Description of the main S & T results/foregrounds:

The theoretical background of the TFBS prediction approach

Principles of comparative bioinformatics

The central hypothesis of the comparative bioinformatics is that the structure is more conserved than the sequence itself. Furthermore, the evolution is structure-dependent, therefore we can predict the structure of homologous sequences by comparing them.

When the sequences are promoter regions, the structure to be predicted is the set of transcription factor binding sites (TFBSs) situated in the promoter region. The simplest model for structure-dependent evolution is that the evolution of TFBSs is slow, while the evolution of other sites is fast. Hence, *ab initio* comparative prediction methods predict slowly evolving regions as TFBSs.

Knowledge about site-specific evolution in TFBSs is also gained. Such *a priori* knowledge can be represented in several ways, the position-specific weight matrices (PWMs) and profile Hidden Markov Models (profile-HMMs) are the two most common approaches. Using these data-representations, it is possible to found an already known motif as a TFBS in a promoter sequence in which the existence of the TFBS in question was not known so far.

Both approaches (*ab initio* prediction methods and methods based on *a priori* knowledge) can achieve intermediate/good results on predicting TFBSs. Combining the two methods might lead to a better TFBS prediction method. The TransFoot software implements such combined approach. It uses the TRANSFAC database as *a priori* knowledge about the TFBSs, and also tries to identify slowly evolving regions as potential, so-far unknown TFBSs in a similar way than our previous software, BigFoot did.

Bayesian considerations and data augmentation

Description of the model

In the background model of TransFoot, sequences evolve on a rooted evolutionary tree. The evolution consists of insertions, deletions and substitution events. Evolutionary events on different branches of the tree are independent from each other. Substitutions are modeled with continuous-time Markov models, insertions and deletions with the TKF92 model. The TKF92 model models long insertions and deletions by assuming that the sequences consist of unbreakable fragments, and these unbreakable fragments are inserted and deleted from the sequence. This means that once a long fragment is inserted into the sequence, only the entire fragment can be deleted, it is not possible to delete a fraction of the fragment. Obviously, this hardly can be justified from a biological point of view. The reason we chose this model was that the TKF92 model still describes the real evolutionary process better than the TKF91 model, which does not allow long insertions and deletions at all, furthermore, other models considering long insertions and deletions have a significantly increased computational demand compared with that of the TKF92 model. When putting the TKF92 model onto an evolutionary tree, it does not need an additional computational demand to “refragment” the sequences at the internal node, namely, to wipe away the current fragmentation of the sequence having arrived to an internal node, and choose new fragmentations for both outgoing branches, independently from each other. This means that insertion-deletion events are independent from each other on different branches of the evolutionary tree, they do not depend on each other even via common fragmentations. It can be seen that the computationally hard part is to model the continuous time dynamics of long insertions and deletions.

The evolution of sequences starts at the root of the tree with a random sequence drawn from a distribution given by an HMM describing the common structure of the sequences. The HMM

consists of a ‘slow’ state, a ‘fast’ state and profile-HMMs describing the *a priori* known TFBS motifs. After generating the ancestral sequence, each site evolves according to its structure. Slow and fast sites evolve in a similar manner, the difference is that there is a real value factor for fast sites, and evolutionary times are multiplied with this factor for fast sites. Sites belonging to a known motif evolve in the same way as the slow sites, the difference between them is in the likelihood calculations at the root: the characters at the root follow the emission distribution of the state of the profile-HMM. Inserted sites inherit the structure of their parents in case of slow and fast sites, and the state of an inserted site in a motif will be the next insertion state of its parent site in the profile HMM.

Bayesian estimations

We would like to calculate the probability of structures, given the data, namely

$$P(S|D) \tag{1}$$

where *S* is the structure, and *D* is the data. In the previous version, the structure was part of the set of parameters:

$$\theta := \{S,T,J,\lambda,\mu,r,f,M\} \tag{2}$$

where *T* is the evolutionary tree, *J* is the set of jumping probabilities in the HMM generating the structure of the ancestral sequence, λ, μ and *r* are the parameters of the TKF92 model, *f* is the factor for fast states described above, and *M* is the set of parameters in the substitution model. Let *C* denote the continuous parameters and let τ denote the tree topology. From the Bayes theorem, we have

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \tag{3}$$

and from this, $P(S|D)$ can be expressed by the marginalization of parameters

$$P(S|D) = \sum_{\tau}\int_{C} P(\theta|D)dC = \frac{\sum_{\tau}\int_{C} P(D|\theta)P(\theta)dC}{P(D)} \tag{4}$$

Calculating the integral and the summation in Equation (4) is computationally hard, therefore we do a Monte Carlo integration instead. Even the calculation of the likelihood, $P(D|\theta)$, is computationally demanding, since it involves the summation over all possible alignments. Therefore we do a classical data augmentation for sequence alignments. Let *A* denote the information about insertions and deletions happened on the branches of the evolutionary tree. Then

$$P(D|\theta) = \sum_{A} P(D,A|\theta) \tag{5}$$

and $P(D,A|\theta)$ can be calculated easily. The alignment *A* contains information about the insertion-deletion events, but not the substitution events. It is possible to sum over all possible substitution events quickly using the Felsenstein’s reverse traversal dynamic programming algorithm. Therefore characters in the sequences labeling the internal nodes are represented with the so-called Felsenstein’s wild-cards, which contain data necessary for the dynamic programming recursions (see also Holmes I, Bruno WJ. Evolutionary HMMs: a Bayesian approach to multiple alignment. Bioinformatics. 2001 Sep;17(9):803-20.). The rule-of-thumb here is to do data augmentation only as much as necessary for efficient likelihood calculations. Calculating $P(D|\theta)$ is hard due to the many possible insertion-deletion events, and not the large number of possible substitution events,

hence it is sufficient to do data augmentation only for the insertion-deletion events.

We define the augmented probability as

$$P^*(\theta^*, A|D) := \sum_S P(\theta, A|D) = \sum_S \frac{P(\theta, A|D)P(\theta)}{P(D)} \quad (6)$$

where θ^* contains the parameters except the structural information *S*. We are going to sample from the distribution

$$P^*\left(\theta^*, A|D\right) = \sum_S \frac{P\left(A, D|\theta\right)P(\theta)}{P(D)} \quad (7)$$

and estimate the distribution $P\left(S|D\right)$ by marginalizing the parameters and calculating the posterior probabilities for each position and structure HMM state, using the posterior decoding algorithm for HMMs. The sampling is performed by a Markov chain Monte Carlo method, which needs the calculation of probabilities only up to an unknown normalizing constant. Hence, Equation (7) can be rewritten in the form

$$P^*\left(\theta^*, A|D\right) \propto \sum_S P\left(A, D|\theta\right)P(\theta) \quad (8)$$

and we calculate only $\sum_S P\left(A, D|\theta\right)P(\theta)$ instead of $P^*\left(\theta^*, A|D\right)$ for each MCMC state (θ^*,*A*). For each sample (θ^*,*A*), the posterior probabilities for the structure is calculated with the posterior decoding algorithm, and the estimation for the structure is the average of these posterior decoding values.

The prior distribution is set as simple as possible. It is the uniform distribution for parameters ranging in a finite interval, these are the *r* parameter of the TKF92 model and the jumping probabilities *J*. The remaining parameters might be arbitrary non-negative numbers (edge lengths of the tree, the λ and μ parameters of the TKF92 model and the parameters in the substitution model *M*, except *f* which can be any real number greater than 1. We set the standard exponential distribution for these parameters, and the appropriately shifted standard exponential distribution for *f*.

Markov chain Monte Carlo

We would like to explore the space of the distribution $P^*\left(\theta^*, A|D\right)$ using Markov chain Monte Carlo. For this, we need transition kernels providing an irreducible chain, namely, with non-zero probabilities, it should be possible to reach any state (θ^*,*A*) from any other state ($\theta^{*'}$,*A'*) in a finite number of steps. Furthermore, the following rules must be satisfied:

- If a transition from (θ^*_1,A_1) to (θ^*_2,A_2) has non-zero probability, then the transition from (θ^*_2,A_2) to (θ^*_1,A_1) also must have non-zero probability.
- The probability of any transition must be computable at least up to a possibly unknown normalizing constant. If necessary, calculations might be restricted for a window, see the section "Correctness of the sampler" in the Lunter et al. 2005 paper (http://www.biomedcentral.com/1471-2105/6/83/)

Above these rules, we applied the rule of thumb that the transition kernel distributions should avoid flat tails if possible. We have proven cases when such flat tailed distributions cause torpid mixing of the Markov chain, see for example, Miklós, I., Mélykúti, B., Swenson, K. (2010) The Metropolized Partial Importance Sampling MCMC mixes slowly on minimum reversal rearrangement paths ACM/IEEE Transactions on Computational Biology and Bioinformatics, 4(7):763-767.

