cum_weights = np.cumsum(weights/np.sum(weights))

        def random_numbers(size):
            # Generate random numbers uniformly in [0, 1)
            random_values = np.random.rand(size)
            # Find the indices where random values fall in the
                CDF
            indices = np.searchsorted(cum_weights,
               random_values)
            return indices
        return random_numbers
```

Figure 8.2 Python code for calculating the distance based on the solution and expression data.

For this reason we reimplemented the classic bit-flip mutation simply by running this weighted bit-flip mutation with uniform weights. Analysis of the runtime of the different implementations is provided in the results section. After trying different setups, we found out the setting crossover rate to 0.02 and mutation rate to 0.005 yields the best results.

For selection, we tried all the multi-objective selection methods that the `DEAP` library offers. `NSGA2` showed to be quite fast, but failed to converge for a lot of datasets. `SPEA2` did mostly converge without problems, however was significantly slower. NSGA3 with uniformly distributed reference points had similar issues as NSGA2. It mostly first pulled the p-value criterion all the way to 0 (that is to p-value higher than 0.2), but then while further optimizing the number of extracted genes, the p-value started to grow again and GATAI often failed to converge. This problem was first solved by distributing most of the reference points so that the search often prioritizes low p-value over a low number of extracted genes. Later, we tried to reproduce the divergent behaviour of NSGA3 with the uniformly distributed reference points, but find out that due to speeding up the whole computation and as a result enlarging the population size, GATAI behaves well also under those circumstances. Moreover, having too many reference points with high values for the p-value objective, GATAI takes longer to reduce the number of identified genes, possibly because less solutions with less identified genes, that have a lower p-value are kept in the population. These could be later crossed over with solutions with a higher p-value to create solutions with less genes identified and a high p-value. To prevent the possible divergent behaviour of the algorithm on some datasets that GATAI has not been tested on, while not slowing down the computation significantly, we decided to generate only few extra reference points prioritizing the p-value objective. In total, we generated 10 uniformly distributed reference points and another 3 with prioritizing the p-value objective.

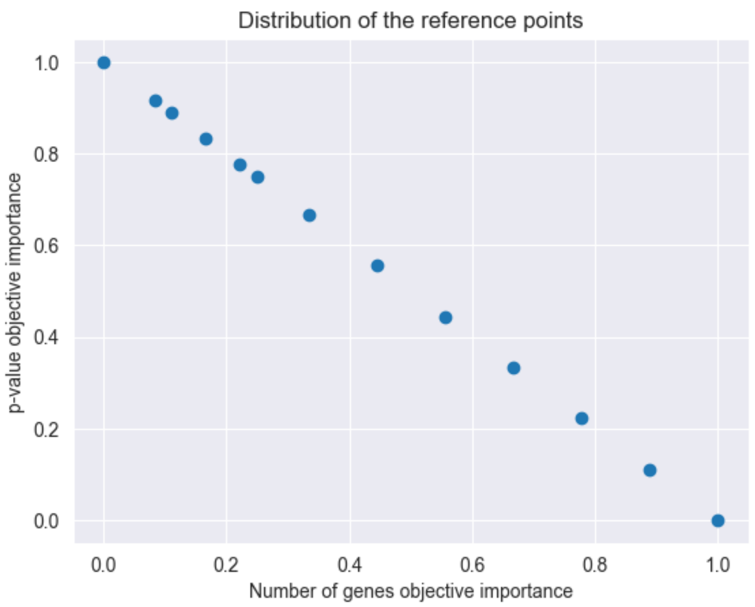


Figure 8.3 Reference points used by the NSGA3 selection algorithm. More points are centered around the high importance value for the p-value.

8.3 Integration with SetGA

The integration with SetGA is seamless, because SetGA was originally part of GATAI and only later developed in a stand alone library. To run the SetGA minimizer, we call the SetGA function `setga.select_subset.run_minimizer()` with number of elements equal to the number of genes, the fitness function presented above, ”Variance” as the name of the optimization objective, variance as the optimization objective to select the best solution in every generation for the log book, mutation and crossover rates (probability of an index to be mutated (crossed over)), population size of 150 and 4 islands. These parameters were experimentally determined to make every generation be computed fast, while keeping the number of generations reasonably low. Mutation and crossover and selection function were selected as described above, with the custom reference points for the `NSGA3` selection. We set the number of generations without any change in the number of number of selected genes to 200.

8.4 Final solution

Final solution is extracted by first combining the populations from all islands and then pre-selecting those solutions, with p-value lower than 0.4, as the TAI patterns of the datasets with the genes identified by those solutions are at least based on variance unrecognizable from the TAI patterns of the datasets with randomly permuted gene ages. We set the threshold intentionally fairly low, because we compute the p-value from the sampled variances (as described in chapter 4) of all the genes in the original dataset. As explained, this approach is slightly inaccurate. We suspect based on empirical trials that for the dataset without the identified genes, the sampled variances will be slightly lower, thus the p-value slightly higher. Finally only those genes are kept in the final solution, which occur in more than 0.9 of the solutions. All these parameters are tunable.

8.5 Postprocessing

Genes which we identify with the help of our multi-objective optimization algorithm are mainly highly expressed genes which can be placed downstream of the true key regulators within a hierarchical gene regulatory network. However, because they are up or down-regulated by those upstream regulators, they might show similar expression patterns, only with different magnitudes. That is why as followup analysis, we identify correlated genes by computing the Pearson index for each of the identified gene expression profiles with all of the other gene expression profiles in the dataset. As a result, we return for each the associated ones with correlation coefficient higher than 0.95. We allow running this correlation procedure only for datasets with more than 30 developmental stages, because for a lower number of stages the probability of genes being correlated simply by small pattern space grows rapidly [25].

For a lot of downstream analyses, such as homology detection or common domain (functional unit of a protein) identification, we need to know the sequences of the identified genes. Those we usually have stored in one fasta file (standard format to store biological sequences) for the whole dataset. This function creates a fasta file with sequences only of the identified gene set.

8.6 Usage

GATAI can be installed by pipy by calling `pip3 install gatai`. The user is provided with a commandline interface. By running `gatai -h` all options are displayed. The most important command is `select_genes`, which will take the input dataset and name of the output folder and run the main algorithm that finds the set of genes minimizing the variance of the TAI pattern.

Having the resulting gene set at hand, one can run `gatai find_coexpressed` which searches for genes with similar expression patterns to some of the identified ones. Assuming a fasta file with the sequences of all genes is present, the user can also get a fasta file with only the sequences of the selected genes by running `gatai get_fastas`.

A help for all functions is provided by adding the `-h` or `--help` flag. The complete documentation can be found in the github repository[1].

8.7 Single cell

We decided to extend GATAI to be able to work with single cell datasets. So far, we have been focusing on the bulk transcriptomics case as we considered such datasets to be easier to work with, because of the vastly lower number of stages. The bulk datasets tend to have between 5 and 50 number of developmental stages, while the single cell ones can reach up to tens of thousands of cell differentiation or developmental stages. This adds not only computational complexity, but with so many stages it might be difficult to disentangle just the few key regulators of the whole developmental process.

8.7.1 Sparse encoding

Fortunately, for single cell expression data, the genes tend to have zero signal for most of the stages. For that reason, we tried to utilize the sparse matrix encoding implemented in `scipy` [39]. We implemented a sparse matrix optimized version of the computation of the sampled variances. By using just operations that were able to handle sparse matrices, we were able to speed up the variance computation significantly. However, we were not able to match the speed of the computation of the bulk case. Due to the slower variance computation, we were forced to decrease the number of permutations. Luckily, our experiments (presented in the results section) showed that this does not seem to harm the optimization.

We also needed to reimplement the fitness computation. In difference to the bulk dataset, every operation that iterates over all the developmental stages is expensive for the single cell case. That is why we precomputed a column wise sum of the expression matrix and of the age-weighted expression matrix, to obtain the age-weighted total expression for every stage and the total expression for every

[1]https://github.com/lavakin/gatai

stage. In the TAI computation, these two matrices are divided to get the TAI for every stage. To compute the total expression of the current solution, we simply go over the selected genes (those with 1 in the solution), for every stage sum their expressions and subtract those stage-wise from the precomputed total expression. Analogically for the age-weighted expression.

```python
def compute_distance(expression_data, solution):

    # Stage-wise sum of expression of selected genes
    a_w_result = solution @ expression_data.age_weighted
    a_result = solution @ expression_data.expressions_n

    # "Removing" the selected genes from the dataset
    numerator = expression_data.weighted_sum - a_w_result
    denominator = expression_data.exp_sum - a_result

    division_result = np.divide(numerator, denominator)

    distance = np.var(division_result)

    return distance
```

Figure 8.4 Python function to compute distance based on expression data and solution.

We tried to implement this approach also for the bulk datasets, but it proved to be slower than the previous matrix-operation approach possibly effectively optimized by `numpy`.

After these optimizations, one generation for single cell case is still significantly slower than for the bulk one, but luckily the single cell datasets seem easier to optimize, thus don't require that many generations to converge, which makes the optimization fairly quick. All analysis can be found in the results section.

9 Results

Having an algorithm that can identify a set of genes that drive the TAI pattern, we need to test, if the algorithm performs well on various datasets. That is, if it indeed optimizes the given objectives (p-value and number of genes), finishes in a reasonable time and outputs a gene set that is stable over multiple runs of the algorithm.

9.1 Datasets

We were fortunate to be able collect many published datasets across the tree of life. The datasets come from different studies and the expression was measured by different experimental setups. We collected even more datasets than listed below, unfortunately some of those didn't show significant TAI pattern even before the optimization. We take 0.1 as the minimal significance level. When running GATAI on datasets that do not show a significant TAI pattern, usually just a couple of genes are identified, which can destroy the whole pattern. For this reason we think that such patterns are caused by a measurement error, thus the identified genes might not hold much biological relevance, hence we do not consider those datasets for testing the tools functionality.

