BIOINFORMATICS

Vol. 00 no. 00 2005
Pages 1–7

Neighbourhood Thresholding for Projection-Based Motif Discovery

James King, Warren Cheung and Holger H. Hoos
University of British Columbia
Department of Computer Science
{king,wcheung,hoos}@cs.ubc.ca

ABSTRACT
The PROJECTION algorithm by Buhler and Tompa is one of the best existing methods for solving hard motif discovery problems for monad motifs of fixed length l. In this paper we introduce the AGGREGATION algorithm, which like PROJECTION projects all l-mers from the given input sequences into buckets, but uses a different scheme for selecting buckets for subsequent refinement search. This new neighbourhood-based thresholding scheme allows AGGREGATION to discover motifs in biased background sequences that cannot be found by PROJECTION. In other cases, AGGREGATION finds motifs of the same quality as PROJECTION substantially more efficiently.

1 INTRODUCTION

Many important cellular processes involve the recognition of short, conserved subsequences in genomic DNA. The problem of discovering such motifs in a given set of DNA sequences is of considerable interest, and many computational approaches for motif discovery have been developed. In this work, we consider the following well-known combinatorial formulation of this problem due to Pevzner and Sze [5]:

> **The Planted (l, d)-Motif Problem:** Let M be a fixed but unknown nucleotide sequence (the motif consensus) of length l. Suppose that M occurs once in each of t background sequences of common length n, but that each occurrence of M is corrupted by exactly d point substitutions in positions chosen independently at random. Given the t sequences, recover the motif occurrences and the consensus M.

When it was first introduced, the $(15, 4)$ problem with $t = 20$, $l = 600$ posed a substantial challenge to the best motif discovery algorithms at the time. Since then, several algorithms have been developed to solve this ‘Motif Challenge Problem’. These include Pevzner and Sze’s WINNOWER and SP-STAR algorithms [5], as well as the PROJECTION algorithm by Buhler and Tompa [1, 2], which forms the basis for the work presented here, and the more recent PatternBranching and ProfileBranching algorithms by Price *et al.*[6]. It should be noted that, like earlier motif finding methods, these algorithms are not restricted to the (l, d)-motif problem and have been applied to biological data very successfully.

In this paper, we introduce a new algorithm, dubbed AGGREGATION, that performs significantly better than PROJECTION on the (l, d)-motif problem, particularly when the base distribution in the background sequences is non-uniform. AGGREGATION hashes the l-mers (subsequences of length l) from the given sequences into buckets exactly as PROJECTION does; the key difference is that AGGREGATION uses a different approach for selecting the buckets on which a refinement search for the motif is performed. More precisely, rather than selecting the buckets to be refined based solely on the number of l-mers hashed into them, our new algorithm refines each bucket *whose neighbourhood* contains a sufficiently large number of l-mers. This reduces the number of buckets refined using EM, which more than compensates for the additional complexity of the selection scheme.

We show experimentally that AGGREGATION performs much better than PROJECTION on problem instances in which the base distribution is biased, and is therefore able to solve a wider range of problem instances than PROJECTION can. AGGREGATION also performs significantly faster than PROJECTION while achieving slightly better solution quality, even when the base distribution is uniform. This improvement is maintained for variations of the challenge problem. We show that the improvements are not restricted to synthetic challenge problems, but can also be observed when AGGREGATION is applied to the sets of biological sequences used by Buhler and Tompa [2]. We also show that PatternBranching, which is generally faster than AGGREGATION but does not achieve quite the same solution quality, is actually slower than AGGREGATION when the distribution of the background bases deviates significantly from uniform.

The remainder of the paper is structured as follows. In Section 2, we describe the AGGREGATION algorithm in detail and discuss its properties. Empirical results are presented in Section 3. Section 4 contains conclusions and points out some directions for future work.

2 FROM PROJECTION TO AGGREGATION

The PROJECTION algorithm by Buhler and Tompa [1, 2] works as follows. In the first phase (the projection phase) every l-mer from the given set of sequences is hashed into a bucket uniquely determined by k of its l positions. The k positions used for hashing are determined uniformly at random at the start of the projection phase. Two l-mers are hashed into the same bucket if, and only if, they have the same bases in those k positions. l-mers are in this way projected onto k-mers; the latter represent the fingerprints associated with each bucket. In the second phase (the refinement phase) each bucket containing more l-mers than the given threshold is refined using the Expectation Maximisation (EM) algorithm [4], where the threshold depends on input parameters l, d, n, and t. (For further details of the PROJECTION algorithm, see [2].)

