

# Improvement of the PROJECTION Motif Finding Algorithm

Mohammed Alam

University of British Columbia  
Department of Computer Science  
malam@cs.ubc.ca

Juan Gabriel Estrada

University of British Columbia  
Department of Computer Science  
estrada@cs.ubc.ca

Warren Cheung

University of British Columbia  
Department of Computer Science  
wcheung@cs.ubc.ca

James King

University of British Columbia  
Department of Computer Science  
king@cs.ubc.ca

March 16, 2004

## Abstract

In 2001 Buhler and Tompa [1] introduced their PROJECTION algorithm for solving the motif discovery problem for motifs of fixed length  $l$ . The algorithm finds good candidates for local optimisation by projecting all  $l$ -mers present in the input onto a smaller subspace and analysing dense points in that subspace. We have developed AGGREGATION, a modified version of PROJECTION that analyses dense regions of the subspace rather than dense points. Our subtle modification more than doubles the speed of the algorithm for many problem spaces while maintaining the same level of accuracy.

## 1 Introduction

Given a number of DNA sequences, the motif finding problem is the task of discovering a particular base sequence that appears (perhaps in a slightly mutated form) in every given sequence. We are considering the *challenge problem* as defined by Pevzner and Sze[8], and as stated by Buhler and Tompa[1]:

**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  $M$ .

Much work has been done on the problem, but there are certain variations of the problem for which satisfactory solutions have not yet been found. The problem becomes harder if, for example, insertions/deletions are considered to be legal mutations or the motif does not occur in all of the  $t$  background sequences.

## 2 Related Work

Pevzner and Sze [8] posed their challenge problem for motif discovery and introduced two new algorithms that were more successful than previous attempts had been. WINNOWER treats every occurring  $l$ -mer as a node in a graph, with nodes adjacent if and only if they differ in at most  $2d$  positions and they occur in different sequences. It then finds cliques of size  $t$  and works from those starting points. SP-STAR tries testing every occurring  $l$ -mer in turn and considers the possibility that it is a mutated occurrence of the motif consensus. Using this technique it essentially does an enumerative search, but only over the  $l$ -mers occurring within data rather than the entire space of  $4^l$   $l$ -mers.

More recently there have typically been two approaches used by motif discovery algorithms. Profile-driven approaches make use of weight matrices to calculate the score of different candidate motifs, returning the highest scoring motif as an answer. Pattern-driven approaches search for a motif of length  $l$  by assigning each of all possible  $4^l$   $l$ -letter patterns a score based on its number of occurrences in the sample (or another more sophisticated function). Clearly the latter is impractical for large  $l$ . Therefore a variation of the pattern approach — the sample-driven approach — limits the set of possible candidates but may have difficulty detecting subtle motifs. Sinha and Tompa [10] designed an enumerative search algorithm for yeast (YMF, Yeast Motif Finder), which illustrates the reliability of the pattern-driven approach. However, this program requires as input the specific characteristics of the motif that is being sought and is only intended for finding short motifs.

More recently, Keich and Pevzner [4] designed MULTIPROFILER, a motif discovery algorithm that balances the pattern-driven and sample-driven approaches to yield a better ability to detect subtle motifs while avoiding the complexity of pure pattern-driven approaches. They also propose an objective tool for analysing the performance of motif finding algorithms [5], with particular focus on hard to detect, or 'subtle' motifs. They suggest that in these cases even the most reliable algorithms are more prone to picking up randomly occurring spurious motifs simply due to the overwhelming presence of spurious motifs that, according to the scoring models used, are as good or better than the planted motif.

Rocke [9] has proposed a modified version of the Gibbs sampling algorithm (GibbsDNA) of Lawrence et al. [6] that uses suffix trees for the purpose of gapped motif discovery. Pevzner and Sze also provide extensions to their SP-STAR algorithm to handle gapped and variable length motifs.

GuhaThakurta and Stormo [3] have noted that many of the regulatory signals that appear in unaligned DNA sequences are composite patterns that are groups of monad (single motif) patterns occurring near each other. Eskin and Pevzner [2] have noted that it is difficult to find composite

patterns using the current monad-based approaches, as such patterns are often composed of one or more monad signals that are 'too weak' (*i.e.* not statistically meaningful enough). They proposed a monad pattern discovery algorithm, MITRA (MISmatch TRee Algorithm), for discovering composite patterns by first considering the problem as a larger monad pattern discovery problem through preprocessing of the sample to be analysed.

### 3 The PROJECTION Algorithm

In 2002, Buhler and Tompa [1] introduced their PROJECTION algorithm. This algorithm 'projects' every  $l$ -mer from the given data onto a smaller space by hashing. The hash function is based on  $k$  of the  $l$  positions that are selected at random when the algorithm begins.  $l$ -mers are hashed into the same bucket if they have the same bases in those  $k$  positions. In this way,  $l$ -mers are hashed to  $k$ -mers; these are the fingerprints that correspond to the buckets.

