A Parallel Architecture for Regulatory Motif Algorithm Assessment

Daniel Quest,[°⋆] **Kathryn Dempsey,**[⋆] **Mohammad Shafiullah,**[⋆] **Dhundy Bastola,**[⋆] **and Hesham Ali**[°⋆]
[⋆] College of Information Technology, University of Nebraska at Omaha
[°] Department of Pathology and Microbiology, University of Nebraska Medical Center
Omaha, NE 68182-0694 USA
E-mail: djquest@unmc.edu

Abstract

Computational discovery of cis-regulatory motifs has become one of the more challenging problems in bioinformatics. In recent years, over 150 methods have been proposed as solutions, however, it remains difficult to characterize the advantages and disadvantages of these approaches because of the wide variability of approaches and datasets. Although biologists desire a set of parameters and a program most appropriate for cis-regulatory discovery in their domain of interest, compiling such a list is a great computational challenge. First, a discovery pipeline for 150+ methods must be automated and then each dataset of interest must used to grade the methods. Automation is challenging because these programs are intended to be used over a small set of sites and consequently have many manual steps intended to help the user in fine-tuning the program to specific problems or organisms. If a program is fine-tuned to parameters other than those used in the original paper, it is not guaranteed to have the same sensitivity and specificity. Consequently, there are few methods that rank motif discovery tools. This paper proposes a parallel framework for the automation and evaluation of cis-regulatory motif discovery tools. This evaluation platform can both run and benchmark motif discovery tools over a wide range of parameters and is the first method to consider both multiple binding locations within a regulatory region and regulatory regions of orthologous genes. Because of the large amount of tests required, we implemented this platform on a computing cluster to increase performance.

1. Introduction

De novo cis-regulatory motif detection has evolved significantly over the past 20 years. Currently there exists over 150 different programs that discover cis-regulatory motifs in a set of sequences (you can find our partial list of these programs at `http://biobase.ist.unomaha.edu`). Each of these tools have distinct advantages and disadvantages but the primary differences are the scoring function and model used by the algorithm, the representation of a motif used by the algorithm, and the data presented to the algorithm. A historical review of these algorithms was written by Brazma *et al.* in 1998 [1]. More recently, Sandve *et al.* proposed an integrative framework for classifying motif discovery algorithms [5].

With the large amount of options available today for cis-regulatory motif discovery, it is increasingly important to be able to evaluate the performance of these tools over different datasets. Upon publication, most tools provide their own dataset and benchmark themselves against only a few other approaches (e.g. approaches that most closely resemble their own approach). Unfortunately, this does not indicate which motif discovery methods are most successful overall or which motif discovery programs perform well over specific datasets. In an environment of so many options, it is difficult for biologists to know which method will likely perform the best. This makes motif algorithm assessment increasingly important.

In 2004, Tompa *et al.* presented a benchmarking strategy to evaluate motif discovery tools by evaluating 13 tools on transcription factors found in yeast, flies, mice, and humans. While the study was seminal in its importance, it may not be realistic because it only contained 8 sites from *Saccharomyces cerevisiae*, 6 sites from *Drosophila melanogaster*, 12 sites from *Mus musculus*, and 26 sites from *Homo sapiens*. When one considers that RegulonDB contains 164 known binding sites for transcription factors in *Escherichia coli K12* it appears that many motifs are not annotated in the benchmarking dataset. Tompa attempted to address this problem by constructing synthetic versions of his datasets and evaluating them. However, his approach does not consider the combinatorial interactions that are known to exist in cis-regulatory modules. Also, many steps in his assessment were manual. Consequently, the benchmark is limited because it can not be run hundreds of times to fine-tune algorithm parameters, it can not easily adapt to account