The key idea underlying this procedure is that background l-mers are essentially random noise that will be distributed roughly evenly between the buckets. The mutated occurrences of the motif will tend to accumulate around the same bucket as the bucket in which the motif consensus would be hashed (the *planted bucket*) as they will often not be mutated in any of the k positions selected for projection. PROJECTION is likely to find the motif consensus if it refines the planted bucket. It also sometimes finds the motif consensus by refining a bucket close to the planted bucket. By running the above 2-phase procedure for a sufficient number of iterations, PROJECTION will refine the planted bucket at least once with high probability.

Typically, more than half of PROJECTION's running time is taken up by the EM algorithm used in refinement. For this reason, one approach for improving the motif discovery algorithm is reducing the number of buckets to which this refinement is applied. However, care needs to be taken to ensure that the probability of finding the correct motif is not decreased.

Our new AGGREGATION algorithm is based on the following key observation: when hashing the l-mers into buckets as in PROJECTION, many planted occurrences of the motif consensus will not end up in the planted bucket B_P (unless d is very small compared to l), but rather in a bucket whose fingerprint is close to that of B_P. AGGREGATION therefore refines any bucket *whose neighbourhood* contains more l-mers than some threshold. This threshold is chosen to give a high probability of refining the planted bucket while keeping the probability of refining other buckets low (see Figure 1, which will be explained in more detail shortly). We define the neighbourhood of a bucket B as the sum of all buckets whose fingerprints are at Hamming distance exactly 1 from the fingerprint of B. This neighbourhood could be extended beyond Hamming distance 1, leading to a further increase in the probability of a planted occurrence falling into the neighbourhood of the planted bucket, but this would make our threshold tests prohibitively expensive, and would also increase the sensitivity of the algorithm to variations in the background distribution.

Another improvement used when solving synthetic motif discovery instances is a more sensitive scoring mechanism. PROJECTION uses σ scores, a count of number of sequences in which the motif has instances with at most d mutations, to rate the motifs it recovers when solving synthetic problems. However, in cases where many motifs are found, this simple scoring method cannot distinguish spurious motifs from the true motif, assigning maximal σ scores to both. AGGREGATION also uses σ scores to rate the motifs it finds, but when two motifs have the same σ score, it uses the likelihood ratio scores computed during refinement to distinguish between spurious motifs and the planted motif. The likelihood ratio of a motif is essentially a measure of how different a motif is from the background distribution (see [2] for details). Spurious motifs will generally be more similar to the background distribution (*i.e.* will have lower likelihood ratios) than the planted motif, so this tie breaking works very well in practice.

An analysis of the expected probabilities of refining the planted bucket and any other bucket, respectively, provides a good basis for choosing good threshold parameters for AGGREGATION as well as for comparing AGGREGATION with PROJECTION. These probabilities can be approximated using the simplifying assumption that background l-mers are independently distributed. From these values, the expected number of buckets refined and the number of iterations required to achieve a 95% probability of refining the planted bucket can be determined; Figure 1 shows these measures for AGGREGATION as they vary with the refinement threshold. Not surprisingly, when choosing a good threshold we face a tradeoff — a threshold that is too low will send too many spurious buckets to refinement, whereas a threshold that is too high will cause the algorithm to require too many iterations before the planted bucket is refined.

The same analysis can be carried out for PROJECTION; Figure 2 shows for both algorithms the expected number of iterations needed to reach 95% probability that the planted bucket is refined *vs.* the total number of buckets refined as the refinement threshold is varied. As can be seen clearly from the figure, AGGREGATION offers a much finer level of control over this tradeoff than PROJECTION.

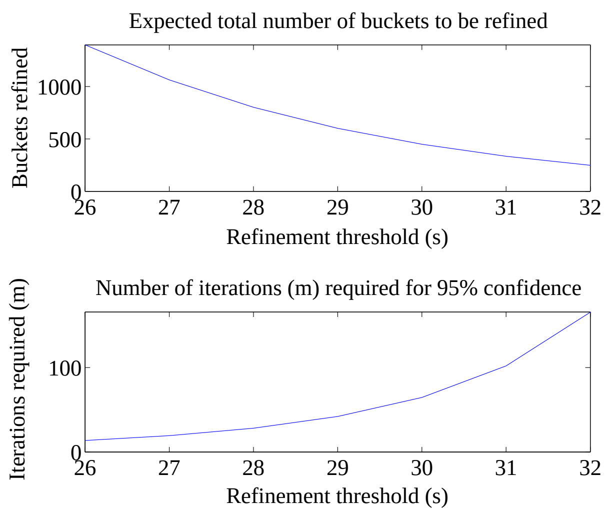


Fig. 1. Expected total number of buckets refined and iterations required by AGGREGATION to reach 95% confidence for varying refinement threshold. A good threshold keeps both of these values low; in this example, 28, 29, and 30 would make good thresholds. These curves are based on approximations of tail probabilities of the planted bucket (and other buckets) containing enough l-mers to be sent to refinement. These values are for $l = 15$, $d = 4$, $n = 600$, $t = 20$.