Kingdom	**Species**	**Technology**	**Normalization**	**Study**
Plants	Arabidopsis thaliana	Microarray		[40]
Plants	Brassica rapa	RNA-seq	TPM	[41]
Plants	Brassica nigra	RNA-seq	TPM	[41]
Plants	Brassica juncea	RNA-seq	TPM	[41]
Plants	Brassica napus	RNA-seq	TPM	[41]
Bacteria	Bacillus subtilis	RNA-seq	RAW	
Fungi	Coprinopsis cinerea	Microarray		[42]
Animals	Mnemiopsis leidyi	RNA-seq	RAW	[2]
Animals	Drosophila melanogaster	RNA-seq	RAW	[2]
Animals	Caenorhabditis elegans	RNA-seq	RAW	[2]
Animals	Platynereis dumerilii	RNA-seq	RAW	[2]
Animals	Schmidtea polychroa	RNA-seq	RAW	[2]

Table 9.1 Summary of datasets, methods and sources

For the datasets from the levin study [2], no normalization is required, because of the specific experimental setup.

For each dataset, GenEra was run on the protein sequences to retrieve the gene ages.

9.2 GATAI statistics

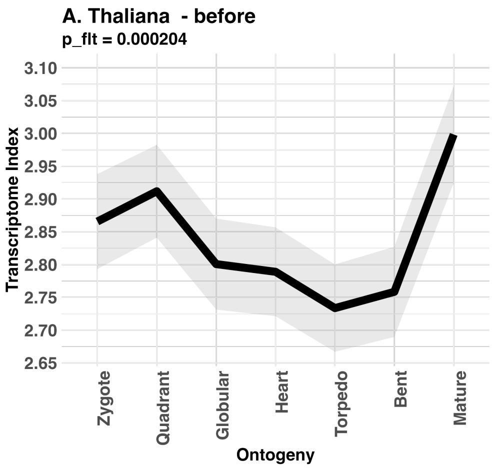
The main requirements on the algorithm were to optimize well the two optimization criteria, the p-value and the set size and to identify a stable set of genes. In this section, we will test how well GATAI can fulfill all these requirements.

9.2.1 p-value

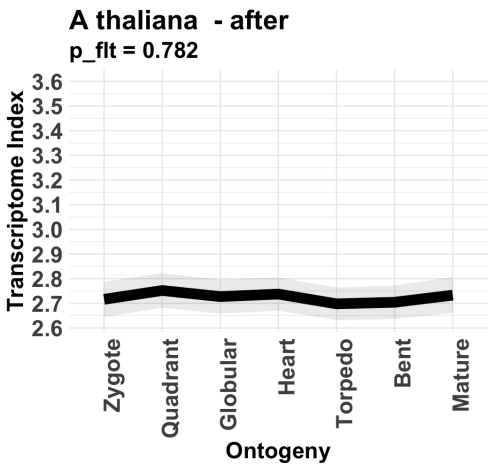
First, we look at the p-value before and after and the time needed for the optimization. The p-value as computed in the fitness function outputs 1 for any TAI pattern whose variance is higher than all sampled variance, thus it does not give us any information on how much higher the variance is than the sampled ones. For this reason, we use the `myTAI` [11] function `FlatLineTest` with 100000 permutations. The `FlatLineTest` function first estimates a gamma distribution from the sampled variances and computes the p-value of the TAI pattern variance belonging to this distribution afterwards.

Species	p-value before	p-value after	Optim. time	Num. genes	Min. genes in sol.	Num. generations
A. thaliana	0.0002	0.782	73.01 s	45	58	672
B. rapa	4.38e-6	0.429	223.85 s	62	105	2771
B. nigra	0.062	0.118	114.24 s	28	49	1431
B. juncea	0.031	0.227	151.41 s	29	49	1591
B. napus	8.85e-8	0.43	22 min	649	1473	12000
B. subtilis	3.5e-11	0.471	125.21 s	32	47	1001
C. cinerea	1.80e-20	0.51	324.58 s	458	473	5351
M. leidyi	0.00026	0.191	105.95 s	32	45	1101
D. melanogaster	4.16e-12	0.328	10 min	54	93	1222
C. elegans	0.00014	0.569	440.60 s	24	30	1041
P. dumerilii	1.8e-7	0.473	13 min	50	68	1421
S. polychroa	0.00183	0.616	30.74 s	8	9	421

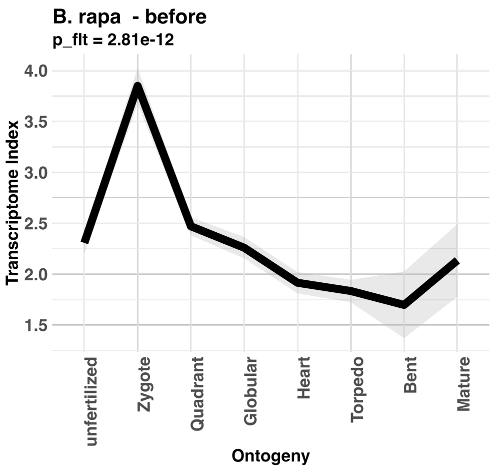
Table 9.2 GATAI statistics for all available datasets capturing the p-value of the original TAI pattern, the p-value after removing the identified genes, time spend on the optimization (excluding the time needed to compute the permutations) and on generating the variances by permuting the gene ages. Number of genes in the final identified gene set, minimum number of genes identified between all good solutions from population and number of generations.



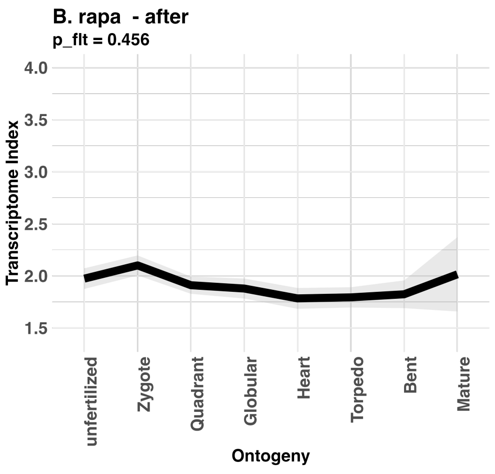
(a)



(b)

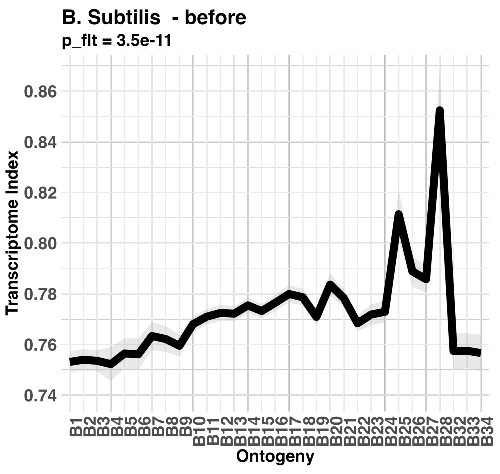


(c)

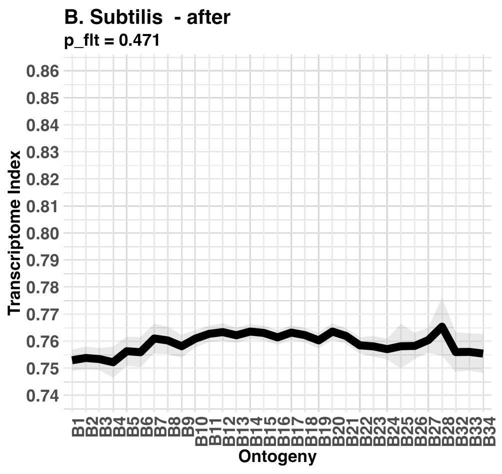


(d)

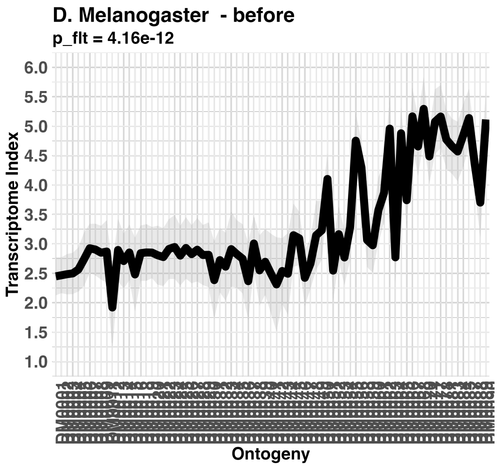
Figure 9.1



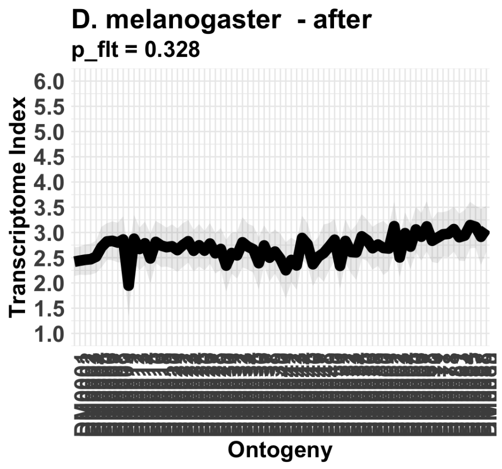
(a)



(b)



(c)



(d)

Figure 9.2 TAI pattern before and after removing the identified genes alongside their respective p-values for representative datasets. Plots generated by `myTAI` function `PlotSignatures`

From the Table 9.2 and Figures 9.1, 9.2 is apparent, that `GATAI` was indeed able to make the TAI pattern of all the datasets insignificant after removing the identified genes. Again, we are taking 0.1 as the significance threshold. We can see that for some datasets, the significance is not very high, even-though in `GATAI` we set the optimization threshold for the p-value (threshold after which we manually set the p-value 1) to 0.8. We think, that the relatively low significance can be caused by two things.
First case is, if the original TAI pattern is significant, but its p-value is close to 0.1, like in the case of Brassica Nigra and Brassica Juncea. In those cases the pattern is heavily influenced by just a couple of genes and removing those influences not only the pattern heavily, but also the sampled variances. From the Figure 9.3 we see that the variance distribution is shifted right, which causes the same variance to have higher p-value under this distribution compared to the original distribution.