Having said this, we have the following types of transition kernels

- Changing a continuous parameter
- Changing the alignment on a subtree in a window.

Note that we do not have to walk on the annotations, as they are analytically summed out. The two types are selected with prescribed probabilities. If the first type is selected, one of the parameters is selected with uniform probability, and a standard Metropolis-Hastings algorithm is applied to modify the parameter. The method of alignment changing happens in the same way as described in Miklós, I., Novák, Á., Satija, R., Lyngsoe, R., Hein, J. (2009) Stochastic Models of Sequence Evolution including Insertion-Deletion events, Statistical Methods in Medical Research, 18:453-485. We do not describe here the approach in detail, readers are referred to this paper, as well as to the Java source code documentation and the source code itself. When a window is selected in resampling an alignment, the window length follows a distribution that is the convex combination of a peaked distribution (binomial distribution) and the uniform distribution. This provides that the distribution will not have a flat tail, however, the expectation of the window length might be controlled. This is especially important since the new alignments are chosen using the Forward-Backward sampling algorithm of a pair HMM (see for example, Durbin et al., Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids, Cambridge Univ Pr, 1999), whose running time grows with the product of the length of the sequences to be realigned. Therefore it is desired that the expected window length should be approximately the square root of the sequence lengths, thus the running time of the realignment step grows only linearly with the sequence lengths.

The likelihood of the newly proposed state must be calculated for the Metropolis-Hastings ratio. Some changes affect the whole state (θ,A), in these cases, the likelihood calculations must be started from scratch. An example for this is the change of any of the parameters in the TKF92 model. However, there are changes that affect only part of the current state. For example, changing an edge length affects only the likelihood calculations above the edge in the evolutionary tree. We implemented several methods that speeds up the likelihood calculations by recalculating only those parts that are affected by the proposed change.

Implementation and test of the TransFoot package on real biological data

High level description of the program

We show the general description of the TransFoot program on Figure 1. First, the program reads input data (genomic DNA from promoter regions), prior information and parameters. From this, an initial MCMC states is created, then the random walk on the state space starts. Samples from this random walk are generated, the uninteresting parameters are marginalized, and the results are visualised.

The built up of the MCMC start state is comprised of the following tasks:

- Building a rooted evolutionary tree
- Putting a sequence alignment onto the tree
- Selecting the TRANSFAC motifs building the HMM that generates the ancestral sequence

We discuss these tasks in details below.

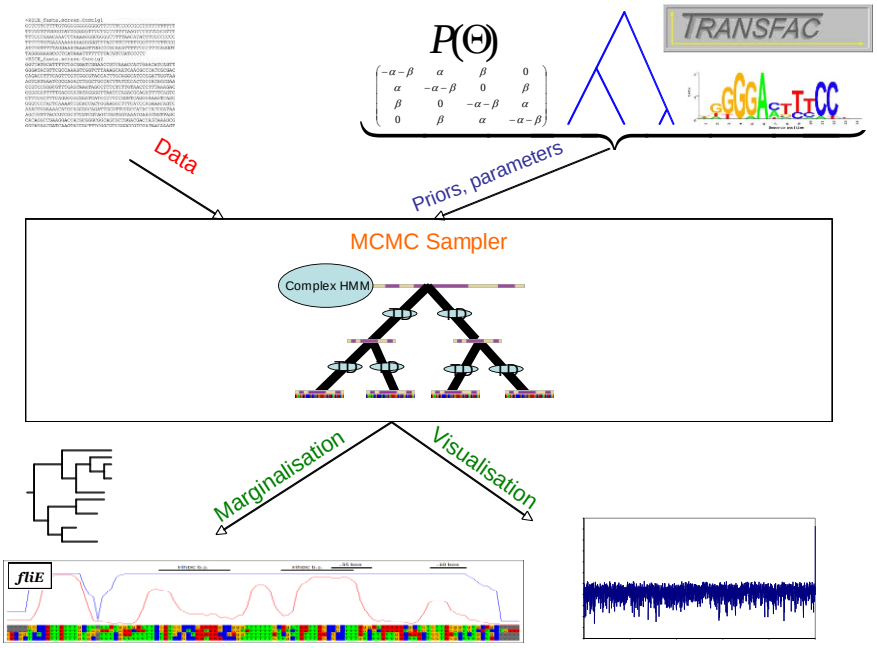


Figure 1. The schematic picture of the TransFoot software.

Building a rooted evolutionary tree

The topology of the rooted evolutionary tree might be given by the user or must be generated by the program. There are several tree-building methods like Neighbor Joining and UPGMA. In the prototype, we implemented the UPGMA method so far, and adding further methods is an option for further software development.

Putting a sequence alignment onto the tree

There might be two cases: either the user gives a multiple alignment of the sequences or not. In case a multiple sequence alignment is given, the user might ask to construct an alignment on the tree that agrees with the alignment of observed sequences. If there is no such constraint, the alignment on the tree might be constructed in any way.

In the current implementation, we consider only this later case. The sequences are aligned together in a progressive way. First, the pairs of sequences in cherries are aligned via a common ancestor being in the root of the cherry motif. Once the sequences in cherry motifs are aligned, their common ancestors are treated as sequences labeling the leaves, and the pairs of sequences are aligned in the so-emerging new cherries. This procedure is continued till we reach the root of the tree, the last cherry.

One more possible option is to build the tree and initial alignment simultaneously. Such a simultaneous built might or might not be the subject of constraint given by the initial alignment.

As can be seen, there are a large variety of options how to build the initial state. We opted to dedicate an individual class for each option, and thus, Tree is an abstract class and any extensions differ in the construction of the tree. In this way we can avoid that a single class have a huge list of constructors, with some artificial parameters for overriding reasons, and the class itself be a giant class.

Selecting the TRANSFAC motifs building the HMM that generates the ancestral sequence

There are thousands of known motifs in the TRANSFAC database, and our knowledge on known motifs can only increase, thus the number of known motifs. For any input data (here we talk about a set of homologous promoter regions as input), the number of known motifs appearing in the region will be very limited. A fast filtering method that filters out the motifs that are very unlikely to be in the set of homologous promoter regions can improve the performance of the TransFoot program. Without such filtering, the MCMC would spend the majority of its running time by proposing motifs that are not appearing in the analyzed sequence. Though these proposals are rejected due to small likelihood values, the Markov chain will not move, which naturally decreases its mixing time.

Obviously, if we filter out a motif that appears in the promoter region, we are unable to correct this error later in the MCMC analysis. Therefore, the filtering should be designed to minimize the false positive errors while trying to reduce the false positive errors. Another constraint is to keep the overall running time of this filtering low. Our filtering strategy is to consider only those motifs that can be found in any promoter region with at most two mismatches. This filtering method is based on the suffix-tree of the known motifs, and can simultaneously scan the promoter sequences for all motifs, thus being a very efficient implementation. We also implemented a filtering method that selects TRANSFAC motifs based on the posterior decoding values of their profile-HMM, however, this method turned out to be quite time-consuming.

We cannot say that the above strategies are optimal in any sense. We have to keep in mind that the initial state affects only the time needed for the Markov chain to converge to the prescribed distribution $P^*\left(\theta^*, A|D\right)$. Though we can save quite a lot of running time by reducing the convergence time, it does not gain any if it comes together with an extremely increased running time needed to build up the initial state.

Once the initial state is constructed, the Markov chain starts. It takes a prescribed number of burn-in steps, and then generates a prescribed number of samples taking another prescribed number of steps between two samples. From the sampled states, the fraction a site spends in slow/fast/known motif state is calculated for each site in the promoter sequences.