978-1-4244-1694-3/08/$25.00

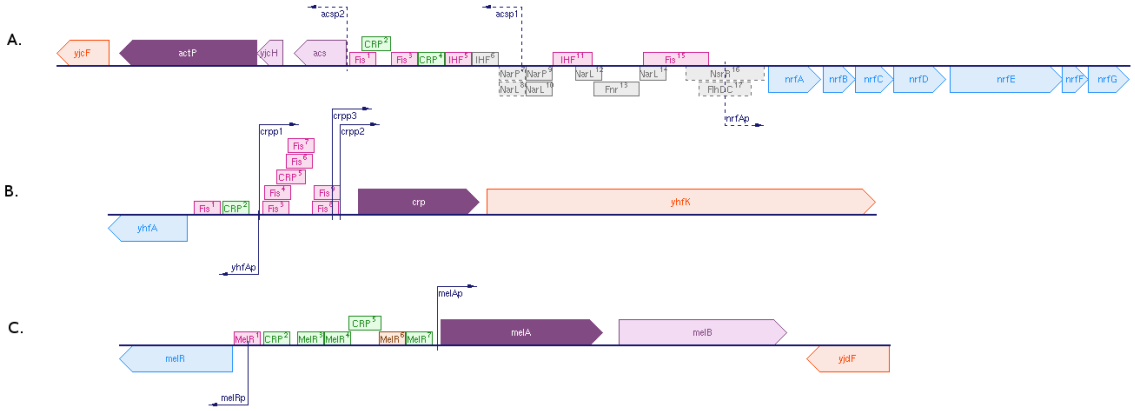

Figure 1. Three example regulatory modules taken from EcoCyc. Arrows represent coding sequences. Green boxes represent transcription factor binding positions for activators. Pink boxes represent binding positions for repressors. Brown boxes denote a site that has a dual function.

for newly discovered transcription factors (especially those factors that exist in the same upstream sequence), and its method of assessment can not be changed. Sandve *et al.* expanded Tompa's original paper to include a more robust set of machine learning benchmarks for assessment [4]. Although these additions are helpful and address many important concerns, the dataset used is the same as Tompa's and therefore carries the same inaccuracies. Hu *et al.* [2] assessed 5 tools in *E. coli* using RegulonDB [3]. Because RegulonDB contains so many more known transcription factor binding sites in relation to all the possible transcription factor binding sites when compared to the Transfac [7] data that was used to construct the Tompa test set, a more realistic representation of tool performance can be expected. Unfortunately, this study does not include Weeder, the tool that performed best in the Tompa assessment. The assessments presented here also do not consider methods that integrate phylogentic information in the algorithm despite the fact that phylogenetically related sequences are readily available for both of these datasets.

Motif prediction algorithms are constantly improving and assessments are generally made obsolete by new tools by the time they are published. Tool developers and the biological community would be able to more quickly understand and correct deficits in the current approaches if an automated method for evaluating performance over many transcription factor datasets were available. However, such a tool has not emerged because of the difficulty in automating motif detection tools and the processing power required to test thousands of runs over hundreds of cis-regulatory regions.

This paper proposes a method that automates motif prediction tools and performs benchmarks on these tools using a computing cluster. Our goal is to perform assessments on the genome level and consider every regulatory motif found in an organism to evaluate motif tool performance. We believe this framework has a substantial advantage over traditional methods because it allows individual practitioners and experimental biologists to perform benchmarks themselves. We feel that this approach can rapidly accelerate the understanding of the motif discovery problem and help to fine-tune the performance of current approaches on a wide variety of datasets.

2. Motivation

Better automation techniques and better evaluation platforms go hand in hand in their necessity for one another. Evaluation technology could be greatly improved if it could consider more important parameters of the problem. Discovery technology can be improved if programs benchmarking is fair and rapid. In this section, we motivate the need for our approach by outlining the difficulties associated with automating motif discovery tools and with the current benchmarking techniques.

2.1. Motivation for evaluation

Given a list of sets of co-expressed genes, the objective of *de novo* regulatory motif detection is to mark the binding positions of regulatory proteins upstream of all genes in the list. For example, given a set of genes $S_1 = g_1, g_2, g_3$

that are co-expressed under one set of environmental conditions and a second set of genes $S_2 = g_2, g_3, g_4$ that are co-expressed under another set of environmental conditions, our goal is to use the list $S_1, S_2, \ldots$ to label the transcription regulatory networks upstream of all genes in the genome computationally. These binding positions can then be used to help determine the combinatorial control on each of the genes in the genome and to diagnose the interworkings of the resulting cellular interaction networks.