Second case is, if a pattern can be destroyed by many different combinations of genes. Since we at the end of the optimization take just the genes, that appear

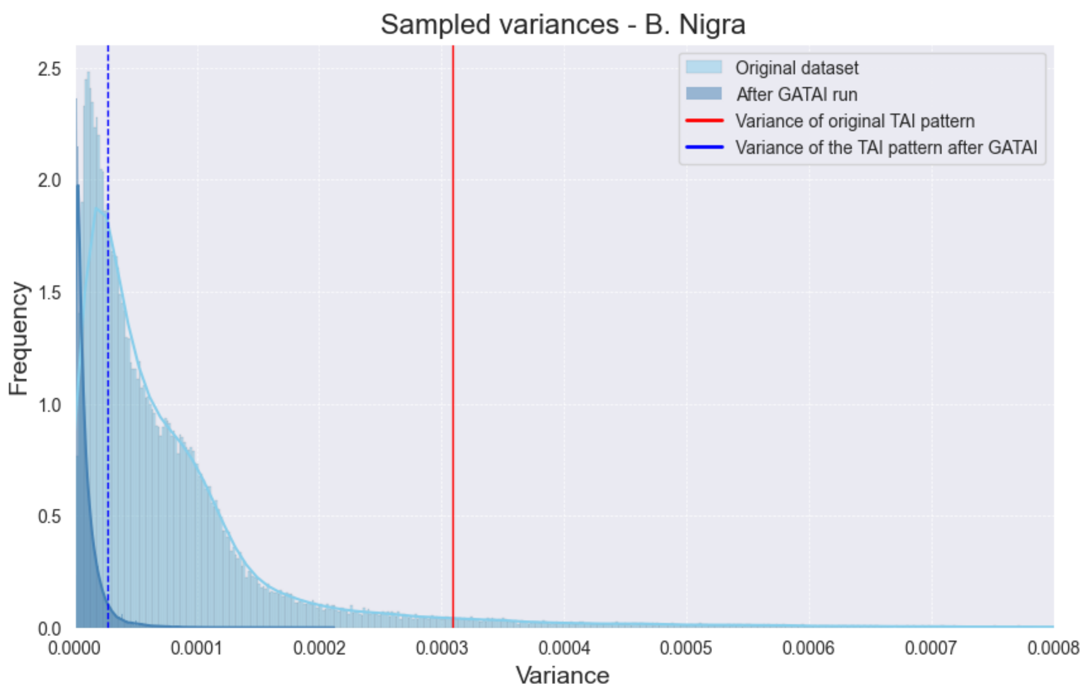
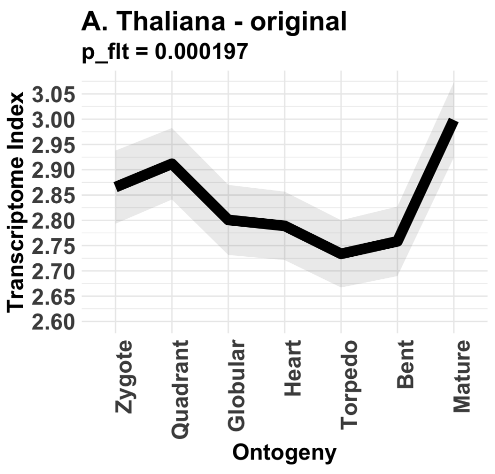


Figure 9.3 Distribution of variances of the TAI pattern after permuting the gene ages for the original dataset (light blue) and for the dataset after removing the identified genes (dark blue). The variance of the TAI pattern after removing the identified genes (blue dashed line) is very insignificant for the light blue distribution, but less insignificant for the dark blue, thus having a lower p-value after recomputing the sampled variances.

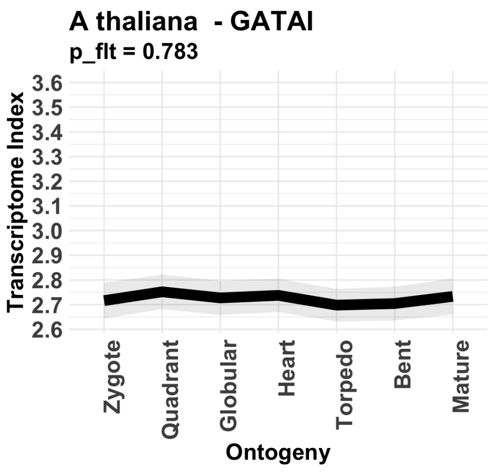
in at least three quarters of the solutions, in such case many of the genes that were taking part on destroying the pattern are not included in the final solution. This factor might also contribute to the low p-value of the TAI pattern of the aforementioned Brassicas.

9.2.2 GATAI importance

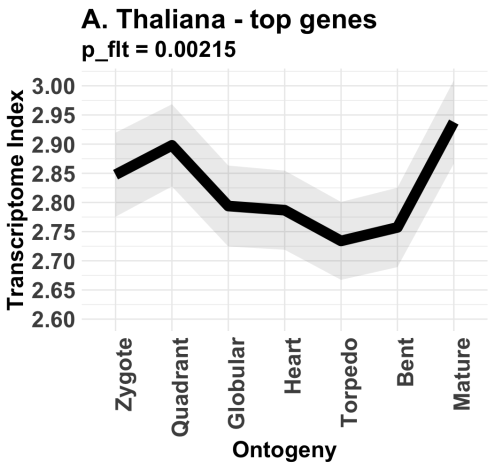
An important question is, do we really need a sophisticated optimization? Wouldn't we achieve the same result just by removing the most variant genes from the dataset? Doesn't GATAI at the end capture exactly those? This would mean, that the extracted genes do probably not hold much biological relevance and using a GA to tackle the problem is not necessary.



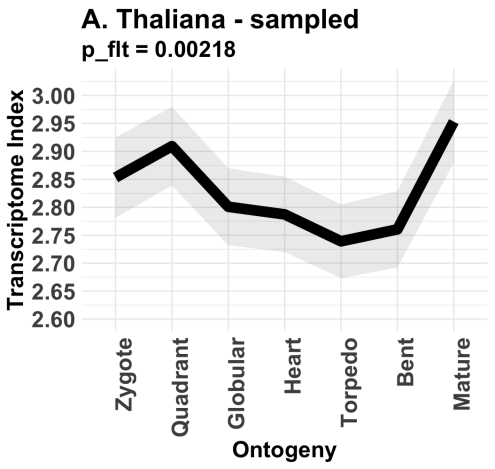
(a)



(b)

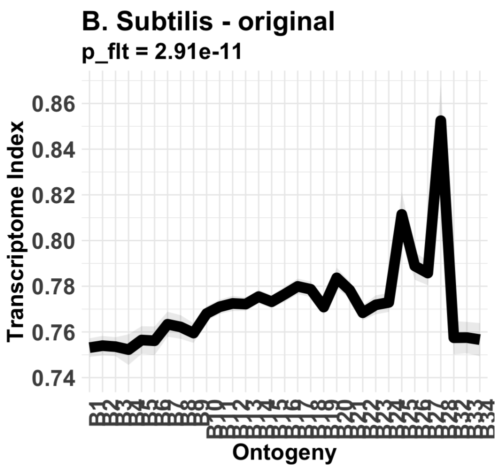


(c)

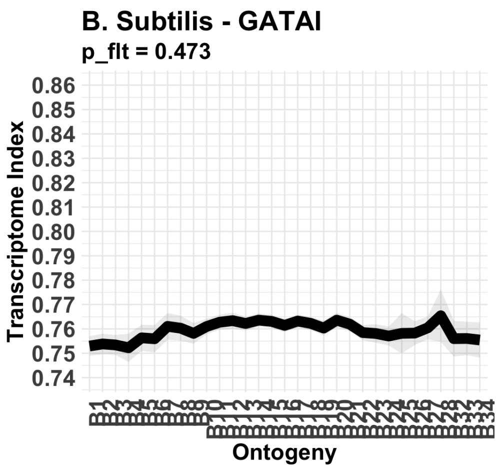


(d)

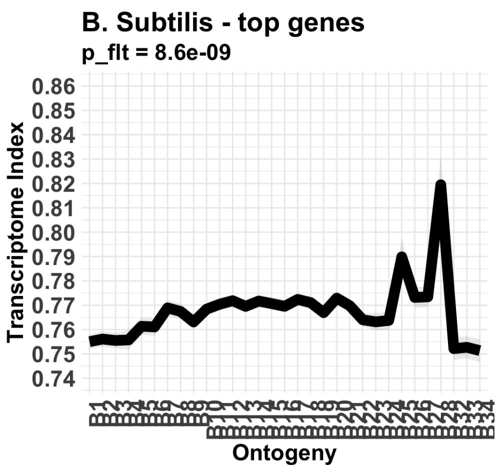
Figure 9.4 TAI pattern of the Arabidopsis Thaliana (a) original dataset, (b) the dataset after removing the genes identified by `GATAI`, (c) the dataset after removing n of most variant genes, where n is the number of genes identified by `GATAI`, (d) the dataset after removing n random genes, where the probability of a gene to be selected is proportional to its variance.



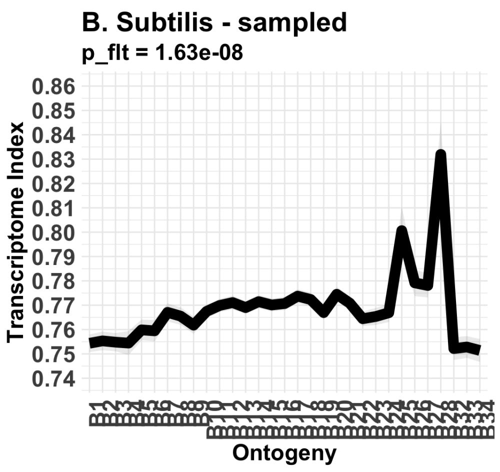
(a)



(b)



(c)



(d)

Figure 9.5 TAI pattern of the Bacillus Subtilis(a) original dataset ,(b) the dataset after removing the genes identified by `GATAI`, (c) the dataset after removing n of most variant genes, where n is the number of genes identified by `GATAI`, (d) the dataset after removing n random genes, where the probability of a gene to be selected is proportional to its variance.

From the Figures 9.4, 9.5 its apparent that removing the most expressed genes, or random genes, where the probability of a gene being selected is proportional to its variance does reduce the variance slightly, but the pattern is still far from being insignificant.