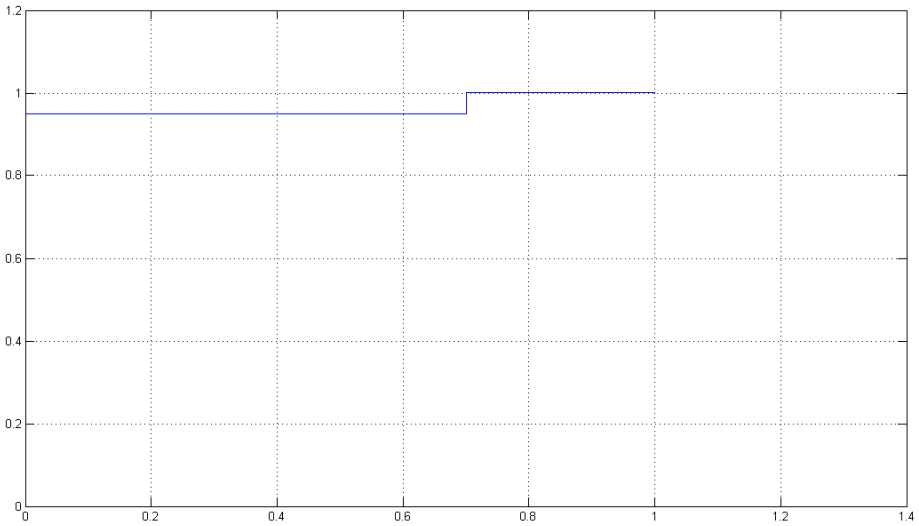
Test on real biological data

We tested the final version of TransFoot on 20 TRANSFAC motifs. The motifs were mapped onto the hg19 human genome, the genomic multiple alignment around this region were cut, and the monkey sequences were collected from this multiple alignment. The raw sequences were the input of the program.

The average posterior probabilities on the motif and outside of the motifs were calculated. We got the following values, first value is the posterior probability of the motif at the known place, the second is the posterior probability of the motif outside of the known position:

R29741: 0.9472199243581549, 0.00811359615688335
R03477: 0.9536383485666275, 0.035313570879946174
R26472: 0.9137539590482481, 0.0268487974352408
R21652: 0.9643489359206868, 0.17411383407455439
R02909: 0.9007985793644695, 0.02655168743015786
R14422: 0.9001635334771052, 0.004219410075880571
R26596: 0.9483347153493611, 0.02444304634845193
R00148: 0.9532165831294592, 0.10815414068425207
R01851: 0.8213738187809038, 0.19449820419936117
R13028: 0.7511557019910684, 0.08575918145358374
R04789: 0.634846054778527, 0.13151630366775463
R05034: 0.792359665944077, 0.011786465813487298
R16051: 0.9256226318163723, 0.12012744789634712
R09600: 0.5513885370000025, 0.12884982555013522
R13476: 0.8405098862122383, 0.029829751504346736
R16992: 0.025941029526038828, 0.0011642708730035505
R17045: 0.9081862765010342, 0.06404988019788965
R14437: 0.7178527773704112, 0.10399336304004295
R08648: 0.8774942716354668, 0.02995393829148741
R27636: 0.9618611298861386, 0.011834662687849693

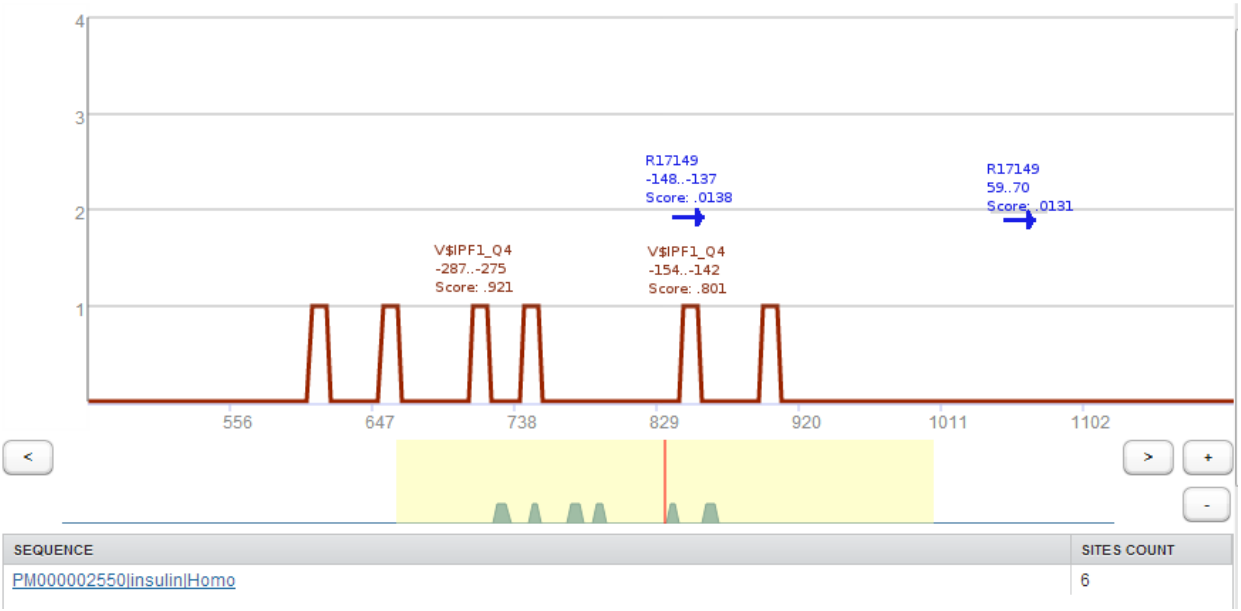
The corresponding ROC curve is:



If we set the critical value for the posterior probability anywhere between 0.2 and 0.55, we can predict 95% of the known motifs correctly without making any false positive error.
We also illustrate the applicability of the TransFoot package on the promoter region of the INS2 and CYP1A1 genes, comparing it with other top quality TFBS prediction methods. In the insulin promoter there should be multiple IPF1 (PDX1) binding sites around -200 and around -100. Sequences were from -1000 to 500 around transcription start for human, mouse and rat (not aligned). We run transfoot (fast) algorithm to predict sites for IPF1. In motifset_cl there is only one motif, R17149 for IPF1 transcription factor.
Transfoot predicted the following sites: 101-111, 158-169, 254-264, 353-362, 714-726, **852-863**, **1059-1070**, 1362-1373. Bold sites are sites with strong evidence.
The classical match algorithm with p-value 0.02 predicted 6 sites:

(Sequence_name	start	end	matrix_id	score	strand)
PM000002550|insulin|Homo	610	622	V$IPF1_Q4	766	+
PM000002550|insulin|Homo	655	667	V$IPF1_Q4	726	-
PM000002550|insulin|Homo	**713**	**725**	**V$IPF1_Q4**	**921**	**+**
PM000002550|insulin|Homo	744	756	V$IPF1_Q4	711	+
PM000002550|insulin|Homo	**846**	**858**	**V$IPF1_Q4**	**801**	**+**
PM000002550|insulin|Homo	899	911	V$IPF1_Q4	676	+

Bold sites are also sites with strong evidence.
So, comparing to match, transfoot confirmed that there is a binding site around position 850, that is -150 in promoter coordinates, and found a new site in postiton 59-70 that was not found by match.



In the CYP1A1 promoter there should be multiple Ahr:Arnt binding sites, e.g. around -1000
CYP1A1 promoter -1500..500 for human, mouse and rat
Transfoot
1 site R25443
475-499, 518-526, 582-607, 837-862, 976-1001, **1082-1108**, 1208-1233,
Match

Undefined	103	119	V$AHRARNT_01	924	-
Undefined	284	300	V$AHRARNT_01	801	-
Undefined	362	378	V$AHRARNT_01	762	-
Undefined	**502**	**518**	**V$AHRARNT_01**	**818**	**-**
Undefined	**590**	**606**	**V$AHRARNT_01**	**851**	**-**
Undefined	660	676	V$AHRARNT_01	761	+
Undefined	981	1000	V$AHRARNT_02	765	-
Undefined	985	1001	V$AHRARNT_01	773	-
Undefined	**1088**	**1104**	**V$AHRARNT_01**	**836**	**-**
Undefined	1426	1442	V$AHRARNT_01	800	+

-1000 in promoter coordinates corresponds to 500 in sequence coordinates
In this case Transfoot performed even better, because its sites are more concentrated near desired region.