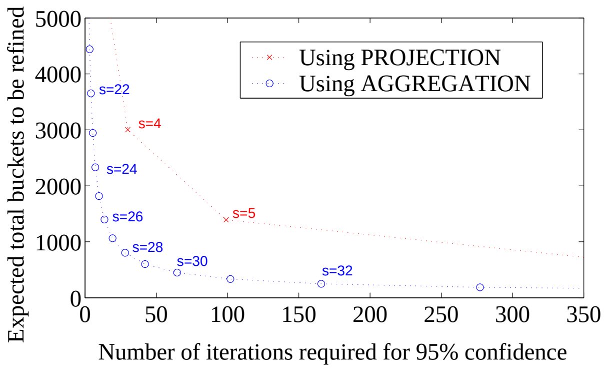


Fig. 2. Theoretical comparison of PROJECTION and AGGREGATION with varying refinement thresholds. These values are for $l = 15$, $d = 4$, $k = 7$, $n = 600$, $t = 20$.

Furthermore, when running both algorithms for the same number of iterations, AGGREGATION can achieve 95% confidence after refining less than half as many buckets as PROJECTION.

Because of the simplifying assumptions underlying our analysis and the fact that AGGREGATION incurs more overhead when selecting the buckets to be refined in each iteration, these analytical results, while encouraging, are by no means any proof that AGGREGATION is more efficient than PROJECTION. However, as we will see in the following section, experimental results confirm the performance advantage of AGGREGATION over PROJECTION.

3 EXPERIMENTAL RESULTS

To evaluate the performance of AGGREGATION *vs.* PROJECTION on synthetic instances of the (l, d)-motif problem, we use two measures: the performance coefficient and the expected solution time. The performance coefficient is a measure of how accurately the planted motif is recovered. It is defined as $|K \cap P|/|K \cup P|$, where K is the set of $l \cdot t$ base positions in the t planted motif instances, and P is the set of $l \cdot t$ base positions in the motif instances predicted by the algorithm [5]. The performance coefficient therefore quantifies the amount of overlap between the predicted motif and the actual motif. Expected solution time captures the time required by the algorithm to recover a solution to the given problem instance, *i.e.* an l-mer of which instances with at most d mutations exist in all t given sequences; it can be estimated from the empirical success probability of the algorithm, which is calculated as the frequency of iterations in which a solution was found and is measured over a large number of iterations. Note that a solution can be different from the actual planted motif, as long as it is as well preserved as the planted motif. This notion of solution has been previously used in the literature [3], as the planted motif cannot be discerned from a spurious motif with any certainty. For the experiments on biological data, we directly compare the solutions found by AGGREGATION to the results obtained by Buhler and Tompa [2] on the same data.

All empirical results were obtaind on a PC with dual 2.0 GHz Pentium Xeon processors with 512 KB cache (only one these processors was used for the experiments) and 1 GB RAM, running Redhat Linux 9 unless otherwise stated. AGGREGATION was implemented by modifying the source code for PROJECTION obtained from Jeremy Buhler as described in Section 2.

3.1 Results on Synthetic Challenge Problems

We compared the performance of AGGREGATION and PROJECTION on randomly generated instances of the (l, d)-motif problem that have been identified as particularly challenging in previous work [2]. For PROJECTION, we used the same parameters ($k = 7, s = 4$) as Buhler and Tompa [2], and the parameters for AGGREGATION were set to ($k = 7, s = 28$). To ensure that both algorithms were given comparable amounts of CPU time, the number of iterations, m, for AGGREGATION was set to twice the value of m used by PROJECTION; this reflects the fact that AGGREGATION performs substantially fewer refinements per iteration. (For actual runtimes, see Table 1.)

For each of the (l, d) combinations, 100 randomly generated sets of input sequences ($t = 20, n = 600$) were generated (as in [2]), and PROJECTION and AGGREGATION were both run once on each of these sequences. We measured mean performance and also report 95% confidence intervals over the instances from the respective test sets.

As seen in Tables 2 and 3, AGGREGATION reaches the same or better performance coefficients than PROJECTION while requiring less than half the expected solution time. We also assessed the quality of the AGGREGATION thresholding process by comparing the number of refinements which yielded the planted motif to the total number of refinements. The data shown in Table 4 indicates that the primary performance gain in AGGREGATION

Table 1. Runtime for PROJECTION *vs.* AGGREGATION on synthetic challenge problems, measured in CPU seconds. The data shown are the mean of the runtimes over m iterations and the respective 95% confidence interval.