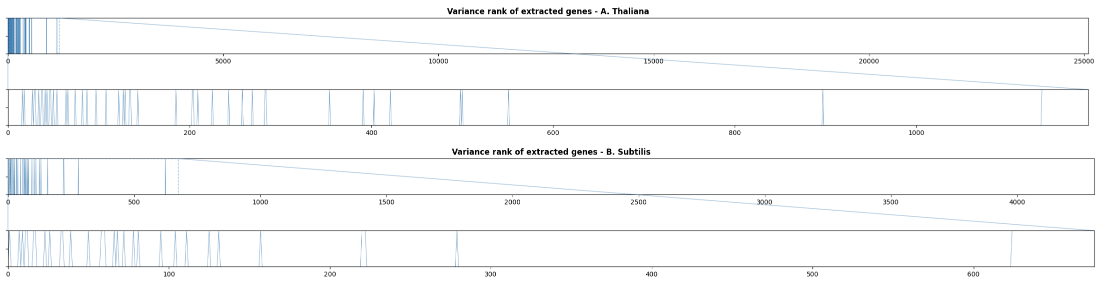


Figure 9.6 Genes of A. Thaliana and B. Subtilis sorted by variance. Genes identified by `GATAI` marked in blue. Zoom-in on the range of the plot, where the identified genes are present.

From the Figure 9.6 we see that the extracted genes are indeed the highly variant one, but the subset of the genes destroying the TAI pattern needs to be carefully selected from those.

9.2.3 Number of permutations needed

To estimate the p-value needed for the fitness computation, we need to first compute the sampled variances, that is permute the gene ages and recompute the variance of the TAI pattern. We need to look at how many sampled variances do we need. In other words how many permutations of the gene ages, from which the sampled TAI values are computed, are necessary. There might be two problems with computing too little variances and thus having not many variances to estimate the p-value from. First is that the selection of the solution for the next generation of the genetic algorithm is partially based on the p-value, thus the algorithm has less guidance on which solutions to prefer. However, second part of the fitness is purely the variance of the solution, which might guide the search if two solutions have variance between two sampled variances. The other problem might be not approximating the true variance distribution well. This might result in stopping the optimization while the true p-value is still not yet satisfyingly low.

Permutations	**p-value**	**Num. genes**	**Optim. time**
10	0.596	42	61.49
100	0.763	47	79.68
1000	0.741	44	77.94
10000	0.793	43	76.44
100000	0.837	43	77.65

Table 9.3 p-value, number of identified genes and the optimization time for different number of permutation (sampled variances).

From the Table 9.3 is apparent that the optimization runs well even for a low number of permutations, but the final p-value gets worse because of the insufficient estimation of the true distribution of the sampled variances. This hints toward a further speed up opportunity of the algorithm by not computing the p-value of the TAI pattern for every solution during the optimization and guiding the search only by the variance and just computing the variance, which has a p-value of 0.8 and setting such variance as the optimization threshold, where any lower variance is automatically set to 0. In such case, we would not need to compute the number of sampled variances that are higher than the variance of the solution for every solution in a population in every generation. While this is not a costly operation, it would still speed up the computation slightly.

9.2.4 Stability of solutions

To test if the identified set of genes vary significantly between the `GATAI` runs, we ran the tool multiple times and examined the resulting gene sets. We performed this test for the microarray dataset of Arabidopsis Thaliana and the RNA-Seq dataset of Bacillus Subtilis, to span different methods and kingdoms.
If the set of identified genes would not be stable across the runs, we would need to make the criteria for the optimization stricter, for example rise the p-value threshold or rise the threshold for in how many percent of solutions a gene needs to appear to be included in the final gene set.

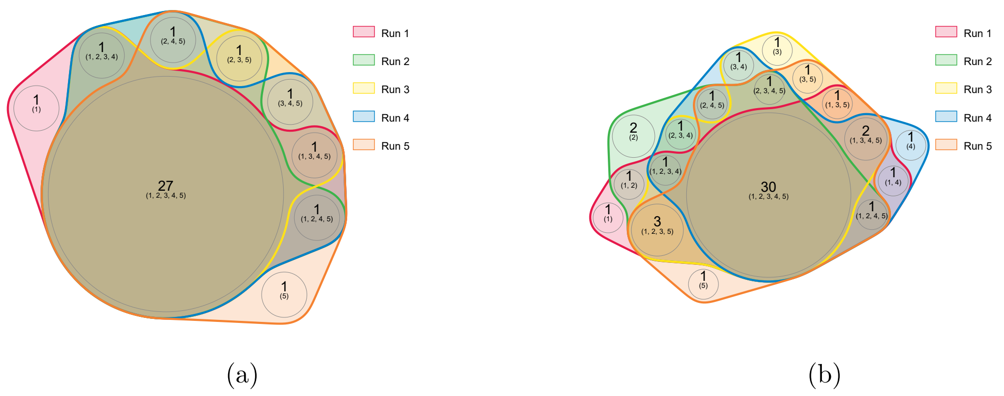


Figure 9.7 Euler diagrams of the overlapp of the gene sets identified by different runs of `GATAI` with identical parameters, for A. Thaliana (right) and B. Subtilis (left). The diameter of the circles is proportional to the size if the overlap.

As we see from the Figure 9.7 for both datasets, most of the identified genes are stable across the runs of the algorithm.

9.2.5 Influence of different genetic operators

Next we test, if our custom mutation and crossover operators tailored for this specific problem, where the probability for a given index to be mutated (crossed over) is proportional to the square root of variance of the gene on that index, performs better than other available operators. For A. Thaliana and D. Melanogaster, we test the differences in runtime and number of generations between the different operator combinations.

Dataset	Mutation	Crossover	Num. generations	Optim. time
A. Thaliana	**Half weighted**	**Weighted**	**672**	**73.01 s**
A. Thaliana	Half weighted	Uniform opt.	1072	113.63 s
A. Thaliana	Bit-flip opt.	Weighted	782	83.70 s
A. Thaliana	Bit-flip opt	Uniform opt.	1011	105.38 s
A. Thaliana	Bit-flip	Uniform	1251	612.08 s
D. Melanogaster	**Half weighted**	**Weighted**	**1222**	**10 min**
D. Melanogaster	Half weighted	Uniform opt.	1487	13 min
D. Melanogaster	Bit-flip opt.	Weighted	1392	12 min
D. Melanogaster	Bit-flip opt.	Uniform opt.	1782	15 min
D. Melanogaster	Bit-flip	Uniform	1662	27 min

Table 9.4 Table of the number of generations and the optimization time (excluding the time needed to compute the permutations) for A. Thaliana and D. Melanogaster for different combinations of mating operators, where opt. means the optimized version of the original `DEAP` operator. Half weighted mutation means half of the islands are mutated by the weighted mutation and half by the optimized version of bit-flip mutation. The operators combination that `GATAI` uses are marked in bold.

Lower number of generations generally signify better mating operators, because the algorithm is able to converge faster because of them. Higher optimization time by similar number of generations signifies higher computational complexity of computing the operators (by otherwise identical setup).

From the Table 9.4 we see that the genetic operators `GATAI` is using have the lowest run time and number of generations needed to converge, although for A. Thaliana we can speculate that there is not a significant difference to using the optimized version of bit-flip mutation. Mainly, we see a very significant difference between using the optimized versions of the operators we developed and the ones `DEAP` provides.

9.3 Biological results

Apart from showing that `GATAI` behaves well on our data and optimizes well both of the optimization objectives, we need to show, that the identified genes are biologically meaningful. This part is just a small portion of the analysis that needs to be done, but in this work we want to focus more on the algorithmic side of the problem. We put identified gene sets for a selection of datasets in the attachments A.1.

9.3.1 Function of the identified genes

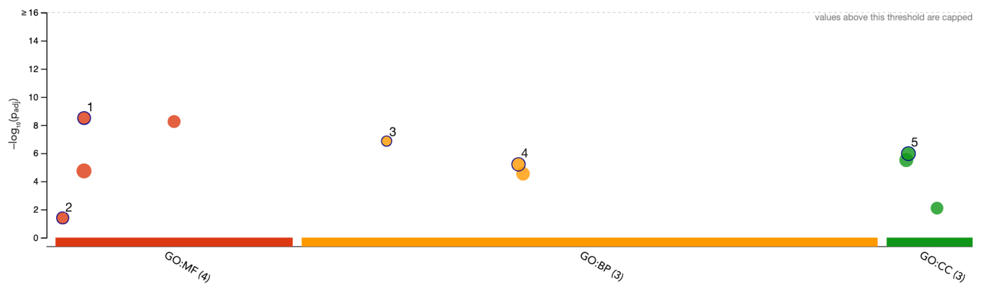
After extracting the gene sets from the datasets, we can examine the identified genes. The easiest way to find out, which functions are enriched is to look at the Gene Onthology (GO) annotations. For this purpose, we used the panther database [43] and gprofiler [44]. Unfortunately, for most of these species, no reference annotations are provided. For the datasets, for which the tools provided the reference, we quickly found that for most of them a big part of the identified genes lack any annotation (unknown function). We tried to use the eggnogmapper [45], which searches for known function also for orthologous genes (genes which originated from the same gene). Here, we got at least some annotation for a big portion of the identified genes for the plant datasets, but for the other datasets, not many annotations have been found. Even though substantial amount of annotations for the plant datasets are development related, we cannot claim it to be significant, because we lack any information on the background distribution of the biological functions.

The lack of annotations might be caused by the lack of interest in those genes by the scientific community, but it may also signify the lethality of mutations in those genes. The latter case would make them not appear in a candidate gene set after mutagenesis, hence their function would not be uncovered. The latter case would also support our claim that the identified genes are development related.

We report interesting results in Figure 9.8 for the Drosophila Melanogaster dataset. This may be due to Drosophila being one of the ten most annotated species [46] and having a very significant TAI pattern. For the Danio Rerio dataset, the set of identified genes is enriched in development related GO categories as well, however the TAI pattern on the dataset is not significant, hence we do not report the extracted genes.

Form the Figure 9.8 we see that all the GO categories that the extracted genes were enriched in are related to development with a very significant p-value, except for one, which is also the one with the lowest p-value. This supports our claim that we are indeed capturing genes involved in development.

We could definitely perform a more detailed analysis, for example look into the gene families or the single protein domains. However, this work is focused more on the algorithm and proper analysis of the identified gene sets will be done in future works.