The MatchPortal

Overview

The MatchPortal facilitates TFBS prediction on large scale sequences using position-specific weight matrices. Beyond mere calculation and retrieval of predictions, the MatchPortal enables filtering of search results according to genomic intervals of interest. This combination of functionality was chosen to cover a wide range of use cases. Furthermore, the system provides for data management of sequences, PWMs, genomic intervals and search results.
A particular goal of the MatchPortal is to enable programmatic access via web services. Therefore, individual actions such as sequence or PWM upload as well as intersection of genomic intervals with binding site predictions can be performed both via the graphical user interface and on the programmatic level. This way, binding site searches can be included into custom pipelines and complex workflows, yet provide for the necessary freedom with respect to the customizable choice of sequences, PWMs or genomic intervals.
Several data sets have been pre-loaded and are accessible by default in every user account including guest accounts. Pre-loaded data sets encompass reference genomes for human (hg19) and mouse (mm9), the public part of the Transfac PWM library, and conserved segments of the human genome as computed by the PhastCons algorithm on a set of mammalian genomes. Furthermore, binding site predictions have been calculated for the two reference genomes using included PWMs at a P-value threshold of 1e-4. Altogether, the pre-loaded data sets provide for a useful basis that can be easily complemented with user data such as custom PWMs, sequences or custom genomic intervals. Although mentioned data sets have a focus on gene regulation analysis in mammalian systems, it should be noted that the methods provided by the MatchPortal are not restricted to this group of species and can be applied as well to other taxa. A binding site analysis workflow using the graphical interface is demonstrated below.

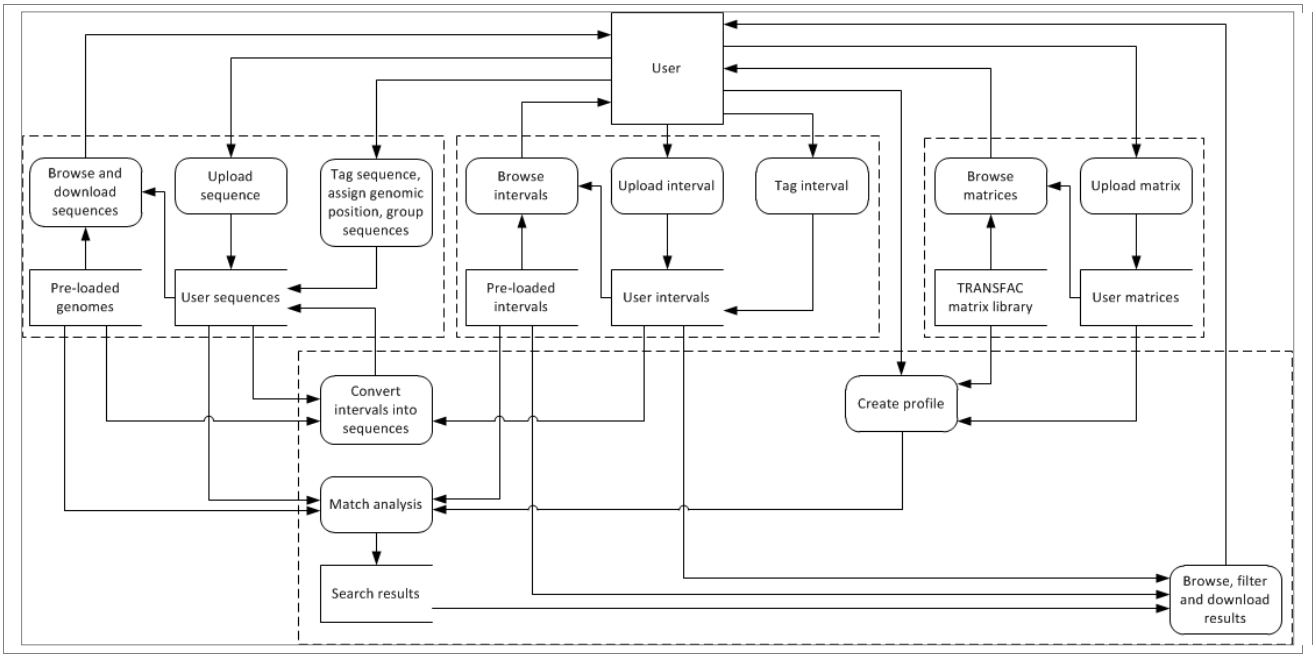


Figure 2. The standard data flow diagram (DFD) of the MatchPortal webservice. It is shown what kind of data a user can provide to or retrieve from the system in various formats. The diagram also indicates how the pre-loaded genomic data and matrix libraries are incorporated besides the user content. The 4 boxes highlights the 4 main parts of the portal: uploading and tagging sequences, uploading and tagging intervals, uploading and using score matrices, and the TFBS prediction and retrieving the results.

The structural design of the MatchPortal is depicted on Figure 2, along with its features, namely manipulating sequences, intervals and PWMs and running Match analysis. It is shown what kind of data a user can provide to or retrieve from the system in various formats. The diagram also indicates how the pre-loaded genomic data and matrix libraries are incorporated besides the user content. Each function is available both through the graphical interface and as through a web service. The "**Materials and methods involved in MatchPortal**" section below also contains the description of the used methods and algorithms. Several sophisticated algorithms have been implemented in MatchPortal, including efficient P-values calculations, sophisticated algorithms on suffix arrays and linear algorithm for finding significant matches of position weight matrices.

Code structure

The MatchPortal and MatchServices code structure consists of several levels. MatchPortal is implemented based on the Model-View-Controller architecture. It calls the MatchServicesLibrary that performs scheduling tasks and data retrieval. Tasks are executed by workers. Then data from tasks goes to the database and displayed in MatchPortal.

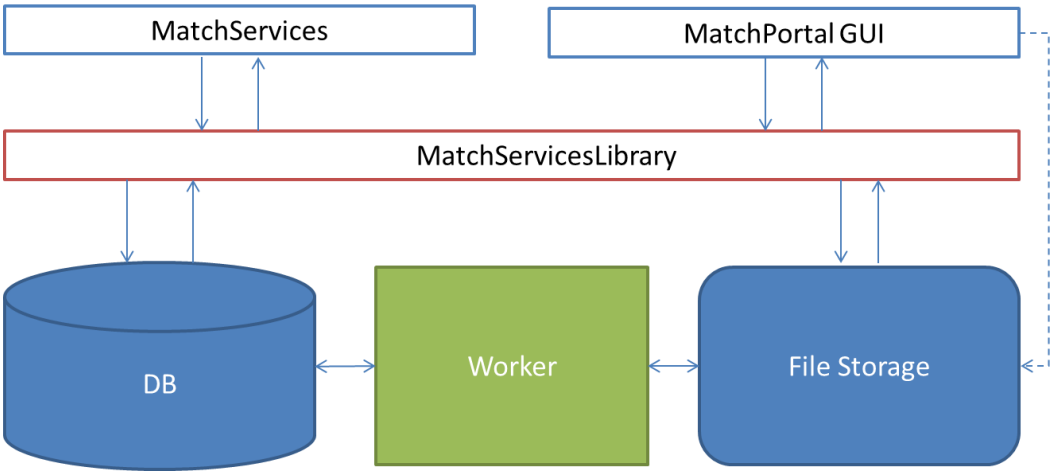


Figure 3. MatchPortal overall code organization. Match SOAP services and MatchPortal GUI interact with the MatchServicesLibrary that connects to database and file storage. Separate instances of "worker" processes are used for launching large-scale processes.

MatchPortal is implemented using the Vaadin framework (https://vaadin.com). It allows building web-based interfaces using the Java language and Vaadin libraries. MatchPortal contains several views that represent different forms and tables. Each view creates and calls a controller (sequence controller, matrix controller, etc).The controller creates an appropriate library instance and prepares data to call a library function to retrieve data or run the analysis. A sites and frequencies visualizer is implemented as separate Vaadin widget.
The MatchServiceLibrary contains beans that represent simple atomic structures like matrices, profiles, sequences, intervals and interval files. Those beans are created when necessary from the database and sent to the MatchPortal or MatchServices by request. Each bean has a database wrapper – active record. This record fills a bean with corresponding fields from database, or saves a given bean to database. There are also several utility classes and classes representing services logic (manipulation with beans, providing data and schedule tasks).