l	d	PROJ. m	AGGR. m	PROJECTION Runtime	AGGREGATION Runtime
9	2	1483	2966	1896 ± 15	1730 ± 50
11	3	2443	4886	3223 ± 18	2800 ± 80
13	4	4178	8356	5550 ± 20	4570 ± 120
14	4	647	1294	704 ± 2	584 ± 14
15	4	172	344	179.9 ± 0.7	157 ± 4
16	5	1292	2584	1683 ± 5	1420 ± 30
18	6	2217	4434	2432 ± 6	1940 ± 40

Table 2. Mean performance coefficient of PROJECTION *vs.* AGGREGATION on synthetic challenge instances.

l	d	PROJECTION Perf. Coeff.	AGGREGATION Perf. Coeff.	PROJ. m	AGGR. m
9	2	0.08 ± 0.03	0.16 ± 0.05	2966	1483
11	3	0.04 ± 0.02	0.08 ± 0.03	4886	2443
13	4	0.05 ± 0.03	0.05 ± 0.02	8356	4178
14	4	0.74 ± 0.04	0.78 ± 0.03	1294	647
15	4	0.93 ± 0.02	0.93 ± 0.02	344	172
16	5	0.59 ± 0.08	0.72 ± 0.06	2584	1292
18	6	0.68 ± 0.07	0.70 ± 0.07	4434	2217

Table 3. Mean solution time of PROJECTION *vs.* AGGREGATION on synthetic challenge instances.

l	d	PROJECTION Mean Solution Time [CPU sec]	AGGREGATION Mean Solution Time [CPU sec]	Soln. Time Ratio PROJECTION : AGGREGATION
9	2	23 ± 9	5 ± 1.6	4.6 ± 2.4
11	3	57 ± 17	20 ± 6	2.9 ± 1.2
13	4	280 ± 120	97 ± 53	2.9 ± 2.0
14	4	100 ± 20	38 ± 8	2.7 ± 0.8
15	4	6.2 ± 0.5	2.5 ± 0.3	2.5 ± 0.3
16	5	410 ± 140	160 ± 60	2.5 ± 1.2
18	6	1100 ± 300	460 ± 120	2.4 ± 0.9

is due to the significant reduction in unnecessary refinements. The ratios correspond directly to those in Table 3, as the bulk of the computation time is spent performing refinement.

3.2 Results for Biased GC Content

Since the base distribution in real DNA sequences is frequently non-uniform, it is important that motif finding algorithms perform well when the background sequences have non-uniform base distributions. Like Buhler and Tompa [2], we consider input sequences where the background $G + C$ fraction is below 50%. For each of our synthetic input sequences, each background base is chosen randomly, with $\Pr[G] = \Pr[C]$, $\Pr[A] = \Pr[T]$, and $\Pr[G] + \Pr[C] = 1 - \Pr[A] - \Pr[T] = G + C$ fraction. By symmetry, the results apply for $G + C$ fractions above 50%, as well as for all other cases with pairwise equal base frequencies. The planted motif itself remains randomly generated with all bases equally likely.

Moving the $G + C$ fraction away from 50% skews the base distribution of background l-mers compared to the base distribution in the planted motif. When determining whether a bucket should be sent to refinement, both AGGREGATION and PROJECTION take into account the expected number of background l-mers in that bucket based on its fingerprint by using a higher threshold for buckets that expect more background l-mers. However, because threshold values are integers and AGGREGATION uses higher thresholds than PROJECTION it can adjust the threshold more subtly based on the fingerprint of the bucket being inspected and is hence able to solve this type of problem more effectively.

Buhler and Tompa [2] state that the performance of PROJECTION on sequences with biased base distribution is limited by the increasing number of spurious motifs "as good as the planted motif." This is true for PROJECTION, but it is not true for AGGREGATION. While these spurious motifs are often as well preserved as the planted motif, *i.e.* they often appear in all of the input sequences, they are seldom as statistically significant as the planted motif. Since the base distribution of the planted motif remains uniform (as is common in nature), it 'sticks out' from biased background sequences more than spurious motifs do and is therefore discernable from spurious motifs by its greater likelihood ratio, providing it can be found at all.

Our experimental results for variations of the $(15, 4)$ challenge instance clearly indicate that for skewed $G + C$ background distributions, the performance advantage of AGGREGATION over PROJECTION increases (see Figure 3 and Table 5). Even stronger results were obtained for $(16, 5)$ instances, where for all background distributions including the uniform distribution, AGGREGATION consistently reaches higher performance coefficients than PROJECTION given the same amount of CPU time (see Figure 4). The same is true for other difficult (l, d) combinations (see Figure 6). Furthermore, qualitatively similar performance differences arise for non-uniform background distributions with a single-base bias (see Table 7).

3.3 Other Variations of the Challenge Problem

We evaluated the performance of AGGREGATION on two other variations of the previously studied challenge problems: problem instances with longer background sequences and problem instances in which the motif is planted in fewer than t input sequences. For each of these experiments, we generated several sets of 50 problem instances each and ran AGGREGATION and PROJECTION on every instance from these sets. Unless otherwise stated, the instances were generated using the following parameters: $(l, d) = (15, 4)$, $n = 600$ and $t = 20$.