ID	Source	Term ID	Term Name	p_{adj} (query_1)
1	GO:MF	GO:0005214	structural constituent of chitin-based cuticle	3.176×10^{-9}
2	GO:MF	GO:0003713	transcription coactivator activity	3.979×10^{-2}
3	GO:BP	GO:0010171	body morphogenesis	1.391×10^{-7}
4	GO:BP	GO:0040003	chitin-based cuticle development	6.192×10^{-6}
5	GO:CC	GO:0031012	extracellular matrix	1.076×10^{-6}

version	e111_eg58_p18_f463989d
date	13/07/2024, 17:12:20
organism	dmelanogaster

g:Profiler

Figure 9.8 GO categories enriched in the identified gene set for Drosophila Melanogaster. Most of the categories (except for the second one) are development related.

9.3.2 Homology

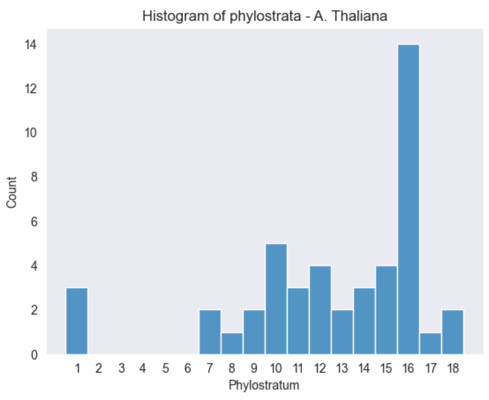
One could ask if the identified genes of different species are related in some way. To answer this question, we used the pairwise sequence alignment tool `diamond2` [47], which when provided with database and a set of queries, can for each query identify a list of similar sequences and output their identity score. We ran `diamond2` for every pair of the identified gene sets of the datasets from the Levin study [2] and every pair of the identified gene sets of the Brassicas [41]. Each time using the sequences of one gene set as database and the sequences of the other gene set as query and other way around.

We set the identity threshold to 50% and searched for the homologous genes. We found no homologs between the gene sets of the Levin datasets, however, we found clusters of homologs for the Brassica datasets, some spanning all the Brassica gene sets. To confirm our results, we ran `OrthoFinder` [48] (which also runs diamond2 internally), on the sequences of the identified genes of Brassicas, to find the clusters of genes that share evolutionary origin and confirmed our results. We then ran eggnogmapper on the sequences of the cluster to find the function of the genes in each cluster.

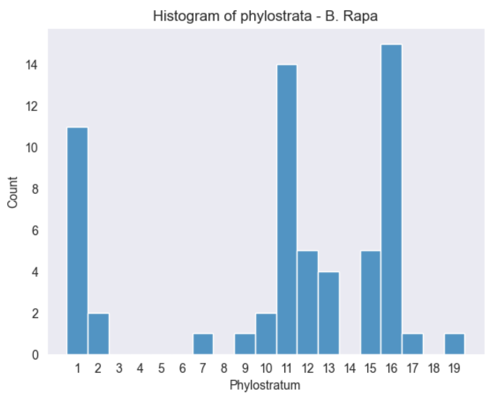
For the cluster spanning the largest number of genes, that is 27, all of the genes were annotated to code a 2S seed storage protein. The fact that we were able to identify orthologous genes across closely related species supports the claim that we are indeed not capturing just random highly expressed and higly variant genes. The biggest orthologous cluster extracted being annotated with a development related function brings another strong proof that `GATAI` is truely capturing what it is supposed to. Unfortunately, that was the only annotated cluster, but it would be very interesting to test if the genes in the other big cluster have a development related function.

9.3.3 Gene age and expression

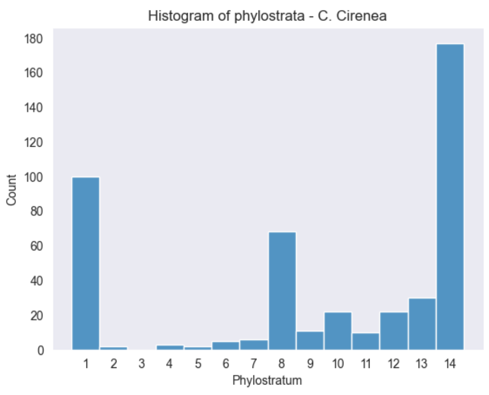
We look into what are the gene ages of the genes that are captured by GATAI. Given the fact that we are capturing the highly expressed genes, we would think that we capture more of the evolutionary younger genes, because these tend to be lower in the gene regulation network hierarchy and such genes tend to be more expressed, than older genes which are possibly transcription factors, which tend to be lowly expressed [49].



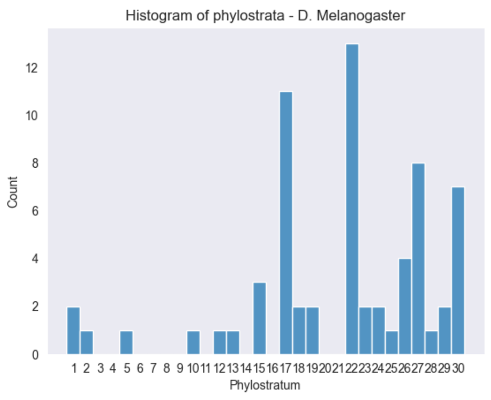
(a)



(b)



(c)



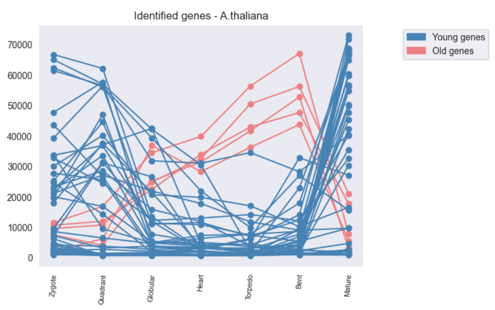
(d)

Figure 9.9 Distributions of gene ages (phylostrata) of the identified gene sets for dataset across the kingdoms.

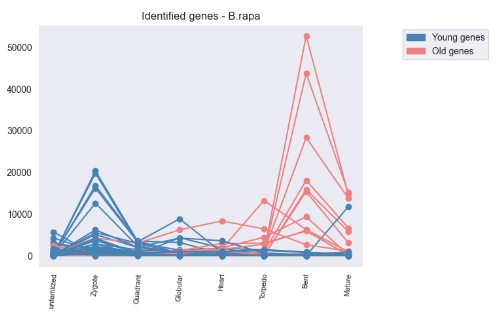
Indeed, in the Figure 9.9 we see, that most of the genes are evolutionary younger. This might also be the reason why we did not find and homologs for species that are not closely related.

Another interesting property to look at is how the TAI pattern comes together. We expect that the evolutionary older identified genes have peaks in their expression pattern in the minima of the TAI pattern and the evolutionary younger ones have peaks at the maxima of the TAI. This would rise the TAI value for the stages where the identified older genes are highly expressed and lower the

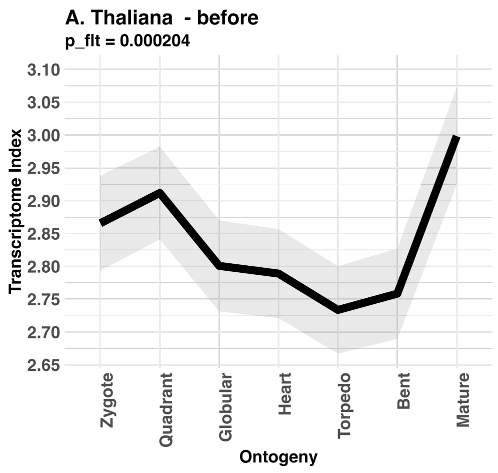
TAI value where the identified younger genes are highly expressed. To examine this behaviour, we look at the expression patterns for Arabidopsis Thaliana and Brassica Rapa. These are one of the few species that have small enough number of stages and small enough number of extracted genes for a nice visualization.



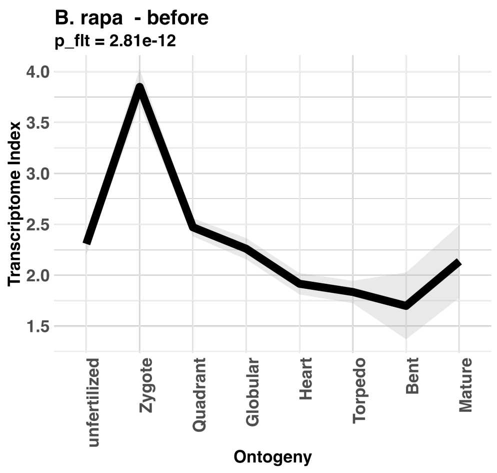
(a)



(b)



(c)



(d)

Figure 9.10 **(a,b):**Expression patterns for the gene sets identified by `GATAI` for A. Thaliana (a) and B. Rapa (b). Genes with phylostratum at most 3 colored pink, genes with higher phylostratum colored blue. **(c,d):** TAI pattern for A. Thaliana (c) and B. Rapa (d).

As we see from the Figure 9.10, the expression patterns are as expected for both gene sets. The older genes are more expressed in the minima of the TAI pattern and the younger genes are more expressed in the maxima.

9.3.4 Dataset transformation

So far we focused on untransformed datasets. However, users might want to run `GATAI` after transforming the dataset by one of the common transformations used for expression data. Hence, we look into how square root and logarithm (being the most common transformations) influence the runtime of `GATAI` and the identified gene sets.

From the Figure 9.5 we see that significantly more genes are extracted from the transformed datasets. This is caused by reducing the impact of the highly variant genes on the overall pattern. This results in overall more candidate genes,

Dataset	Num. genes	Num. generations	Optim. time
A. Thaliana	45	672	73.01 s
A. Thaliana - sqrt	180	1501	154.23 s
A. Thaliana log	591	6201	634.37 s
D. Melanogaster	56	1222	10 min
D.Melanogaster - sqrt	456	7131	38 min
D.Melanogaster - log	896	7971	33 min

Table 9.5 Table of the number of genes, number of generations and the optimization time (excluding the time needed to compute the permutations) for A. Thaliana and D. Melanogaster under different transformations.