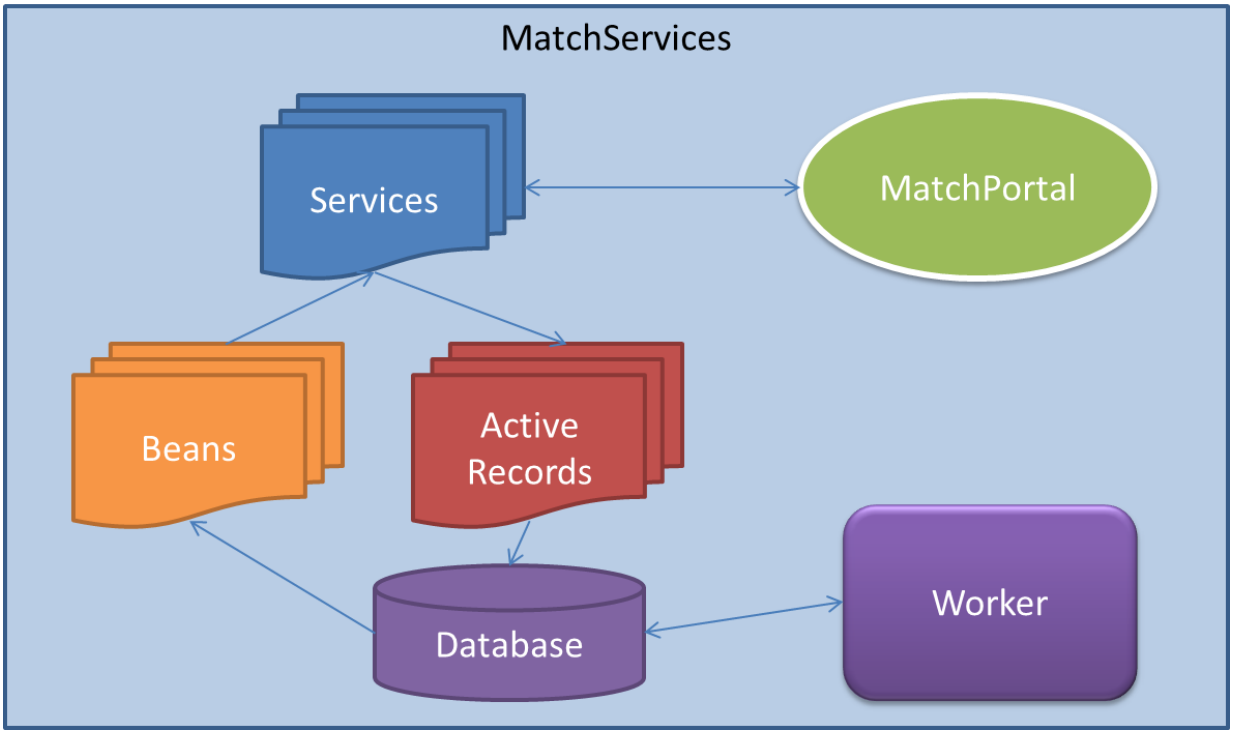


Figure 4. MatchServiceLibrary organization. Services use ActiveRecords classes to request database and retrieve information in beans.

Worker is organized in the way to have multiple instances of workers running on the same machine. Each worker can process one task at a time. Workers take tasks from a global pool of tasks and mark taken tasks as Running. When a task is finished it is marked finished and results are present in the database. For some tasks file storage is also used. Sequences are stored in files; Match configurations are stored in files. Match outputs are stored in a separate database.

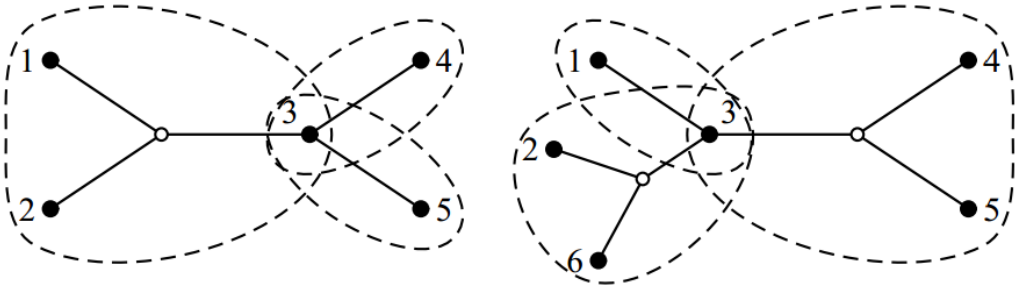
The phylogenetic segmentation module

Overview

Phylogenetic segmentation provides a sensible way to decompose the input rooted evolutionary tree into a number of smaller overlapping components (segments) that can be analysed separately with the above-described TransFoot algorithm. The TFBS prediction results provided by TransFoot for each component can be transferred to further components to be analysed as prior information about the sequences in which the consecutive components overlap. As a result, the prediction of TFBSs obtained for a component depends both on the Transfoot analysis of the component itself and also the prior information transferred from previous components. The components to be analysed are ordered in a way that the "interesting sequence" for which the user is interested to have TFBS annotations predictions (referred to as the target sequence) belongs to the last component analysed. Therefore the predicted annotations of the target sequence depends on information gathered from the entire phylogenetic tree (i.e. from evolutionarily more distant sequences than those belonging to the same component). On the other hand, since the analysis of a series of small tree components is far less time-consuming than the analysis of the original input tree as a whole, phylogenetic segmentation extends TransFoot's capability by allowing to analyse a larger number of sequences in realistic time.

The segmentation package offers a set of alternative optimization schemes that aim to transform the input tree into a k-restricted Steiner tree (i.e. a Steiner tree where the maximum number of terminal nodes in any full component is at most k). We refer to the full components of a Steiner tree as the set of subtrees spanning the terminal nodes of the original phylogenetic tree (i.e. the input sequences). Note that full components are independent except for shared terminal nodes. These are the overlapping terminal nodes through which we can transfer information between the connected segments as priors. Approximating the input tree as a k-restricted Steiner tree involves modifying the original phylogeny and the goal is to introduce as few changes to the original tree as possible. Therefore the alternative segmentation schemes aim to minimise the difference between the original and the modified tree. To this end, however, they use a series of different measures to be optimized (see Optimization Schemes).

The following figure shows two examples of 3-restricted Steiner trees. If the node terminal node labelled 3 had been a Steiner node, the left tree would have been a 4-restricted Steiner tree and the right tree would have been a 5-restricted Steiner tree, and both trees would have been unrestricted Steiner trees. The dashed lines enclose the full components of the trees.



Optimization schemes

1. Minimise contraction (MC)

Finds the k-restricted Steiner tree that minimises the total length of contracted edges.

2. Minimise contraction power (MCP)

Finds the k-restricted Steiner tree that minimises the total length of contracted edges raised to a power p (specified as a parameter).

3. Minimise extension (ME)

Finds the k-restricted Steiner tree that minimises the total length of remaining edges when they are seen as being elongated to the terminal node of incident contracted edges.

4. Minimise extension power (MEP)

Finds the k-restricted Steiner tree that minimises the total length of remaining edges raised to a power p (specified as a parameter) when they are seen as being elongated to the terminal node of incident contracted edges.

5. Minimise move (MM)

Finds the k-restricted Steiner tree that minimises the total amount internal nodes are moved, when they are seen as being moved to the leaf that they are connected to by a series of contracted edges.

6. Minimise move power (MMP)

Finds the k-restricted Steiner tree that minimises the sum of internal node movement raised to a power p (specified as a parameter), when internal nodes are seen as being moved to the leaf that they are connected to by a series of contracted edges.

7. Minimise relative edge lengths (MRE)

Finds the k-restricted Steiner tree that minimises the product of relative length increases for remaining edges, when they are seen as being elongated to the terminal node of incident contracted edges.

8. Minimise sum of `slacks' (MS)

Finds the k-restricted Steiner tree that minimises the sum of differences between the sizes of full components created and the maximum size of full components.

9. Minimise slack power (MSP)

Finds the k-restricted Steiner tree that minimises the sum of `slacks' raised to a power p (specified as a parameter), where slack is the difference between the size of a full component created and the maximum size of full components.