Labeling all of the transcription factor binding positions computationally is a very ambitious goal because it is complicated by many smaller problems. First, multiple binding positions occur upstream of any given coding sequence. In Figure 1, we have an illustrative example of three regulatory regions from E. coli. CRP binds one or more times to each one of these regulatory regions. Multiple types of binding positions often serve to confuse algorithm objective functions that are designed to sort the binding sites from the 'background noise'. Second, the number and location of the binding positions relative to the transcription start site varies to control the binding rate of the regulatory proteins to the DNA and therefore the level of expression of the regulated gene. Third, the variety found in the motifs that serve as docking sites controls the frequency of binding. From a computational standpoint, this high variability makes detection of regulatory binding motifs extremely difficult because it is hard to discriminate between binding sites and 'background noise'. Regulatory proteins bind to DNA with different conformation profiles depending on the type of DNA-protein interaction. The differences in DNA-protein interaction profiles make finding an accurate universal model of binding sites problematic. In addition, we do not know for sure if the genes in our set are co-expressed because they have a common regulatory interaction and are therefore co-regulated, or if they are co-expressed because of similar responses to the environmental conditions. Lastly, we do not know the context of the DNA-protein binding within the regulatory mechanism. In some instances, a protein may bind in combination with many other factors to activate transcription. In other cases, it may either repress transcription or activate transcription of a gene on the alternative strand. For many of these problems, techniques have been developed that solve many of the critical issues in isolation. A key limitation in moving motif prediction algorithms from a single target to simultaneously consider multiple targets across the genome is the lack of a robust benchmarking platform that compares methods and makes context specific tool recommendations.

2.2. Motivation for automation

The traditional goal of a cis-regulatory motif discovery program is to mark the positions (start position, end position, regulator profile, and strand) on the DNA where regulatory proteins bind. These programs accept a set of genes $S_i = g_1, g_2, \ldots, g_k$ that we believe to be co-regulated, user defined parameters P, and the set of sequences $U = u_1, u_2 \ldots$ upstream of those genes. The program outputs a mapping of regulatory motifs found on the sequences to positions on U. In the process of marking the binding positions, the motif discovery program runs a series of steps to distinguish putative binding positions from all possible positions in the input set. Performance of a method is not just the method itself but the entire pipeline of steps performed by the user.
Performance variations can be caused by the following:

- Pre-Processing: Users may select a different series of steps to produce the upstream sequences, background sequences, phylogentic upstream sequences, and repeat masked sequences.
- Manual Steps: Users may be required to manually extract sequence information or provide input information.
- Paramater Tuning: Experts set parameters based on information in Transfac, protien structure data, binding assays and other biological information that allows them to narrow down their search.
- Advancements: A tool found in one step along the motif detection pipeline may undergo revision or another method may substantially improve in performance and replace a step.

Many would argue that these attributes increase the value of motif detection software because users can fine-tune programs for their needs. We feel that there is value in explicitly defining each step in the pipeline to be used because the performance can then be benchmarked and we can improve prediction quality based on training data and formal methods from the machine learning community.

3. Problem Description

Given a set of data $D = G_1, G_2, \ldots, K_1, K_2, \ldots$, a set of pre-processing functions, R, and a set of motif detection algorithm pipelines, $M = M_1, M_2, \ldots, M_i$, we wish to construct tests, $T = T_1, T_2, \ldots$, that we will use to evaluate the performance of M_i on T_l. This run will create a marking of predicted sites for M_i on T_l. We will then generate the marking of M_i over all elements in T_l. The data contains genome sequences in Genbank format, G_j, and expression profiles. It is possible to include chromatin imunosurpression on arrays, transcription factor binding assays, or data found in databases such as Transfac [7], however we