which makes the optimization more complicated thus the optimizer needs more time to converge.

From the Figure 9.11 we see that with the transformations, we capture many more evolutionary older genes. This might be because evolutionary older genes include transcription factors which tend to be less expressed than the young genes [7] and with the transformations, the differences become less significant.

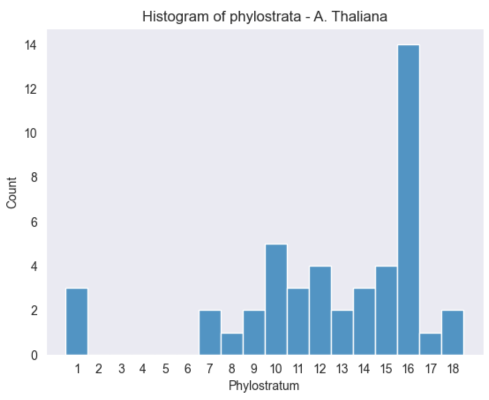
9.4 Single cell datasets

The interpretability of the TAI metric has not been tested for single-cell data. Once more studies of the single-cell case will be made, better optimization function might be invented. Thus, our aim now is to show that our tool can handle such a huge datasets, but the identified gene set will not and in our opinion shall not be interpreted. In this section we thus test the tool just in terms of the time complexity, not the biological interpretation.

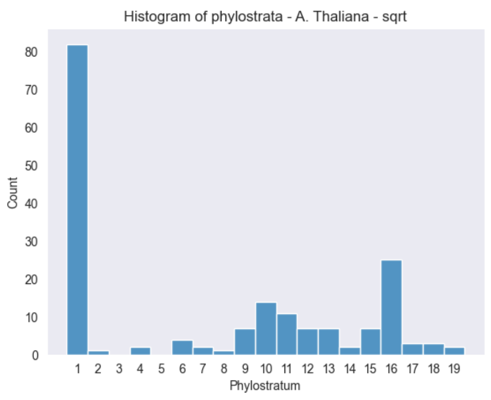
We took four single cell datasets for Arabidopsis Thaliana from [50], with very different number of stages. For all the datasets the expression of the same 22487 has been measured.

During some preliminary analyses we found that most of the single cell datasets are easy to optimize, but each generation takes significantly more time than in the bulk case. That is why we run the optimization with 3 islands only and 70 individuals in every island.

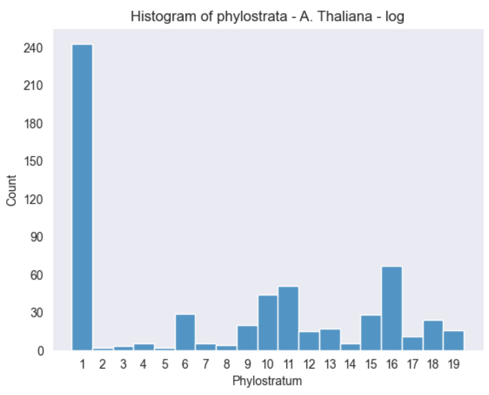
First we look into how the number of stages influences the runtime of the algorithm. We hope, that the algorithm will scale well due to the utilization of sparse matrices.



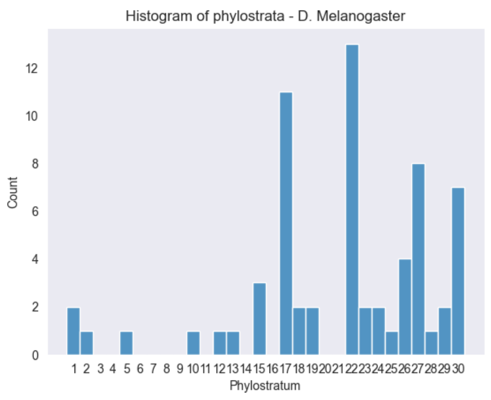
(a)



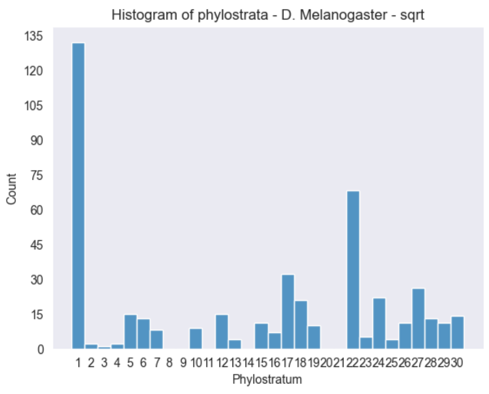
(b)



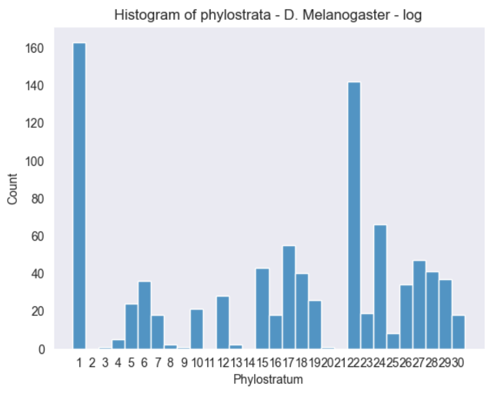
(c)



(d)



(e)



(f)

Figure 9.11 Histogram of the phylostrata distribution for A. Thaliana and D. Melanogaster with different transformations (logarithm and sqare root).

The total runtime of the algorithm is influenced by the significance of the original TAI pattern and by the contribution of each gene. To mitigate this bias, we compute the average time needed to preform one generation of the algorithm.

Acknowledging not having enough data to do proper analysis, we still measure the total time needed for the minimization for all the single cell datasets and the number of generations required. This give us some idea, if for datasets with more stages it is harder to find a small set of genes creating the TAI pattern.

Dataset	Stages	Optim. Time	Perm. Time	Generations	Time. per gen.	Ident. genes
5	19603	57.45 s	161.13 s	59	0.97 s	2
17	9676	546.55 s	40.84 s	361	1.51 s	29
21	4459	125.26 s	11.98 s	261	0.48 s	16
31	339	104.43 s	2.54 s	831	0.13 s	101

Table 9.6 Comparison of the time of the optimization (excluding the time needed to compute the permutations), permutation time, number of generations, time per generation and the identified number of genes between the single-cell datasets.

Based on the Table 9.6 we can clearly see that the total runtime of the optimization is not dependent on the number of stages the dataset captures, but rather on the expression patterns of the single genes. We can also see that single cell datasets have the same problem as bulk data of the whole pattern being created only by a improbably small number of genes. We also observe, that the time needed for one generation is not dependent only on the number of stages but also probably on the sparsity of the sparse matrix the expressions are stored in.

Next, we are going to look at the influence of the number of permutations on the identified gene set. From the permuted gene ages, the sampled TAI patterns and their variance are computed. The variances sampling might be a major bottle neck if a lot of permutations are required. For this reason we will examine the behaviour of the optimization with different number of sampled variances.
For our analysis we chose the dataset 17. As presented in Table 9.6, this dataset spans almost 10000 stages. We will also show that the pattern is significant.

Because of the high number of stages, we cannot use `myTAI` to compute the p-value of the dataset before and after removal of the identified genes. That is why we compute the *empirical p-value* the same way as during the optimization. For this computation, we take all 50000 variances computed for the `GATAI` run with 5000 permutations. We did not recompute the p-value after actually removing the identified genes from the datasets, as we are more interested by the tendency of the p-values for different number of permutations than the actual value.

Similarly to the bulk datasets we see, that the number of permutations does not have much influence on the optimization time and number of identified genes. Yet again we see that the p-value of the solutions tends to get lower with lower number of permutations.

We also look if the identified gene sets are different for different number of permutations. We compare the identified gene sets from the `GATAI` runs with different number of permutations to the same number of runs with the same

Permutations	Perm. Time	Optim. Time	Generations	Ident. genes	p-value
100	4.13 s	541.55 s	361	29	0.73
1000	40.84 s	546.55 s	361	29	0.71
10000	407.43 s	653.57 s	431	30	0.78
50000	2046.29 s	586.57 s	381	28	0.80

Table 9.7 Permutation time, optimization time (excluding the time needed to compute the permutations), number of generations, number of identified genes and the empirical p-value for the different number of permutations, meaning different number of sampled variances.

parameters.

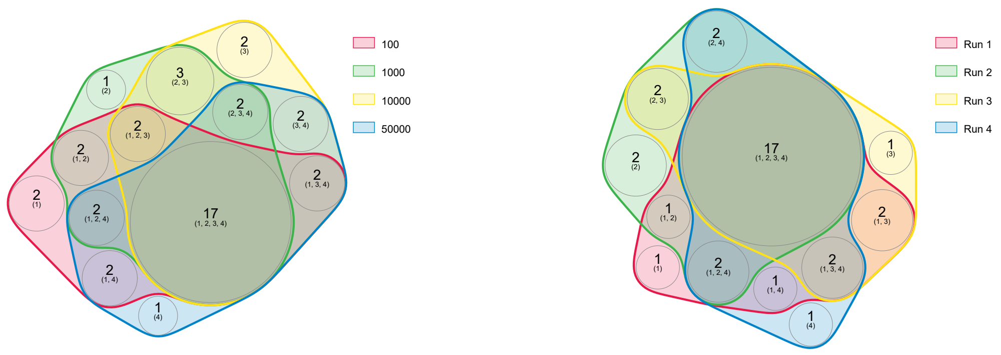


Figure 9.12 Euler plots capturing the overlap of the gene sets for (a) different number of permutations and (b) different runs of `GATAI` with the same parameters

From the Figure 9.12 we see that the number of permutation does not seem to influence the final solution. The identified gene sets do not seem to be significantly more different that gene sets identified by running `GATAI` with the same parameters every time.

Finally, we test how much faster the single cell solution is compared to not using sparse matrices and operations that are optimized for those. For this analysis, we again choose the dataset 17, which spans large number of stages and seems to have a significant TAI pattern.

As apparent from the Table 9.8, without the sparse solution, the optimization takes 10 times longer, hence for datasets that would require more generations the optimization could be infeasible.