10. Pivot minimise lift (PML)

Finds the k-restricted Steiner tree that minimises the sum of distances leaves are lifted, when a pivot is seen as lifting the leaf to the internal node rooting the subtree of the pivot.

11. Pivot minimise slack (PMS)

Finds the k-restricted Steiner tree that minimises the sum of differences between maximum component size and actual component sizes, when a pivot is seen as lifting the leaf to the internal node rooting the subtree of the pivot.

12. Pivot minimise slack and lift (PMSL)

Finds the k-restricted Steiner tree that minimises the sum of differences between maximum component size and actual component sizes (as in scheme 11), but breaking ties by minimising the sum of distances leaves are lifted (as in scheme 10), when a pivot is seen as lifting the leaf to the internal node rooting the subtree of the pivot.

Transferring information between components

As described above, the tree components are processed by TransFoot in a reverse order so that the component containing the user-defined target sequence is the last to be analysed preceded by the components with which it has shared terminal nodes and so forth. For each component, the average posterior decoding profiles obtained from preceding components for the shared terminal nodes are used as prior information.

Let *p* denote the average posterior decoding probability of a given HMM state in a given position of sequence *A* calculated based on a full component C_1. Furthermore, let *N* be the total number of (non-silent) HMM states when analysing the next full component C_2 of which sequence *A* is a shared node with component C_1. First, the posterior probability *p* is transformed to the $G_k(p)$ prior probability using the following simple function:

$$G_k(p) = p + \left(\frac{1}{N} - p\right) k$$

where *k* is a constant from the interval of [0,1] referred to as the "uncertainty factor" representing our uncertainty of the accuracy of the posterior probability p calculated from the preceding component C_1. The larger the value of *k* is, the more the transformation contracts the input [0,1] interval of *p*. In case *k=0* (i.e. we have 100% confidence about correctness of the posterior), the transformation leaves the value unchanged. In the other extreme, if *k=1* (i.e. we have 0% confidence about correctness of the posterior), the prior will be of uniform distribution: $G_k(p) = 1/N$.

At the end points of the [0,1] interval of *p*, we get:

$$G_k(0) = \frac{k}{N} \qquad G_k(1) = 1 - \left(1 - \frac{1}{N}\right) k$$

Consequently, for *k>0*, the HMM states that have zero posterior decoding probability in component C_1 will always have non-zero prior probability in component C_2. This is especially important, since in case the prior was 0, the posterior would also be 0 for component C_2, independently of what TransFoot predicts for that component.

Furthermore, the $G_k(p)$ transformation ensures that the prior probabilities of the different HMM states also sum up to 1:

$$\sum_{i=1}^{N} G_k(p_i) = \sum_{i=1}^{N} \left[p_i + \left(\frac{1}{N} - p_i \right) k \right] = \sum_{i=1}^{N} p_i + \sum_{i=1}^{N} \frac{k}{N} - k \sum_{i=1}^{N} p_i = 1$$

At each step of the MCMC, TransFoot calculates the emission probabilities of different HMM states for every position of the root sequence represented by Felsenstein wildcard characters. When using the phylogenetic segmentation module, prior probabilities are taken into account at this step in the following way.

Let *f* denote a given Felsenstein wildcard character of the root sequence at the position where we are calculating the emission probability of HMM state *x*. Let *E(f,x)* be the emission probability calculated by TransFoot when no phylogenetic segmentation is used. The new emission probability *E'(f,x)* modified by the prior is calculated as

$$E'(f,x) = E(f,x) \cdot \prod_{A,i} \frac{1}{N_c} \sum_{c} G_k \left[p_c(A,i,x) \right]$$

where $p_c(A,i,x)$ denote the average posterior decoding probability of HMM state *x* for position *i* of sequence *A*, obtained from component *c*. Furthermore, N_c is the total number of components with which the current component overlaps in sequence *A*. When there is more than one component providing priors to the current component, the average of these prior values are used. The product runs over all sequence positions (*i*) of the shared terminal nodes (*A*) that are descendents of the Felsenstein wildcard character *f*.

The ExPlain System

We implemented several algorithms to improve prediction of TFBS and integrated them into ExPlain system. An integrated computational system, ExPlain [1] was developed for causal interpretation of gene expression data and identification of key signaling molecules. The system utilizes data from two databases (TRANSFAC and TRANSPATH) and integrates two programs: (1) Composite Module Analyst (CMA) analyses 5'-upstream regions of co-expressed genes and applies a genetic algorithm to reveal composite modules (CMs) consisting of co-occurring single TF binding sites and composite elements; (2) ArrayAnalyzer is a fast network search engine that analyses signal transduction networks controlling the activities of the corresponding TFs and seeks key molecules responsible for the observed concerted gene activation.

Correlation with context features

We built the algorithm of identification of correlatively mutated position in each TFBS and integrated it into ExPlain. By correlative mutations we mean mutations in two or more positions that changed nucleotides in two or more sequences. Example of correlative mutation is represented in figure 5.

```
TTGACA monDom5
TTGGCA rheMac2
TTGGCA canFam2
TTGGCA panTro2
TTGGCA oryCun2
TGGGCA cavPor3
CTGGCA mm9
CTGGCA rn4
TCAGCA loxAfr3
TTGGCA papHam1
TTGGCA equCab2
TTGGCA calJac1
CTGCAG galGal3
TTGGCA hg19
CCAGT- xenTro2
TTGGCA bosTau4
TTGGCA ponAbe2
```

Figure 5. Example of correlative mutation in site R00078. In the positions 2 and 3 of loxAfr3 and xenTro2 there is correlative change of TG pair to CA pair.

Correlative mutations can be one of two kinds: monophyletic and paraphyletic. Monophyletic mutations are the mutations that arise in closely evolutionary related species. This is most possibly one mutation that arises in ancestor sequence and thus is represented in the descendants. Those mutations are not the case of our interest, because we need mutations, which happen independently in distantly related species. Those mutations, called paraphyletic, may influence each other and keep the TFBS in place. In this case, if one mutation presents then other should also be to keep regulation properties of fragment. Mutations from previous example are considered paraphyletic, as it is shown on the evolutionary tree shown in figure 6.

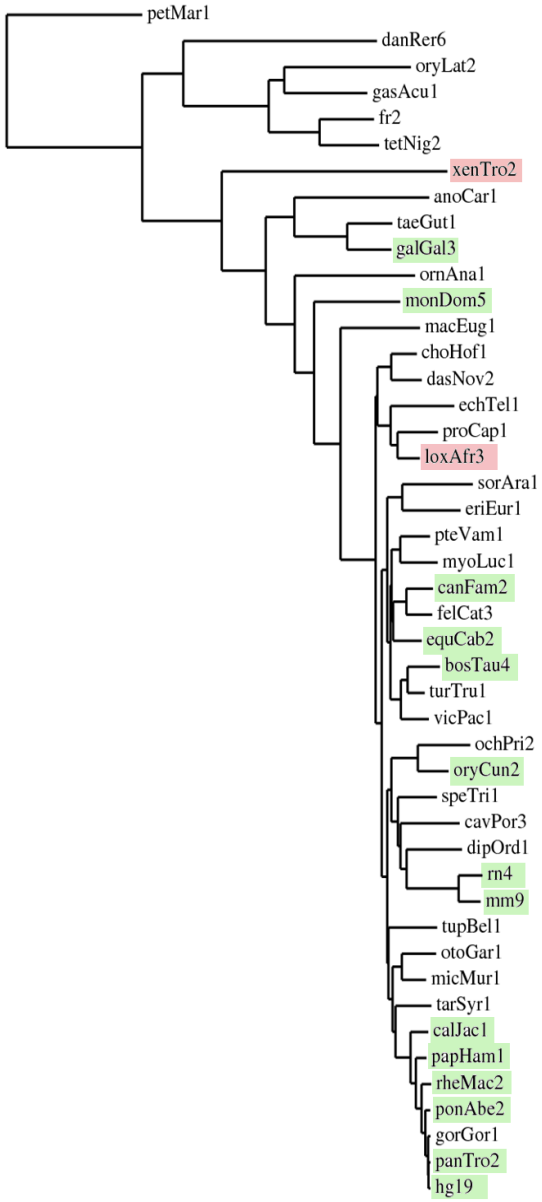


Figure 6. Evolutionary tree of aligned species. Green boxes represent basic species. Red boxes represent species with mutation in positions 2, 3 of site R00078. We can see that species loxAfr3 and xenTro2 are distantly related, so the mutations are considered paraphyletic.