The idea behind PROJECTION is that background  $l$ -mers (essentially random noise) will be distributed fairly evenly between the buckets. Meanwhile, the 'planted bucket' (*i.e.* the bucket into which the motif consensus would be hashed) will have additional  $l$ -mers because some occurrences of the planted motif will not be mutated in any of the  $k$  hashable positions. PROJECTION performs this hashing, then performs refinement on each sufficiently full bucket to find the best motif in the vicinity of the  $l$ -mers in that bucket; a 'sufficiently full' bucket is one that contains a number of  $l$ -mers greater than a threshold, where the threshold depends on input parameters  $l, d, n$ , and  $t$ . The algorithm is likely to find the consensus motif if it ends up refining the planted bucket. PROJECTION also sometimes finds the consensus motif by refining a bucket that is close to the planted bucket. By running the algorithm for a sufficient number of iterations, PROJECTION will refine the planted bucket in at least one run with high probability.

PROJECTION performs significantly better than many other motif finding algorithms, especially for harder instances of the problem such as (14, 4), (16, 5), and (18, 6). For this reason, we have decided to concentrate on improving PROJECTION, which has found great success in the  $l$ -mer hashing technique.

### 4 Modification of PROJECTION

The majority of PROJECTION's running time is taken up by the refinement stage that finds the best motif in the neighbourhood of a given bucket. For this reason, it would be especially advantageous to refine as few buckets as possible in our search for the planted bucket. The difficulty is that refining more buckets is currently advantageous in that it increases the probability that we refine the planted bucket at some point.

We have modified the projection process in a way that makes projection more complicated and time consuming, but allows us to refine few enough buckets that the time gained in refinement more than makes up for the time lost in projection. For the refinement stage of the algorithm, Buhler and Tompa used the expectation maximisation (EM) algorithm developed by Lawrence

and Reilly [7]. We have left this stage of the algorithm completely unmodified.

A major problem with the projection method is that most planted occurrences of the motif consensus will be hashed to buckets other than the planted bucket, essentially being thrown away and considered as noise. The probability that many of these occurrences will be hashed to the planted bucket is small. However, a significant number of motif occurrences will land near, but not in, the planted bucket. Rather than simply refining any bucket that contains a sufficiently large number of  $l$ -mers, we refine any bucket *whose neighbourhood* contains a sufficiently large number of  $l$ -mers, where ‘sufficiently large’ means greater 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). We define the neighbourhood of a bucket  $B$  to be the bucket itself, plus all buckets whose corresponding fingerprints are at Hamming distance 1 from the fingerprint corresponding to  $B$ . We could extend this neighbourhood to include buckets further from  $B$ , thereby further increasing the probability of a planted occurrence falling into the neighbourhood of the planted bucket, but this would make our threshold tests prohibitively expensive.

We will refer to the number of  $l$ -mers hashed to a bucket’s neighbourhood as that bucket’s *score*. Because this value is obtained by aggregating the scores of individual buckets, we will refer to our modified method as AGGREGATION. By performing this score aggregation we essentially focus the effect of the significant data (planted motif occurrences) in relation to the noise (randomly occurring  $l$ -mers). By doing this, we improve the chances of sending the planted bucket to refinement. This means that fewer total refinements are required and the algorithm therefore runs faster.

## 5 Analysis

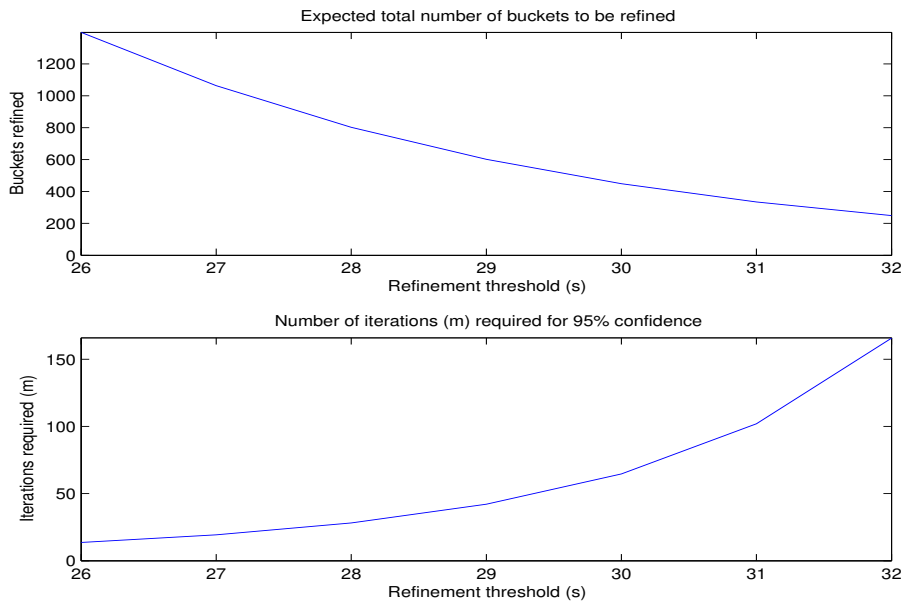
Mathematical analysis of AGGREGATION (and PROJECTION, for that matter) is impractical unless we make the assumption that background  $l$ -mers are independently distributed. This, of course, is not true because some  $l$ -mers in the original data overlap each other. However, the overlap is sufficiently short relative to the background sequence that analysis under our assumption is applicable to the real case.