Algorithm	**Permutation time**	**Time 10 generations**
ScGATAI	82.268 s	24.093 s
Standard GATAI	550.2 s	262.2 s

Table 9.8 Comparison of the runtime between the standard `GATAI` algorithm and the single-cell optimized version (ScGATAI).

Conclusion

Finding genes that play a key role in development has been a hard problem for the field of developmental biology for many years. Experimental methods are expensive and have a long list of limitations and the effectiveness of *in-silico* methods is limited due to capturing a lot of technical noise and the inability to filter out background effects.

In this work, we studied development from the perspective of transcriptomic and genomic phylostratigraphy and utilize the pattern of the transcriptome age index (TAI) to infer candidate genes, that are involved in development. We developed a multi-objective island model genetic algorithm `GATAI` that is able to find a small set of genes which when removed from the dataset, make the original TAI pattern disappear, where we define disappearing in terms of variance. To achieve an effective optimization, we introduced a novel mutation operator tailored for our problem and created an optimized implementation of the crossover operator. Moreover, we show that due to the island architecture, the identified gene set is stable over multiple runs. While we mostly focused on the bulk transcriptomics case where the TAI metric is established, we also implemented a version of `GATAI` that works well with the vastly larger single-cell dataset. Due to not having any constraints on the fitness function except for the speed of its computation, once a good single-cell metric is invented, the tool can be easily used for the single-cell datasets.

We test `GATAI` on many different dataset and by analyzing the resulting gene sets we show that our tool is indeed able to identify genes that are involved in development. Creating an easy to use command line tool that can be installed from `pypi`, enables the wide community to use our tool as a powerful pre-screening tool, that can identify a small gene sets to be experimentally tested, or validated by other in-silico methods.

Furthermore, we generalized our tool that is able to identify a smallest possible gene set that destroys the TAI pattern to a multi-objective genetic algorithm, that is able to minimize the size of a set of any elements that optimizes a user defined set of fitness functions. Again, we published the code to `pypi` to make it easily usable.

The field of developmental and evolutionary biology is a fast evolving field with genomic phylostratigraphy gaining more traction in the recent years. With development of the field, novel metrics and constraints may arise for bulk and single cell transcriptomics, that replace the very simplifying TAI metric. Such innovations can be easily integrated in `GATAI` by simply changing the fitness function, or adding additional ones.

Having a general subset minimizer at hand, we can explore the possibility of applying it to other multi-stage pattern problems, where the value for every stage of the pattern is computed as a summary metric over values of a set of elements.

Bibliography

1. Domazet-Lošo, Tomislav; Tautz, Diethard. A phylogenetically based transcriptome age index mirrors ontogenetic divergence patterns. *Nature*. 2010, vol. 468, no. 7325, pp. 815–818.
2. Levin, Michal; Anavy, Leon; Cole, Alison G; Winter, Eitan; Mostov, Natalia; Khair, Sally; Senderovich, Naftalie; Kovalev, Ekaterina; Silver, David H; Feder, Martin, et al. The mid-developmental transition and the evolution of animal body plans. *Nature*. 2016, vol. 531, no. 7596, pp. 637–641.
3. Quint, Marcel; Drost, Hajk-Georg; Gabel, Alexander; Ullrich, Kristian Karsten; Bönn, Markus; Grosse, Ivo. A transcriptomic hourglass in plant embryogenesis. *Nature*. 2012, vol. 490, no. 7418, pp. 98–101.
4. Gabel, Alexander. Development of a simulated annealing algorithm for uncovering the phylotranscriptomic hourglass pattern. 2013.
5. Mattick, John S; Makunin, Igor V. Non-coding RNA. *Human molecular genetics*. 2006, vol. 15, no. suppl_1, R17–R29.
6. Simon, Stacey A; Zhai, Jixian; Nandety, Raja Sekhar; McCormick, Kevin P; Zeng, Jia; Mejia, Diego; Meyers, Blake C. Short-read sequencing technologies for transcriptional analyses. *Annual Review of Plant Biology*. 2009, vol. 60, no. 1, pp. 305–333.
7. Klepikova, Anna V; Kasianov, Artem S; Gerasimov, Evgeny S; Logacheva, Maria D; Penin, Aleksey A. A high resolution map of the Arabidopsis thaliana developmental transcriptome based on RNA-seq profiling. *The Plant Journal*. 2016, vol. 88, no. 6, pp. 1058–1070.
8. Tang, Fuchou; Barbacioru, Catalin; Wang, Yangzhou; Nordman, Ellen; Lee, Clarence; Xu, Nanlan; Wang, Xiaohui; Bodeau, John; Tuch, Brian B; Siddiqui, Asim, et al. mRNA-Seq whole-transcriptome analysis of a single cell. *Nature methods*. 2009, vol. 6, no. 5, pp. 377–382.
9. Domazet-Lošo, Tomislav; Brajković, Josip; Tautz, Diethard. A phylostratigraphy approach to uncover the genomic history of major adaptations in metazoan lineages. *Trends in Genetics*. 2007, vol. 23, no. 11, pp. 533–539.
10. Barrera-Redondo, Josué; Lotharukpong, Jaruwatana Sodai; Drost, Hajk-Georg; Coelho, Susana M. Uncovering gene-family founder events during major evolutionary transitions in animals, plants and fungi using GenEra. *Genome Biology*. 2023, vol. 24, no. 1, p. 54.
11. Drost, Hajk-Georg; Gabel, Alexander; Liu, Jialin; Quint, Marcel; Grosse, Ivo. myTAI: evolutionary transcriptomics with R. *Bioinformatics*. 2018, vol. 34, no. 9, pp. 1589–1590.
12. Drost, Hajk-Georg; Gabel, Alexander; Grosse, Ivo; Quint, Marcel. Evidence for active maintenance of phylotranscriptomic hourglass patterns in animal and plant embryogenesis. *Molecular biology and evolution*. 2015, vol. 32, no. 5, pp. 1221–1231.

13. Witkin, Evelyn M. Ultraviolet mutagenesis and inducible DNA repair in Escherichia coli. *Bacteriological reviews*. 1976, vol. 40, no. 4, pp. 869–907.

14. Haffter, Pascal; Granato, Michael; Brand, Michael; Mullins, Mary C; Hammerschmidt, Matthias; Kane, Donald A; Odenthal, Jörg; JM van Eeden, Fredericus; Jiang, Yun-Jin; Heisenberg, Carl-Philipp, et al. The identification of genes with unique and essential functions in the development of the zebrafish, Danio rerio. *Development*. 1996, vol. 123, no. 1, pp. 1–36.

15. Capecchi, Mario R. Altering the genome by homologous recombination. *Science*. 1989, vol. 244, no. 4910, pp. 1288–1292.

16. Strumpf, Dan; Mao, Chai-An; Yamanaka, Yojiro; Ralston, Amy; Chawengsaksophak, Kallayanee; Beck, Felix; Rossant, Janet. Cdx2 is required for correct cell fate specification and differentiation of trophectoderm in the mouse blastocyst. 2005.

17. Nasevicius, Aidas; Ekker, Stephen C. Effective targeted gene 'knock-down'in zebrafish. *Nature genetics*. 2000, vol. 26, no. 2, pp. 216–220.

18. Ran, FAFA; Hsu, Patrick D; Wright, Jason; Agarwala, Vineeta; Scott, David A; Zhang, Feng. Genome engineering using the CRISPR-Cas9 system. *Nature protocols*. 2013, vol. 8, no. 11, pp. 2281–2308.

19. Joung, Julia; Konermann, Silvana; Gootenberg, Jonathan S; Abudayyeh, Omar O; Platt, Randall J; Brigham, Mark D; Sanjana, Neville E; Zhang, Feng. Genome-scale CRISPR-Cas9 knockout and transcriptional activation screening. *Nature protocols*. 2017, vol. 12, no. 4, pp. 828–863.

20. Liu, Pentao; Jenkins, Nancy A; Copeland, Neal G. A highly efficient recombineering-based method for generating conditional knockout mutations. *Genome research*. 2003, vol. 13, no. 3, pp. 476–484.

21. Mu, Jinye; Tan, Helin; Hong, Sulei; Liang, Yan; Zuo, Jianru. Arabidopsis transcription factor genes NF-YA1, 5, 6, and 9 play redundant roles in male gametogenesis, embryogenesis, and seed development. *Molecular plant*. 2013, vol. 6, no. 1, pp. 188–201.

22. Mackay, Trudy FC; Anholt, Robert RH. Pleiotropy, epistasis and the genetic architecture of quantitative traits. *Nature Reviews Genetics*. 2024, pp. 1–19.

23. Perez-Garcia, Vicente; Fineberg, Elena; Wilson, Robert; Murray, Alexander; Mazzeo, Cecilia Icoresi; Tudor, Catherine; Sienerth, Arnold; White, Jacqueline K; Tuck, Elizabeth; Ryder, Edward J, et al. Placentation defects are highly prevalent in embryonic lethal mouse mutants. *Nature*. 2018, vol. 555, no. 7697, pp. 463–468.

24. Serin, Elise AR; Nijveen, Harm; Hilhorst, Henk WM; Ligterink, Wilco. Learning from co-expression networks: possibilities and challenges. *Frontiers in plant science*. 2016, vol. 7, p. 185898.

25. Bonett, Douglas G; Wright, Thomas A. Sample size requirements for estimating Pearson, Kendall and Spearman correlations. *Psychometrika*. 2000, vol. 65, pp. 23–28.

26. KRIVENTSEVA, Evgenia V; KUZNETSOV, Dmitry; TEGENFELDT, Fredrik; MANNI, Mosè; DIAS, Renata; SIMÃO, Felipe A; ZDOBNOV, Evgeny M. OrthoDB v10: sampling the diversity of animal, plant, fungal, protist, bacterial and viral genomes for evolutionary and functional annotations of orthologs. *Nucleic acids research*. 2019, vol. 47, no. D1, pp. D807–D811.

27. KIRKPATRICK, Scott; GELATT JR, C Daniel; VECCHI, Mario P. Optimization by simulated annealing. *science*. 1983, vol. 220, no. 4598, pp. 671–680.

28. SCARSELLI, Franco; GORI, Marco; TSOI, Ah Chung; HAGENBUCHNER, Markus; MONFARDINI, Gabriele. The graph neural network model. *IEEE transactions on neural networks*. 2008, vol. 20, no. 1, pp. 61–80.