The algorithm that detects correlative mutations is the following:
Input: alignment A of binding site from TRANSFAC and evolution tree T.
Algorithm:
For each pair of positions in alignment we search for most frequent set of so called distinct pairs of nucleotides. To do that, the substeps are the following:
For each pair of alignment positions we fill matrix C[4,4] by frequency of each combination (AA, AC, AG, …., GG).
For each pair by given matrix C we find 4 positions (i_1,j_1), (i_2,j_2), (i_3,j_3), (i_4,j_4) where $i_k <> i_t$ for each k<>t and $j_k <> j_t$ for each k<>t. In those positions sum of weight in C should be maximal: $sum = \sum_k C(i_k, j_k)$. This is done by applying dynamic programming technique. If value of $C(i_k, j_k)$ is 1 then this pair (i_k, j_k) is not considered further.
We do not consider pair of alignment positions if it is not satisfies some cutoff condition: ***sum/M < 0.8***, where M is the number of species in alignment.

Now for some pairs of alignment positions we have 2-4 pairs of frequently met nucleotides.
For example for the alignment from figure 3 for pair of alignment positions (2, 3) we will have two pairs: a pair TG with frequency 14/17 and pair CA with frequency 2/17. Now we should check for evolutionary tree constraints.
For each pair of positions we determine if our distinct pairs are paraphyletics. We take the most frequent distinct pair (the pair with maximal $C(i_k, j_k)$) as the basic feature.
Then we test other features: for each feature we consider the species in which this feature is observed. Those species should not have common parent: between them there should be at least one species with basic features.
By using the procedure described above we will find all paraphyletic correlation features.

We developed this algorithm as a Java program and integrated it into ExPlain as separate analysis algorithm. It takes a motif set as input and finds all paraphyletic correlation features in each alignment.
We found 37 paraphyletic in 2375 alignments. The list of paraphyletics is:

Align-ment	Col1	Col2	Basic feature	Par. feature	Score
R00078	2	3	TG	CA	0.941
R00380	1	2	CA	TG	1
R00422	15	18	CC	TT	0.867
R02794	4	32	AT	GC	0.938
R04378	2	7	AG	CC	0.938
R04379	2	7	AG	CC	0.938
R08298	1	24	AG	GA	0.8
R09992	12	16	AT	GC	1
R13030	3	5	GA	CG	0.867
R13058	34	35	CT	TG	0.8
R14480	16	17	TG	CA	0.933
R15111	6	34	CG	TA	0.933
R15902	1	4	CC	TT	0.867
R16614	1	18	GG	CC	0.842
R16988	10	11	TC	GA	1
R17064	2	57	TT	CG	0.929
R19631	8	16	GA	AC	1
R20451	19	38	CA	TG	1
R20754	1	6	TC	CT	1
R20869	5	21	TA	CG	1
R20958	8	10	TT	CC	0.929
R21193	9	10	CT	TC	1
R21390	2	38	TT	GC	0.938
R21850	14	16	TC	CT	0.824
R22178	22	71	TA	CG	1
R23895	19	22	CC	TG	0.929
R23903	19	22	CC	TG	0.929
R24494	6	38	GG	CA	1
R24495	6	38	GG	CA	1
R25336	38	39	CT	TG	0.8
R25339	38	39	CT	TG	0.8
R25983	22	26	TT	CC	0.833

R26218	1	4	GG	AA	0.929
R26221	34	35	CT	TG	0.8
R27895	9	19	TC	CT	0.938
R29177	23	85	GC	AT	1
R29765	6	24	TT	CC	0.867

Paraphyletic update for match score calculation

First we improved the matrix search with paraphyletic information from the previous algorithm and tested this approach on the binding sites that were used for Transfac® matrix construction. At first sight, the paraphyletic search performs worse than the naïve search. This can be explained by the conditioning of the matrix and selection of binding sites in the matrix construction process, where an optimal set of binding sites is selected using the search algorithms. Hence, it is clear that matrices and binding sites are optimal for the applied search algorithm and paraphyletic information disturbs matrix optimality. We then decided to extend the search algorithm by using alignments instead of single sequences for binding site prediction. We can calculate a matrix score on each sequence in alignment. Then using this information we can calculate a score for alignment and add a further parameter if we observe both basic and paraphyletic features in this alignment.

Our results demonstrate that with selected cutoffs the paraphyletic search achieves better binding site prediction results than the simple matrix algorithm. Generally, the ROC-curve gives less area which is connected with that we took the same set of sites for testing that was used for building PWM.

Input data:

1. set of paraphyletic search results loaded from file one record consists of following fields:
   String alignmentAccession - accession for alignment where paraphyletic feature was found
   String hg19Site – hg19 subsequence without gaps
   int col1, col2 – column numbers where paraphyletic feature was found
   char[2] pf – two symbols of paraphyletic feature for corresponding columns col1 and col2
   char[2] bf – two symbols of basic feature for corresponding columns col1 and col2
   several results can be found for one accession, we should take into account all them.
   transfac matrix
   input site

The goal is to calculate a new score for a putative site in the primary species (in our case human) which will use information about paraphyletic features identified for binding sites in the Transfac(R) matrix training set.

Let us describe the score calculation that incorporates the paraphyletic information.

Assume that Transfac® matrix **tfm** contains a binding site with accession, which also was found in a set of paraphyletic search results.

String **accession** – accession of alignment where paraphyletic feature was found
int **col1, col2** – column no which contain features
char **bf1, bf2** – symbols of basic features
char **pf1, pf2** – symbols of paraphyletic features

```
currentScore = calcOriginScore(site,tfm);
// loop for paraphyletic results where accession in tfm.accessions
foreach (pf1, pf2, bf1, bf2, col1, col2, accession)
if (accession in (tfm.accessions)) {
       // calculate additional paraphyletic score
       addScore =    max( tfm.score[col1][bf1] + tfm.score[col2][bf2],
                     tfm.score[col1][pf1] + tfm.score[col2][pf2] );
       oldScore =    tfm.score[col1][site[col1]] + tfm.score[col2][site[col2]];
       if ((site[col1]==bf1 && site[col2]==bf2) ||
              (site[col1]==pf1 && site[col2]==pf2)) {
              currentScore -= oldScore;
              currentScore += addScore;
       }
}
```

To test the new approach we should calculate score for all alignments.
Let us describe this algorithm

Assume that for one alignment we have

TfMatrix **tfm** – transfac matrix,
String[] **pfSites** – set of sites for one alignment,
String[] **pfSpecies** – set of corresponding species
double **calcPfScore**(TfMatrix tfm, String site...) – routine for calculating site score. Described above.

Also we have a set of paraphyletic results as described above

The composite score for one alignment is calculated as weighted combination of subscores:

- median score of all species, excluding sites containing too many gaps.
- score for hg19
- equals 1 if both basic and paraphyletic feature appeared in array of sites, 0 – otherwise

for calculating total score we also used different ratio for each subscores:
ratio1, ratio2 >=0; ratio1+ratio2<=1
alignmentScore = ratio1*subScore1 + ratio2*subScore2 + (1-ratio1-ratio2)*subScore3

```
double alignmentScore = 0.0; // resulting score
int nScores =  pfSites.length;
double[] scores = new double [nScores];

// calculating 1st subscores (median score)
for (int i = 0; i < nscores; ++i) {
       scores[i] = calcPfScore(tfm, pfSites[i], ...);
}
sort(scores);
if (nscores == 1) alignmentScore += scores[0];
      else if (nscores%2 == 1) alignmentScore += scores[nscores/2];
      else alignmentScore += 0.5*(scores[nscores/2]+scores[nscores/2-1]);
alignemntScore *= ratio1;
```

```
// calculate 2nd subscore (human score)
for (int i = 0; i < nscores; ++i) {
	if (pfSpecies[i].equals( “hg19” )) {
		alignmentScore += ratio2 * calcPfScore(tfm, pfSites[i], ...);
		break;
	}
}

// calculate 3rd subscore (checks if basic and paraphyletic feature occur)
boolean flagBf = false;
boolean flagPf = false;
for (int i = 0; i < nscores; ++i) {
	if (pfSites[i].charAt(pos1) == bf1 &&  pfSites[i].charAt(pos2) == bf2) {
		flagBf = true;
	}
	if (pfSites[i].charAt(pos1) == pf1 &&  pfSites[i].charAt(pos2) == pf2) {
		flagPf = true;
	}
}
if (flagBf && flagPf) {
	alignmentScore += (1.0-ratio1-ratio2);
}
```