Varying background sequence length It is natural to apply AGGREGATION to input sequences of length greater than $n = 600$. As n increases, the number of noisy l-mers increases. This not only means that the signal of the planted motif is weaker relative to the noise, but also that spurious motifs are more likely to occur. Because the increase in noise makes the signal harder to detect, both PROJECTION and AGGREGATION recover motifs more slowly.

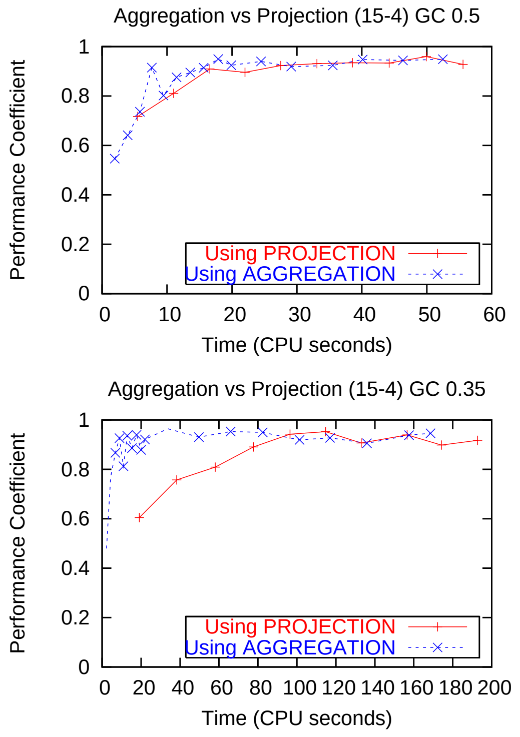


Fig. 3. Mean performance coefficient for PROJECTION *vs.* AGGREGATION for $(l, d) = (15, 4)$, $t = 20$, $n = 600$ with background G+C fraction 50% (top pane) and 35% (bottom pane).

Table 6. Mean performance coefficient for GC = 0.35 of PROJECTION *vs.* AGGREGATION on 50 randomly generated synthetic challenge instances for difficult (l, d) combinations.

l	d	PROJECTION Perf. Coeff.	AGGREGATION Perf. Coeff.	PROJ. Time [CPU sec]	AGGR. Time [CPU sec]
9	2	0.06 ± 0.06	0.28 ± 0.10	466	434
11	3	0.01 ± 0.00	0.19 ± 0.09	471	446
13	4	0.02 ± 0.02	0.18 ± 0.09	454	431
14	4	0.46 ± 0.13	0.75 ± 0.10	437	431
16	5	0.34 ± 0.12	0.70 ± 0.10	426	427
18	6	0.37 ± 0.13	0.76 ± 0.10	474	407

These tests were run on the same machine as all others, but under SUSE Linux 9.1.

Table 4. Solutions per 10^6 refinements of PROJECTION *vs.* AGGREGATION on synthetic challenge instances.

l	d	PROJECTION Solutions	AGGREGATION Solutions	Ratio of Successful Refinements AGGREGATION : PROJECTION
9	2	500 ± 200	2500 ± 800	4.7 ± 2.5
11	3	220 ± 70	640 ± 190	2.9 ± 1.2
13	4	46 ± 19	130 ± 70	2.9 ± 1.9
14	4	100 ± 20	280 ± 60	2.7 ± 0.8
15	4	1640 ± 130	4200 ± 600	2.6 ± 0.4
16	5	31 ± 10	80 ± 30	2.6 ± 1.3
18	6	10 ± 3	25 ± 6	2.5 ± 0.9

Table 5. Mean solution time and performance coefficient for PROJECTION *vs.* AGGREGATION for $(l, d) = (15, 4)$, $t = 20$, $n = 600$ with varying G+C fraction in the background distribution.

G+C Fraction	PROJECTION Mean Solution Time [CPU sec]	AGGREGATION Mean Solution Time [CPU sec]	Ratio of Times PROJECTION : AGGREGATION	PROJECTION Performance Coefficient	AGGREGATION Performance Coefficient
50%	6.2 ± 0.5	2.5 ± 0.3	2.5 ± 0.3	0.93 ± 0.02	0.93 ± 0.02
45%	8.3 ± 0.4	3.2 ± 0.2	2.6 ± 0.2	1.0 ± 0.0	1.0 ± 0.0
40%	11.0 ± 0.5	4.6 ± 0.3	2.4 ± 0.2	0.75 ± 0.01	0.76 ± 0.01
35%	12.7 ± 0.5	1.9 ± 0.1	6.8 ± 0.4	0.85 ± 0.06	0.91 ± 0.00
30%	7.8 ± 0.3	1.4 ± 0.0	5.6 ± 0.3	0.36 ± 0.14	1.0 ± 0.0
25%	5.5 ± 0.1	1.2 ± 0.0	4.4 ± 0.1	0.04 ± 0.06	0.24 ± 0.12

Table 7. Mean performance for PROJECTION *vs.* AGGREGATION for $(l, d) = (15, 4)$, $t = 20$, $n = 600$ with varying single base bias of the background distribution (all other bases are generated with equal probability).