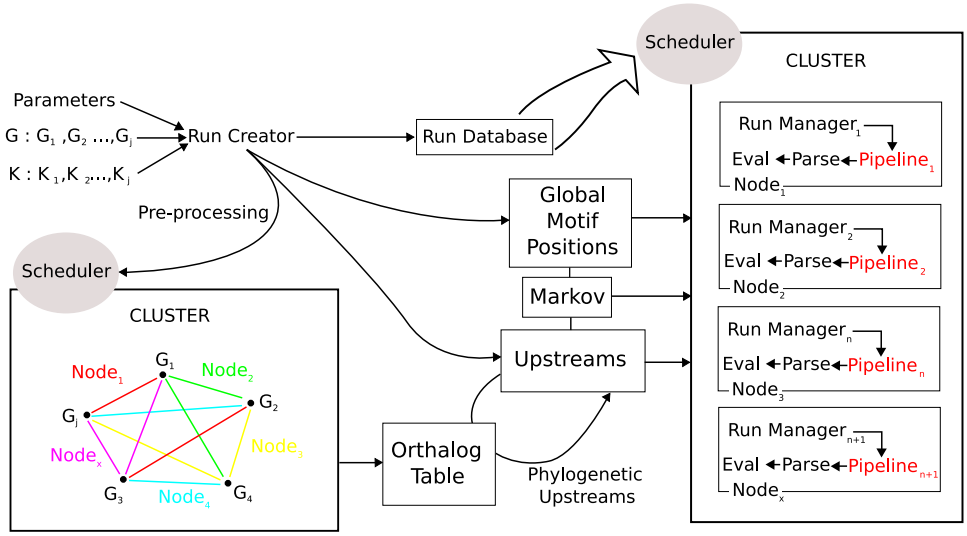

Figure 2. An overview of the MTAP architecture. Parallel components and scheduling exists both in computing orthologs between genomes and in running motif discovery pipelines. Paramaters for what pipelines are to be run, genomes, and known transcription factor binding positions enter the platform and the run creator creates a set of unique runs in the run database. The run creator also creates the upstream files needed. The run database is queried to spawn independent instances of the run manager that can be run on any node of the cluster. Run manager instances run the pipeline, reformat the results and evaluate sensitivity and specificity for the run.

instead choose to use this data to validate our *de novo* strategy. So, we reduce this data into a validation dataset, K. K_k contains features that label subsequences in G_i based on a binding position of protien k. The set of pre-processing functions are utility functions that allow algorithms to collect more relevant data from public resources or mask biologically relevant data. Our goal is to provide the user with the performance in terms of runtime and a sensitivity/specificity trade-offs so that they can evaluate the best approach for their problem.

4. Parallel Architecture

In this section, we will discuss the design of our motif tool assessment platform (MTAP). The software was designed to provide for the requirements of a wide variety of motif detection tools while simultaneously running over many architectures and operating systems in a clustered environment. We constructed a run manager for executing motif detection pipelines that considers many platforms and a run creator that runs on a small subcluster of nodes. The run creator generates unique tuples that completely describe T_l. A unique tuple, Tup, is of the form $[D, R, M_i]$. The run creator then generates T_l and K_l that we will use to score M_i. The run manager then executes M_i on any computer in the cluster. We chose this setup because in our prototype tests, we found running motif prediction algorithms to be the rate-limiting step. We therefore wanted a centralized set of powerful homogeneous nodes to compute the pre-processing and generate the runs because many of these operations are not independent. For running M_i, we opted to manage the program dependencies across many architectures and to separate runs as independent units that can be run at will.

4.1. Pre-processing

The goal of pre-processing is to generate all tests T_l that we will use to benchmark motif discovery pipelines. Generating T_l requires a different set of operations depending on the input required by motif prediction tools in T_l. A test consists of a set of sequences each of a predetermined length n. Windows are defined by two methods 'Completely-realistic' (the sequence n bases upstream of the gene) and 'Semi-realistic' (the sequence n bases around the motif.

Given n and a data generation method, we generate a test T_l by selecting a regulator K_k and generating an upstream sequence U_q for every element K_j such that the regulator name of $K_j = K_k$. All tests in the genome are constructed given initial inputs $G_j, G_{j+1}, G_{j+2}, \ldots$ and K. MTAP then constructs a list of all proteins in each genome