29. HOLLAND, John H. Genetic algorithms. *Scientific american*. 1992, vol. 267, no. 1, pp. 66–73.

30. BACK, Thomas. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

31. DEB, Kalyanmoy; PRATAP, Amrit; AGARWAL, Sameer; MEYARIVAN, TAMT. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*. 2002, vol. 6, no. 2, pp. 182–197.

32. DEB, Kalyanmoy; JAIN, Himanshu. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation*. 2013, vol. 18, no. 4, pp. 577–601.

33. ZITZLER, Eckart; LAUMANNS, Marco; THIELE, Lothar. SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK report*. 2001, vol. 103.

34. DE RAINVILLE, François-Michel; FORTIN, Félix-Antoine; GARDNER, Marc-André; PARIZEAU, Marc; GAGNÉ, Christian. Deap: A python framework for evolutionary algorithms. In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 2012, pp. 85–92.

35. GAD, Ahmed Fawzy. Pygad: An intuitive genetic algorithm python library. *Multimedia Tools and Applications*. 2023, pp. 1–14.

36. BLANK, Julian; DEB, Kalyanmoy. Pymoo: Multi-objective optimization in python. *Ieee access*. 2020, vol. 8, pp. 89497–89509.

37. LEHMAN, Joel; STANLEY, Kenneth O. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*. 2011, vol. 19, no. 2, pp. 189–223.

38. MAHFOUD, Samir W. *Niching methods for genetic algorithms*. University of Illinois at Urbana-Champaign, 1995.

39. VIRTANEN, Pauli; GOMMERS, Ralf; OLIPHANT, Travis E; HABERLAND, Matt; REDDY, Tyler; COURNAPEAU, David; BUROVSKI, Evgeni; PETERSON, Pearu; WECKESSER, Warren; BRIGHT, Jonathan, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*. 2020, vol. 17, no. 3, pp. 261–272.

40. Xiang, Daoquan; Venglat, Prakash; Tibiche, Chabane; Yang, Hui; Risseeuw, Eddy; Cao, Yongguo; Babic, Vivijan; Cloutier, Mathieu; Keller, Wilf; Wang, Edwin, et al. Genome-wide analysis reveals gene expression and metabolic network dynamics during embryo development in Arabidopsis. *Plant Physiology*. 2011, vol. 156, no. 1, pp. 346–356.

41. Gao, Peng; Quilichini, Teagen D; Yang, Hui; Li, Qiang; Nilsen, Kirby T; Qin, Li; Babic, Vivijan; Liu, Li; Cram, Dustin; Pasha, Asher, et al. Evolutionary divergence in embryo and seed coat development of U's Triangle Brassica species illustrated by a spatiotemporal transcriptome atlas. *New Phytologist*. 2022, vol. 233, no. 1, pp. 30–51.

42. Cheng, Xuanjin; Hui, Jerome Ho Lam; Lee, Yung Yung; Wan Law, Patrick Tik; Kwan, Hoi Shan. A "developmental hourglass" in fungi. *Molecular biology and evolution*. 2015, vol. 32, no. 6, pp. 1556–1566.

43. Mi, Huaiyu; Poudel, Sagar; Muruganujan, Anushya; Casagrande, John T; Thomas, Paul D. PANTHER version 10: expanded protein families and functions, and analysis tools. *Nucleic acids research*. 2016, vol. 44, no. D1, pp. D336–D342.

44. Kolberg, Liis; Raudvere, Uku; Kuzmin, Ivan; Adler, Priit; Vilo, Jaak; Peterson, Hedi. g: Profiler—interoperable web service for functional enrichment analysis and gene identifier mapping (2023 update). *Nucleic acids research*. 2023, vol. 51, no. W1, W207–W212.

45. Cantalapiedra, Carlos P; Hernández-Plaza, Ana; Letunic, Ivica; Bork, Peer; Huerta-Cepas, Jaime. eggNOG-mapper v2: functional annotation, orthology assignments, and domain prediction at the metagenomic scale. *Molecular biology and evolution*. 2021, vol. 38, no. 12, pp. 5825–5829.

46. Crécy-Lagard, Valérie de; Amorin de Hegedus, Rocio; Arighi, Cecilia; Babor, Jill; Bateman, Alex; Blaby, Ian; Blaby-Haas, Crysten; Bridge, Alan J; Burley, Stephen K; Cleveland, Stacey, et al. *A roadmap for the functional annotation of protein families: a community perspective*. Oxford University Press UK, 2022.

47. Buchfink, Benjamin; Reuter, Klaus; Drost, Hajk-Georg. Sensitive protein alignments at tree-of-life scale using DIAMOND. *Nature methods*. 2021, vol. 18, no. 4, pp. 366–368.

48. Emms, David M; Kelly, Steven. OrthoFinder: phylogenetic orthology inference for comparative genomics. *Genome biology*. 2019, vol. 20, pp. 1–14.

49. Davidson, Eric H. *The regulatory genome: gene regulatory networks in development and evolution*. Elsevier, 2010.

50. Lee, Travis A; Nobori, Tatsuya; Illouz-Eliaz, Natanella; Xu, Jiaying; Jow, Bruce; Nery, Joseph R; Ecker, Joseph R. A single-nucleus atlas of seed-to-seed development in Arabidopsis. *bioRxiv*. 2023, pp. 2023–03.

List of Figures

List of Tables

List of Abbreviations

TAI	Transcriptome Age Index
GNN	Graph Neural Network
MOO	Multi-Objective Optimization
EA	Evolutionary algorithm
GA	Genetic algorithm
MOEA	Multi Objective Evolutionary Algorithm
SPEA	Strength Pareto Evolutionary Algorithm
NSGA	Non-dominated Sorting Generic Algorithm
TPM	Transcripts Per Million
RPM	Reads Per Kilobase
CNF	Conjunctive Normal Form
SAT	Satisfiability problem
GO	Gene Onthology

A Attachments

A.1 Identified genes:

Arabidopsis Thaliana:

AT1G31330	AT2G18340	AT4G36600	AT1G16730	AT4G30880
AT5G17460	AT5G42060	AT5G48350	AT5G52300	AT1G32560
AT1G44608	AT2G35300	AT2G42000	AT1G07985	AT4G20880
AT2G42560	AT1G12845	AT3G15280	AT5G65207	AT1G17510
AT1G56415	AT1G71470	AT2G20465	AT2G30560	AT2G43530
AT3G06090	AT3G17520	AT3G24510	AT3G50970	AT5G05060
AT5G23830	AT5G42235	AT5G54220	AT5G64900	AT2G41280
AT3G50980	AT4G30450	AT5G53880	AT1G03106	AT1G07500
AT1G16025	AT1G62240	AT2G04063	AT2G46390	AT3G47836

Drosophila melanogaster:

FBgn0000216	FBgn0001174	FBgn0003060	FBgn0003683
FBgn0028537	FBgn0028544	FBgn0028855	FBgn0030186
FBgn0030390	FBgn0030539	FBgn0032282	FBgn0032538
FBgn0032987	FBgn0033721	FBgn0033855	FBgn0033942
FBgn0034201	FBgn0034204	FBgn0034802	FBgn0034819
FBgn0034828	FBgn0035544	FBgn0035547	FBgn0035548
FBgn0035582	FBgn0035858	FBgn0036600	FBgn0036605
FBgn0036606	FBgn0036607	FBgn0036717	FBgn0037178
FBgn0037180	FBgn0037181	FBgn0037261	FBgn0037430
FBgn0038009	FBgn0039434	FBgn0039436	FBgn0039437
FBgn0039441	FBgn0039444	FBgn0039678	FBgn0040393
FBgn0040813	FBgn0040842	FBgn0050457	FBgn0050458
FBgn0051626	FBgn0051813	FBgn0052570	FBgn0052694
FBgn0260011	FBgn0260954		

Bacillus Subtilis:

B4U62_01305	B4U62_02190	B4U62_02515	B4U62_04730
B4U62_04815	B4U62_05270	B4U62_05700	B4U62_06495
B4U62_06500	B4U62_06505	B4U62_06510	B4U62_06515
B4U62_06705	B4U62_07460	B4U62_07465	B4U62_09805
B4U62_09810	B4U62_10795	B4U62_11205	B4U62_11960
B4U62_12045	B4U62_13550	B4U62_14830	B4U62_15020
B4U62_15925	B4U62_16370	B4U62_16500	B4U62_18235
B4U62_19435	B4U62_21585	B4U62_22340	

Brassica Rapa:

BraA01g004810	BraA01g004820	BraA01g004830	BraA01g004840
BraA01g009090	BraA01g017840	BraA01g017860	BraA01g017870
BraA01g017900	BraA01g020050	BraA01g029230	BraA01g030130
BraA01g042550	BraA02g010420	BraA02g014350	BraA02g014460
BraA03g001080	BraA03g004440	BraA03g008390	BraA03g010670
BraA03g018260	BraA03g022130	BraA03g022820	BraA03g022990
BraA03g030630	BraA03g041720	BraA03g046490	BraA04g010970
BraA04g011050	BraA04g021770	BraA04g024630	BraA04g029560
BraA04g030230	BraA04g030260	BraA05g002560	BraA05g010190
BraA05g017760	BraA05g040240	BraA06g001200	BraA06g001990
BraA06g014120	BraA06g039140	BraA06g039150	BraA06g040810
BraA07g018290	BraA07g023010	BraA07g037640	BraA08g003880
BraA08g013400	BraA08g023470	BraA08g031770	BraA09g005520
BraA09g006970	BraA09g007610	BraA09g015830	BraA09g027140
BraA09g035160	BraA09g048290	BraA10g014430	BraA10g014760
BraA10g026450	BraA10g032510		

Caenorhabditis Elegans:

C01G6.1	C01G6.3	C02B10.4	C02E7.6	C02E7.7
C35E7.5	C50F4.6	EEED8.3	F14H12.1	F23A7.4
F23A7.8	F41F3.3	F52E1.1	F53F1.4	F53F1.5
F56G4.2	F56G4.3	K03B4.7	R12E2.7	T05E7.5
Y39G10AR.10	Y47D3B.6	Y47D7A.13	ZK180.5	