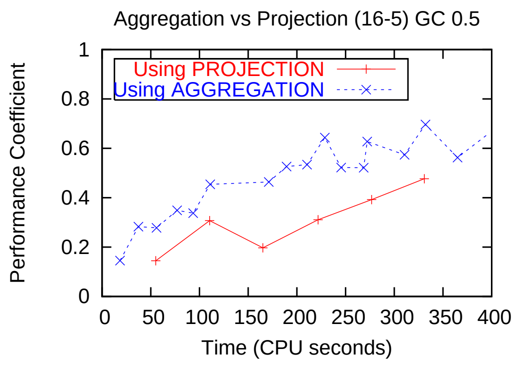
Single Base Fraction	PROJECTION		AGGREGATION	
	Performance Coefficient	Solution Time [CPU sec]	Performance Coefficient	Solution Time [CPU sec]
0.4	0.74 ± 0.11	36.3 ± 0.6	0.90 ± 0.06	31.0 ± 0.7
0.46	0.12 ± 0.09	33.4 ± 0.4	0.96 ± 0.04	27.8 ± 0.8
0.51	0.10 ± 0.08	30.0 ± 0.4	0.96 ± 0.02	29.3 ± 0.8
0.55	0.00 ± 0.00	31.4 ± 0.3	0.71 ± 0.13	29.4 ± 1.0
0.61	0.02 ± 0.04	32.7 ± 0.3	0.46 ± 0.14	31.8 ± 1.1

Table 8. Mean solution time of PROJECTION *vs.* AGGREGATION for $(l, d) = (15, 4)$, $t = 20$ with uniform background distribution and varying length of background sequences.

n	PROJECTION Mean Solution Time [CPU sec]	AGGREGATION Mean Solution Time [CPU sec]	Ratio of Times AGGREGATION : PROJECTION
600	6.6 ± 1.0	2.6 ± 0.5	2.5 ± 0.6
800	27 ± 4	10 ± 2	2.6 ± 0.6
1000	82 ± 25	32 ± 12	2.6 ± 1.3
1200	250 ± 60	96 ± 23	2.6 ± 0.8
1400	600 ± 140	190 ± 60	3.2 ± 1.2
1600	1000 ± 200	420 ± 90	2.5 ± 0.8

However, as can be seen in Table 8, AGGREGATION maintains its speed advantage as n increases. The two algorithms achieved nearly identical performance coefficients in these tests .

Varying number of planted instances When analysing real biological data, the motif we are looking for may not be present in every input sequence. Therefore we are interested in synthetic cases where the motif is not planted in every input sequence. When the motif is not planted in every sequence its signal is weaker; this makes it more difficult to detect. Also, since we do not ask for a motif that is present in every input sequence, spurious motifs are more likely to occur. As when background sequences are increased in length, both PROJECTION and AGGREGATION recover motifs more slowly when the motif is planted in fewer input sequences, but again, AGGREGATION maintains a significant speed advantage (see Table 9) while achieving roughly the same performance coefficient. Note that when the motif consensus is planted in fewer than 12 of 20 sequences, both PROJECTION and AGGREGATION recover spurious motifs very quickly because they are so abundant.



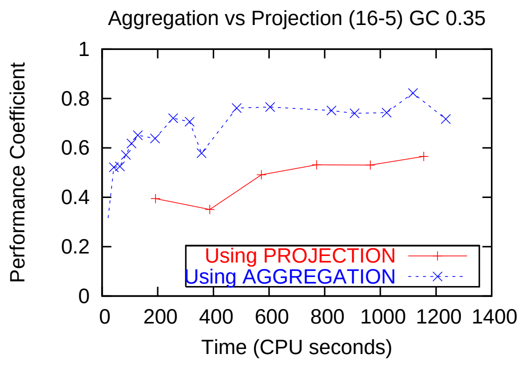


Fig. 4. Mean performance coefficient for PROJECTION *vs.* AGGREGATION for $(l, d) = (16, 5)$, $t = 20$, $n = 600$ with background G+C fraction 50% (top pane) and 35% (bottom pane).

Table 9. Mean solution time of PROJECTION *vs.* AGGREGATION for $(l, d) = (15, 4)$, $t = 20$ and $n = 600$ with uniform background distribution when the motif is planted in fewer than t input sequences.