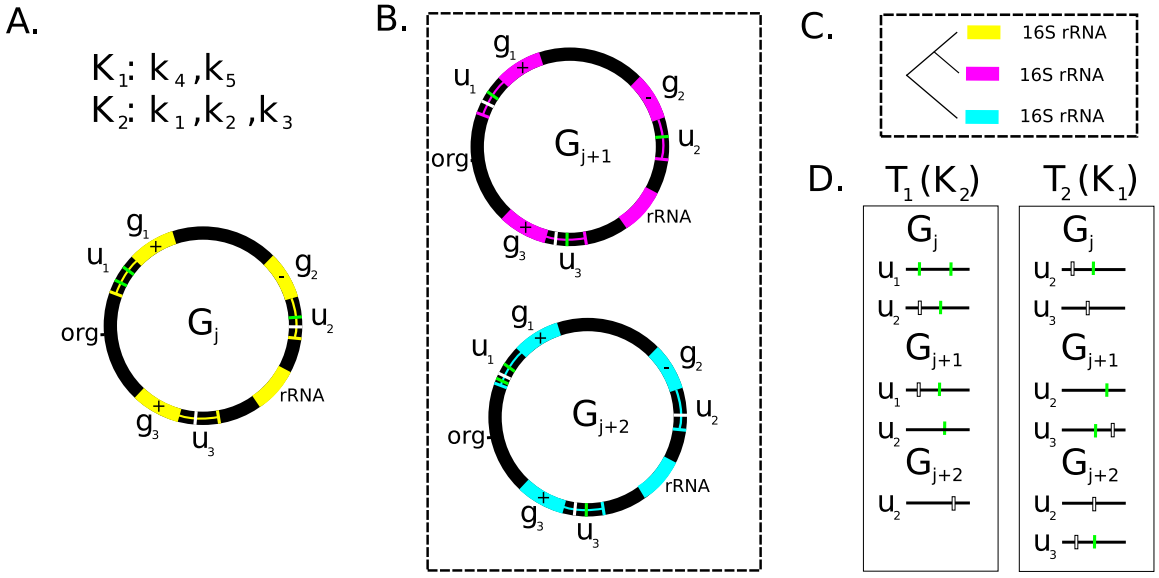

Figure 3. An overview of the pre-processing steps taken in MTAP: A. The source genome G_j with genes g_1, g_2, g_3 with known binding sites k_1, k_2, k_3, k_4, k_5 (found clockwise around the genome from the origin); K_1 and K_2 are the known regulatory proteins that bind to the transcription factors B. Two additional genomes, G_{j+1}, and G_{j+2} that are phylogentically closely related to G_j. G_{j+1}, and G_{j+2} also have binding sites C. The background phylogenetic tree constructed via extracting 16S rRNA D. Two tests T_1 and T_2 corresponding to known positions from K_1 and K_2

$G_j, G_{j+1}, G_{j+2}, \ldots$ and constructs a sequence database Q_j for each genome. For each protein database, we construct an orthalog mapping using RSD between Q_j and Q_{j+1}. Then, when given a test for G_j containing several upstream sequences (for example u_1, u_2 in Figure 3 D.) we compute the downstream gene (or first member of downstream operon) and lookup the protein product in Q_j. MTAP then uses the translation table generated by RSD to find orthologs for all proteins downstream of the elements in T_j. Lastly, MTAP computes upstreams of each of the proteins found in G_{j+1}. MTAP continues this procedure for all genomes $G_j, G_{j+1}, G_{j+2}, \ldots$ to construct the test T_j. This procedure is then repeated to construct $T_1, T_2, \ldots$ An example result is found in Figure 3 D.

Several important points should be made about phylogentic footprinting. First, just because a set of regulatory binding sites exists in G_j it does not imply that this set of regulatory binding positions exist in G_{j+1} in the same positions. In our example in Figure 3, T_1 does not contain a binding position for K_2 in G_{j+2}. It is possible that our criteria to classify orthalogs is too stringent to find an actual ortholog in G_{j+1}. Notice that in our example T_1 contains no upstream corresponding to g_1 in G_{j+2}. In our example, this occurred because the homology threshold criteria excluded g_2 from G_{j+2}. These are problems that will exist in any method that uses phylogeny, however at the present time we have been unable to explore these problems because of the challenges involved in integrating all of these steps and evaluating motif prediction pipelines. We believe an advantage of our method is that it integrates all of these steps together into one platform and allows us to consider for the first time the impact of including different species at different time points in evolution.

Many programs have different background sequence requirements. Phylogenetic programs require a background phylogenetic tree constructed from extracting 16S rRNA from each genome in the study. Some programs require pre-processing steps to calculate an HMM or GC content of the test sequences. Other programs require a background probability distribution of all upstream sequences in G_j. We compute each of these requirements for the programs in our pre-processing stage.

However, the most challenging problem in assessing regulatory motif prediction tools is that it is not guaranteed that all binding sites in the genome will be used to assess the tool. It is often desirable to exclude unknown sites from the tests. To do this we construct a Markov chain (of order m) and use this chain to construct a synthetic test (one for every length m provided by the user). The idea is to use the Markov chain to scramble the upstream sequences in the test and then re-insert the known motifs back into the sequences at the same positions found in the source genome.

We also keep a set of similarly constructed tests without the motifs inserted to check false positive rates. Orthologus upstream sequences need not be scrambled if they are not scored. These synthetic sequences serve to make a more fair test in those cases where very few of the known motifs are marked.