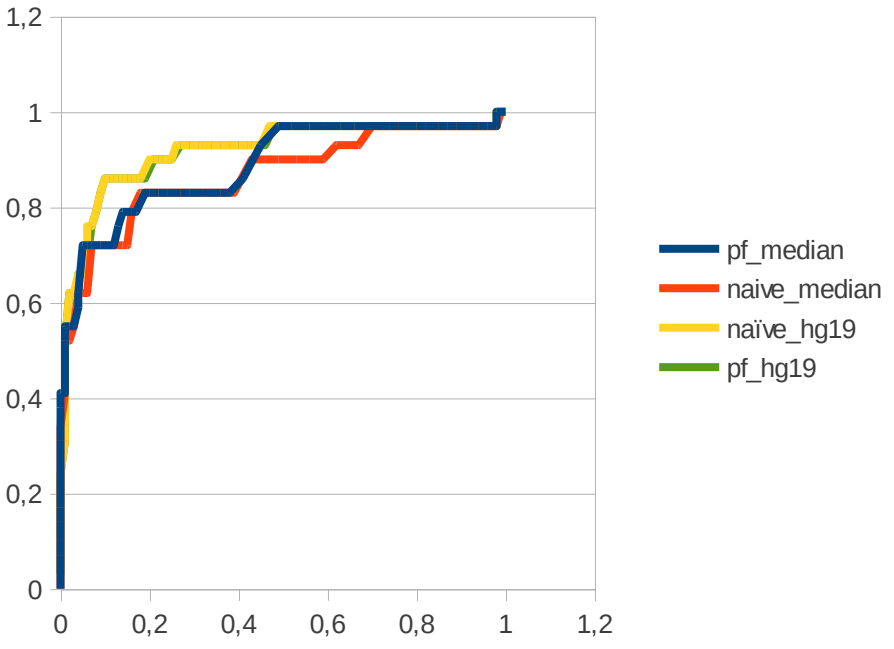
Building ROC-curves

We compared the naïve (simple binding site search) approach with the paraphyletic search on the basis of their ROC curves.

The set of alignments with paraphyletic features was selected as positive set. For each alignment we obtained the corresponding Transfac matrix that cuts a fragment of alignment. The total size of the positive set was 61 (for paraphyletic selection cutoff 0.7).

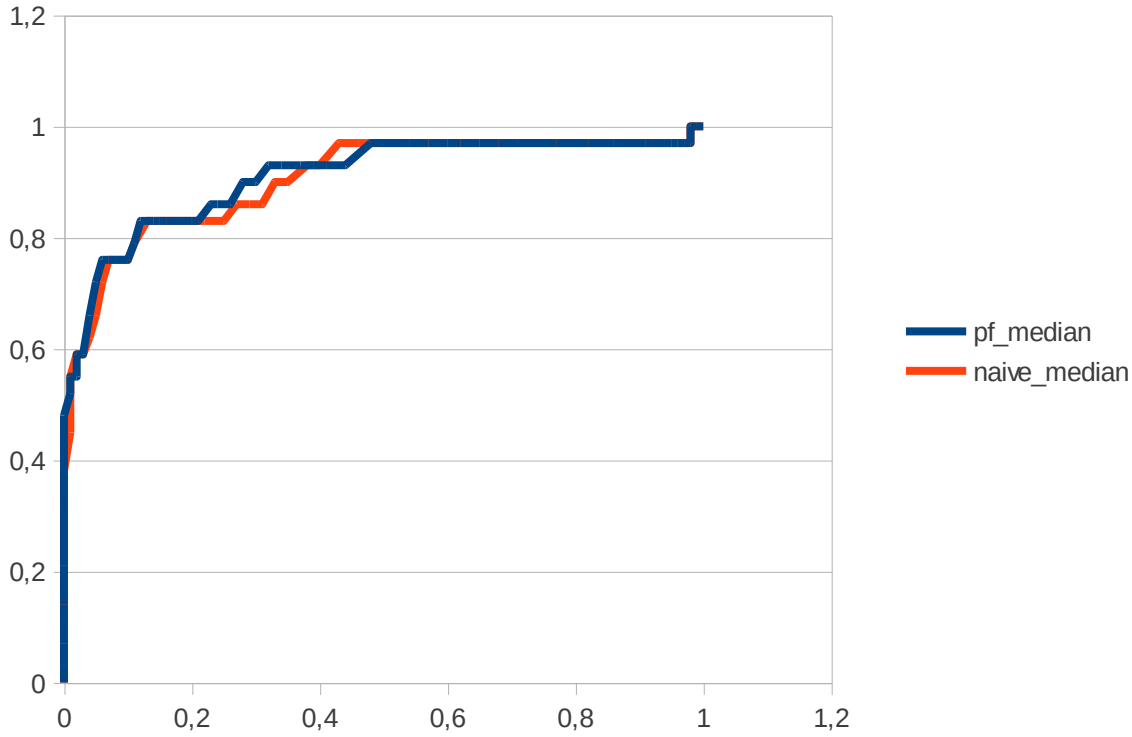
100 random alignments were generated for each alignment from the positive set by shuffling columns to compile a negative set of size 6100.ROC curves were then calculated for the naïve search and composite scores with different weight parameters.

There following results were obtained



- blue plot shows roc-curve for new paraphyletic score algorithm with ratio1=0.7 and ratio2=0.0 (median score for all species was taken)

total area below plot equals 0.888
- red plot shows roc-curve for naive or original score algorithm with ratio1=1.0 and ratio2=0.0
total area below plot equals 0.862
- green plot shows results for paraphyletic score algorithm with ratio1=0.0 and ratio2=1.0 (just human site score took into account)
total area below plot equals 0.913
- yellow plot shows results for naive score algorithm with ratio1=0.0 and ratio2=1.0
total area below plot equals 0.914



- blue plot shows results for paraphyletic score algorithm with ratio1=0.4 and ratio2=0.4
total area below plot equals 0.912
- red plot shows results for naive score algorithm with ratio1=0.4 and ratio2=0.4
total area below plot equals 0.905 the pf ratio=0.2 in both cases.

Discussion

We have developed and implemented a novel scoring scheme for binding site prediction using genomic alignments and paraphyletic features. Our results demonstrate that inclusion of the paraphyletic feature improves detection of binding sites with respect to the background model of shuffled alignments.
At first sight, singe-sequence searches (Fig. 3, yellow and green ROC curves) performed better than the methods including alignment information. This is not surprising as the true binding sites were known to be located in the human sequences and sequences in other species, specifically in alignments that did not

preferably align potential binding sites, were rather likely to incur a less optimal score. This can be observed in the comparison of the median scores (Fig 3, blue and red curves), which achieved lower ROC areas than the naïve predictions. However, comparison of the full paraphyletic scoring scheme to the naïve search (Fig 4), shows that it overall improved the prediction of binding sites. In addition, we expect that on a larger data set of known sites the effect of the phylogenetic method will become more evident. Here, only 61 binding sites with paraphyletic features were available.

Importantly, the paraphyletic scoring scheme accomplishes the most apparent improvement at weak binding sites, which differ more strongly from the profile consensus and are therefore associated with higher false positive rates in the prediction. For such sites, ROC curves of the paraphyletic method show a stronger increase than the naïve method (Fig. 3 and 4, blue curves, false positive rates between 0.2 and 0.8). This is an important finding, because detection of weak but funtional binding sites is considered to be a hard problem for binding site search algorithms. The comparison also indicates that the improved detection of weak binding sites is induced by the paraphyletic feature rather than the inclusion of sequences from other species.

Hence, the approach to search for evolutionary constraints based on paraphyletic evidence and consideration of such constraints in binding site search methods is highly promising to address the problem of detecting weak binding sites.

References

[1] Kel A, Voss N, Valeev T, Stegmaier P, Kel-Margoulis O, Wingender E. ExPlain: finding upstream drug targets in disease gene regulatory networks. SAR QSAR Environ Res. 2008;19(5-6):481-94.