Plantings	PROJECTION Mean Solution Time [CPU sec]	AGGREGATION Mean Solution Time [CPU sec]	Ratio of Times AGGREGATION : PROJECTION
20	6.2 ± 0.5	2.5 ± 0.3	2.5 ± 0.3
18	11 ± 2	5.0 ± 1	2.1 ± 0.6
16	30 ± 6	14 ± 3	2.1 ± 0.7
14	110 ± 30	42 ± 15	2.6 ± 1.2
12	150 ± 60	110 ± 50	1.4 ± 0.9
10	4.5 ± 0.4	4.0 ± 0.4	1.4 ± 0.9

3.4 Results for Biological Data

To test AGGREGATION on more realistic motif discovery problems, we evaluated its performance on the same biological data used by Buhler and Tompa [1], which were kindly provided to us by Jeremy Buhler.

The first biological data set contains orthologous eukariotic promotor sequences from various organisms that occur upstream of preproinsulin, dihydrofolate reductase (DHFR), metallothioneins and c-*fos* genes and are known to contain specific transcription factor binding sites. This set also contains seqeuences from *S. cerevisiae* that are known to contain the ECB element as a common promoter. The motifs in these data sets are very well preserved; the challenge lies in the fact that the background sequences are very long. We used parameters $l = 20$, $d = 2$, just as Buhler and Tompa did, but we increased k from 7 to 13 to achieve a large spread in the background l-mers.

AGGREGATION solved each of the problems in under 15 CPU seconds on our reference machine and almost always discovered the same motifs discovered as previously reported for PROJECTION [1]. The one exception was the third motif discovered for metallotionein, where AGGREGATION had a single point mutation error in the reported motif consensus. In discovering the known motifs, AGGREGATION refined only 10%–30% as many buckets as PROJECTION.

The second biological test set consists of 20 *C. elegans* promoter regions, each of length 1000, that contain the ‘X box’ motif [1]. This motif is somewhat harder to find, because it is not as well preserved as those in the eukaryotic promoter sequences. The motifs found by AGGREGATION precisely match those discovered by PROJECTION; both algorithms recovered 19 of the 20 known motif occurrences, and the 20th recovered occurrence had a higher likelihood ratio than the previously known occurrence. Maintaining its ability to select better starting points for refinement than PROJECTION, AGGREGATION refined only 691 buckets compared to 2832 buckets refined by PROJECTION.

The final biological test set consists of prokaryotic ribosome binding sites. These problem instances consist of thousands of short DNA sequences ($n = 20$) taken from the upstream region of the translation start sites of genes from a number of prokaryotic organisms. The ribosome binding sites are very short subsequences that have a complementarity to a section near the 3’-end of the respective organism’s 16S rRNA. One of the challenges with this data is that not all of the sequences contain a ribosome binding site [2]. Furthermore, as is often the case in prokaryotic genomic DNA, the sequences show highly skewed base distributions.

In our experiments we used the same parameters for PROJECTION as Buhler and Tompa [2]. PROJECTION used fixed thresholds determined by an equation in [2]. For AGGREGATION, we used $k = 5$ and the same m values as for PROJECTION. Since there are only 30 possible hashes with $l = 6$, $k = 4$, only 30 deterministic runs would be required for PROJECTION; likewise, only 6 deterministic runs would be required for AGGREGATION.

The refinement thresholds for AGGREGATION were chosen in a fairly *ad hoc* manner, since the parameters of the data varied significantly. Standard AGGREGATION thresholds were used, then raised or lowered until some but not many refinements were made. AGGREGATION recovered a motif consistent with the 16S rRNA and the highest z-scoring pentamer in all cases except for *E. coli*, where the motif found by AGGREGATION had a higher likelihood ratio (*i.e.* was more probabilistically significant) than the motif recovered by PROJECTION, which was consistent with the 16S rRNA and the highest z-scoring pentamer. On the other hand, unlike PROJECTION, AGGREGATION found a site for *H. influenzae* that fully agrees with the respective 16S rRNA sequence. Hence overall, the quality of motifs recovered by AGGREGATION appears to be as good as of those found by PROJECTION, but AGGREGATION always refined fewer than 10% as many buckets.

AGGREGATION was far more efficient in all biological tests while recovering motifs of the same quality. When PROJECTION consistently recovers the planted motif, AGGREGATION can only outperform it in terms of speed. We therefore challenge molecular biologists to suggest known (or suspected) motifs occurring in nature that cannot be discovered by PROJECTION — we suspect that AGGREGATION may be able to discover some of these.