4.2. Constructing a pipeline

The objective of a pipeline is to mark T with regulatory motif predictions. We feel that the most flexible and powerful way to construct pipelines is through a scripting language interface. Scripting languages are flexible, powerful, widely used and easily understood. MTAP provides a library of utilities that supports all of the pre-processing, post-processing and motif discovery algorithms currently installed. Writing a new pipeline only requires instantiation of a new class inherited from the MTAP library and a series of system calls from within that class to control execution of pipeline steps. The MTAP scheduler automates mapping of resources, dependencies and sequences to steps within the pipeline.

4.3. Running motif pipelines

The MTAP architecture does not enforce any language input parameters or format on M_i. This flexibility implies that M_i must write intermediate results as is passes through each stage in execution. It is important when running pipelines not to create a bottleneck by storing intermediate results in a shared resource, such as a DBMS running on the master node. Our solution ensures that once preprocessing is complete, M_i can execute in parallel without inter-process dependencies.

The run creator first creates an object oriented database, DB, that stores the program dependencies and requirements of each pipeline. The characteristics of the run are then loaded into the DB. This allows us to trim the execution hierarchy tree. For example, a run can contain only tests generated using using the completely-realistic method. The run creator then generates a run list Q for every pipeline Q_i to be executed. The pre-processing manager then creates a directory structure on a network file structure that is accessible across the cluster. This directory structure tree contains all elements described in the DB. Once the directory structure is generated, the pre-processing manager populates each leaf in the tree with a set of tests for all motifs in K. The pre-processing manager then collects the file dependencies for each pipeline run in Q and constructs the necessary files (e.g. it makes a fasta file of all sequences upstream to a gene in G for background training, it constructs a rRNA file for the phylogenetic distances, etc.). When all dependencies for Q_i are met, Q_i is thrown to the PBS server to handle the runs FIFO. The PBS server then starts an instance of the run manager that will run Q_i and the post-processing processes to evaluate the run. At this time, the scheduler presents substantial overhead. Optimizing scheduling algorithms based on runtime properties of individual motif discovery pipelines is a interesting research problem beyond the scope of this paper.

Once the run manager is spawned on a node, each step of the pipeline is executed and intermediate results are written to a unique position on the file system. Upon completion of the run, results are combined back into the shared file system and remaining post-processing is handled by the slave node.

4.4. Post-processing

The purpose of post-processing is to reformat the output from the motif tool to a standard format for visualization and evaluation. Once the raw predictions for T_l are generated, they are re-mapped to global positions on G_x and T_l is scored. Because there are more than 100 different file formats (one for each motif prediction software) and we can not force authors to adopt a universal format, we employ a three tiered parsing strategy. First, for those tools where there already exists a parser or a converter to gff format, we convert the tool's output to gff and then parse it with the gff parser present in our Java scoring framework. For those tools that do not currently have a parser, but do have a simple regular format, we employ martel from the biopython package to generate a regular grammar for parsing the file and map the output to gff format. For those tools that require assembly of information from multiple parts of the file, we employ a recursive decent parser implemented in our java framework to parse the tool. The need for a standard interchange format for presenting prediction results can not be stressed enough. However in the absence of such a format, we feel the best chance for evaluating motif prediction tools is to provide as many opportunities in as many languages as possible to convert tool output into a universal format (in our case gff). Once we have a mapping from the predictions, we score the predicted motif predictions versus the known motif predictions for T_l and dump the gff file for T_l for visualization (using GBrowse for example).

5. Results

In this section we will provide illustrative examples of how our benchmarking technology can be used to evaluate several important parameters in cis-regulatory motif discovery. To construct a series of tests for evaluation, we extracted 2247 known transcription factor binding positions from RegulonDB [3] corresponding to known positions from *Escherichia coli K12*, and 680 known motifs

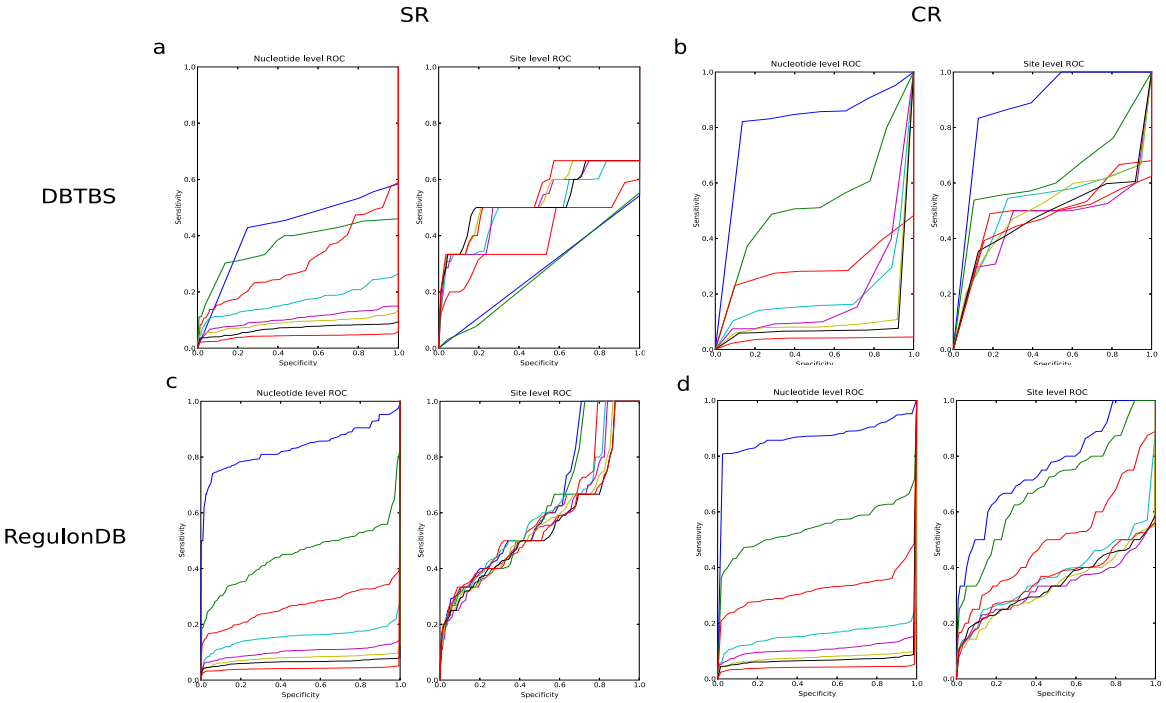

Figure 4. ROC curves for Ann-Spec (20 (blue), 50 (green), 100 (red), 200 (cyan), 300 (magenta), 400 (yellow), 500 (black), and 800 (lower red) basepairs upstream of the gene

from DBTBS corresponding to positions from *Bacillus subtilis.* Then, we extracted 522 positions from Eukaryotes from the dataset developed by Tompa *et al.* [6].

To justify the need for clustered computing, we ran MTAP on one node of our cluster and then over 21 nodes. Running MTAP over RegulonDB on a single AMD Opteron with 4 gigabytes or ram required 3 weeks and 5 hours. The same case study ran over the cluster finished in 7hours 37 minutes using the AMD Opteron as the master node and 20 Pentium D machines as the slave nodes.

Preliminary results (not shown) indicated that the size of T_l plays a roll in motif detection performance. In this section we wanted to determine if the length of the upstream sequences or the number of upstream sequences caused the performance discrepancy. For each motif database, we ran our assessment procedure and generated ROC graphs for lengths 20bp, 50bp, 100bp, 200bp, 300bp, 400bp, 500bp, and 800bp upstream of the gene for DBTBS and RegulonDB. We generated both completely-realistic (cr) and semi-realistic (sr) data. This procedure was run for ANN-Spec, Weeder, ELPH and Glam, however due to space considerations, we have provided the results for ANN-Spec in Figure 4 (which is illustrative of these results). The distribution of performance over the curves as the length of the upstream regions increases is particularly interesting. In the case of 20 bases upstream, the program is forced onto the known motif position, resulting in satisfactory performance. As the window increases, the corresponding performance (AUC) rapidly degrades such that at 300-400bp, very little confidence remains in the predictions at the nucleotide level. The distribution of the motif relative to the transcription start site is important as cr data has a slight performance advantage over sr data. In the site level distribution curve for sr on RegulonDB it appears that site identification is not impacted significantly by window length. However, this performance curve does not remain consistent when the transcription start site in the cr data is considered. This implies that sites are most likely predicted almost by random chance without the help of the transcription start site. The clustering of performance curves from longer upstream files at the site level in Figure 4d gives additional evidence towards this hypothesis. ANN-Spec appears to be selecting a set of regions with high information content and then randomly selecting from those putative sites. Consequently, performance appears random because ANN-Spec can not reliably distinguish statistically significant sites from true sites. Lengthening the upstream sequence to 800bp significantly degrades nucleotide performance but without site level performance being impaired. It is most likely that for for this test in RegulonDB, ANN-Spec's scoring algorithm