4 CONCLUSIONS AND FUTURE WORK

We have developed an enhanced version of Buhler and Tompa's PROJECTION algorithm [1] for the motif discovery problem. This modification, which we call AGGREGATION, typically runs more than twice as fast as the original PROJECTION algorithm when solving the challenge problem posed by Pevzner and Sze [5]. This increase in speed is obtained without sacrificing accuracy. The AGGREGATION algorithm maintains its speed advantage in more difficult variations of the challenge problem, such as when there are more background l-mers and when the motif is not planted in every input sequence.

More important than the speed increase, AGGREGATION significantly outperforms PROJECTION on difficult problem instances with biased base distribution in the background sequences. AGGREGATION can, in fact, solve many of these instances that PROJECTION cannot solve at all.

According to our experimental results, AGGREGATION also achieves at least the same solution quality as PROJECTION on biological data, while requiring substantially less runtime. Since PROJECTION is still a state-of-the-art method for solving certain hard motif discovery problems, we believe that AGGREGATION represents an improvement in the state-of-the-art of solving such problems, especially for problems in which the background base distribution deviates significantly from uniform.

A different improvement of the PROJECTION algorithm has been made by Raphael *et al.* [7]. Instead of using a completely random projection for each iteration, they use 'uniform projections', *i.e.* projections chosen to more uniformly cover the space of all possible projections. This enables PROJECTION to achieve the same success rate while running for 20-50 percent fewer iterations. The projection improvement of Raphael *et al.* is disjoint from our thresholding improvement; it is therefore natural to apply both improvements at the same time. We predict that using AGGREGATION with uniform projections will give all of the advantages of AGGREGATION with an even greater increase in speed. We have yet to implement and test this, though we intend to do so in the near future.

It should be mentioned that the more recent PatternBranching algorithm by Price *et al.* [6] appears to be able to find solutions to the $(15, 4)$ challenge problem substantially faster than PROJECTION and AGGREGATION; however, this seems to come at the cost of slightly reduced accuracy. Furthermore, it is not clear how well this result extends to the more challenging variations of the (l, d) problem studied here. We have conducted preliminary experiments in which we compared the time required for PatternBranching and AGGREGATION to reach given success probabilities and found the following. For the $(15, 4)$ challenge problem AGGREGATION achieves over 90% success rate after 12 CPU seconds, and 100% success rate after 18 CPU seconds on our reference machine (compared to 17 seconds and 33 seconds for PROJECTION, respectively). PatternBranching requires a mere 0.9 seconds to achieve a success rate just below 100%. Although it is notably slower than PatternBranching, AGGREGATION can trade increased execution time for increased accuracy, whereas PatternBranching is limited by the number of l-mers in the input sequences. Also, when the G+C content is decreased to 0.35, the average runtime for PatternBranching rises to 93 CPU seconds for an average success rate of 98%, whereas AGGREGATION achieves the same success rate after 18 seconds (compared to 77 seconds for PROJECTION).

Given the strengths of both methods, an obvious direction for future work is to explore ways in which the approaches underlying AGGREGATION and PatternBranching (or ProfileBranching [6]) can be combined into even more powerful motif discovery algorithms. We made an initial attempt to incorporate PatternBranching's local search strategy into AGGREGATION by using it to replace the EM refinement stage. However, this variant of AGGREGATION does not appear to perform as well as AGGREGATION. Further investigation will tell whether different combinations of AGGREGATION and PatternBranching will achieve better performance.

Relative to other motif finding algorithms, AGGREGATION is most successful at solving problems in which the background base distribution deviates significantly from uniform. We are currently investigating ways to improve the performance of AGGREGATION as the background base distribution becomes more and more extreme. We also intend to apply AGGREGATION to problems arising in real biological data that were not solved by PROJECTION or PatternBranching.

ACKNOWLEDGEMENTS

We thank Jeremy Buhler for useful comments, and for making his PROJECTION source code and data publicly available. We also thank Mohammed Alam and Juan Estrada for their valuable contributions in the early stages of this project.

REFERENCES

[1]Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In *Proceedings of the Fifth International Conference on Computational Biology (RECOMB-01)*, pages 69–76, New York, April 22–25 2001. ACMPress.

[2]Jeremy Buhler and Martin Tompa. Finding motifs using random projections. *Journal of Computational Biology*, 9(2):225–242, April 2002.

[3]U. Keich and P.A. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–81, Oct 2002.

[4]C. E. Lawrence and A. A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7:41–51, 1990.

[5]Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB-00)*, pages 269–278, Menlo Park, CA, August 16–23 2000. AAAI Press.

[6]Alkes Price, Sriram Ramabhadran, and Pavel Pevzner. Finding subtle motifs by branching from sample strings. In *Proceedings of the Second European Conference on Computational Biology (ECCB-03)*, pages 69–76, September 2003.

[7]Benjamin Raphael, Lung-Tien Liu, and Varghese Varghese. A uniform projection method for motif discovery in DNA sequences. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(2):91–94, April 2004.