(and not upstream sequence length) is the key contributor to poor performance. However, at the nucleotide level a large value of n is a key factor in performance degradation. DBTBS performance benchmarks are much more difficult to interpret because of the sparse site annotation density over DBTBS. This indicates that dense motif annotation databases are a key in building accurate motif discovery algorithms. However, using RegulonDB exclusively for algorithm testing has a significant disadvantage in that algorithms will be over-fit to problems specific to *E. coli.*

The result presented in this section is one of many results generated by our pipeline. These results will be discussed more completely in future work.

6. Discussion

In this paper we have presented a method that has several advantages over previous attempts in cis-regulatory motif discovery. First, our method is completely automated. This allows us to evaluate motif prediction tools over many parameters in a way not previously possible. Second, our method considers multiple binding sites that may occur in other regulatory regions. Third, our method can score motif prediction algorithms on a genome scale for the first time. In the past, training data has been very hard to obtain for genome based assessments. Now, with the advent of ChIP-chip data, such training data can be more rapidly obtained. We feel that this contribution will become more important in coming years. MTAP does not allow manual fine-tuning of motif prediction tools to specific problems. We feel this is an asset because authors must provide a robust and flexible methods that can adapt to different scenarios over many types of transcription factor binding profiles. Our method provides a way of assessing what methods have the highest sensitivity and specificity with the least amount of data collection and execution time. This presents a great opportunity because both data collection and tool evaluation are extremely laborious. MTAP is currently only available in a clustered computing environments because of the large scale of parameters that are explored in evaluating motif pipelines. We presented many practical techniques for large scale integration of hundreds of bioinformatics standalone applications and deployment in a clustered computing environment. More importantly, because of the large amount of components installed in our platform, we have provided a model for rapid construction and deployment of motif discovery pipelines. These pipelines can be deployed and executed in heterogeneous clusters with different operating systems and different underlying architectures. We believe that the advent of high density training data combined with MTAP can greatly increase motif discovery pipeline performance. Our goal is to use clustered computing environments to explore the parameter space for motif discovery pipelines with high sensitivity and specificity. This will enable methods to emerge that correctly solve this very challenging problem. These methods will also most likely require fast parallel algorithms to survey genomes for regulatory binding motifs.

7 Acknowledgements

We would like to thank the authors of the motif tools in our pipeline for providing the tools open source and helpful suggestions. We would also like to thank Laura {A.|R.} Quest for help with the manuscript. This research project was made possible by the NSF grant number EPS-0091900 and the NIH grant number P20 RR16469 from the INBRE Program of the National Center for Research Resources.

References

[1] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *J Comput Biol*, 5(2):279–305, 1998.

[2] J. Hu, B. Li, and D. Kihara. Limitations and potentials of current motif discovery algorithms. *Nucleic Acids Research*, 33(15):4899–4913, 2005.

[3] H. Salgado, S. Gama-Castro, A. Martez-Antonio, E. Dz-Peredo, F. Schez-Solano, M. Peralta-Gil, D. Garcia-Alonso, V. Jimez-Jacinto, A. Santos-Zavaleta, C. Bonavides-Martez, and J. Collado-Vides. Regulondb (version 4.0): transcriptional regulation, operon organization and growth conditions in escherichia coli k-12. *Nucleic Acids Res*, 32(Database issue), January 2004.

[4] G. K. Sandve, O. Abul, V. Walseng, and F. Drablos. Improved benchmarks for computational motif discovery. *BMC Bioinformatics*, 8:193+, June 2007.

[5] G. K. Sandve and F. Drabls. A survey of motif discovery methods in an integrated framework. *Biology Direct*, 1(11), 2006.

[6] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. D. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, J. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Rnier, N. Simonis, S. Sinha, G. Thijs, J. v. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23(1):137–144, January 2005.

[7] E. Wingender, P. Dietze, H. Karas, and R. Knppel. Transfac: a database on transcription factors and their dna binding sites. *Nucleic Acids Res*, 24(1):238–241, January 1996.