For any given set sequences, PROJECTION’s probability of discovering the consensus motif in an iteration is fixed. That is, with some fixed probability the projection positions will be chosen in such a way that the consensus motif is found in the refinement. Therefore, for a given set of input sequences, we can consider one iteration of PROJECTION to be a Poisson trial. The expected number of times the consensus is recovered in  $m$  iterations follows a simple binomial distribution (though the parameters for this distribution are extremely complex and can only be estimated). Also, the probability that we *do not* recover the consensus decays exponentially as  $m$  increases (assuming that PROJECTION is capable of recovering said consensus at all). Naturally, the same things can be said for AGGREGATION.

The probability of refining the planted bucket in a single iteration will vary depending on the input data. In terms of performing meaningful analysis we can approximate the expectation of this probability, which is particularly relevant to the challenge problem. It should be noted that

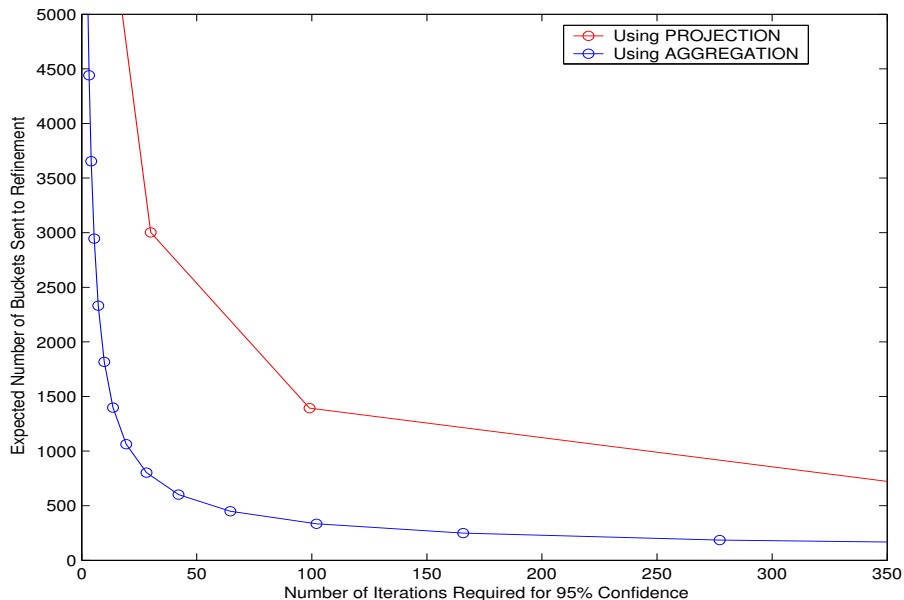
this probability is not exactly the same as the probability of recovering the consensus motif since refinement of the planted bucket will not always yield the consensus and the consensus can be refined from buckets other than the planted one.

Figure 1: Expected number of buckets refined and iterations required for AGGREGATION with varying 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.



By determining the expected probability of refining the planted bucket and the expected probability of refining any other bucket we can make some theoretical comparisons between PROJECTION and AGGREGATION, as well as choose good threshold values for AGGREGATION. Choosing a good threshold is 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 (see Figure 1). To compare PROJECTION and AGGREGATION we use our analysis to determine how many iterations we need in order to refine the planted bucket with 95% probability and how many buckets we should expect to refine during that process. We choose thresholds for AGGREGATION that keep both of these values low. In terms of comparison our analysis suggests that, in general, AGGREGATION can achieve 95% confidence after refining less than half as many buckets as PROJECTION, given that we want to run both methods for the same number of iterations (see Figure 2). The estimated number of buckets refined is equal to the estimated number of iterations required to refine the planted bucket with 95% probability times the expected number of buckets (planted or spurious) refined in each iteration. In terms of assumptions made, these estimates only use the (false but fairly safe) assumption that background  $l$ -mers are independently distributed and the assumption that the planted motif will be recovered if and only if the planted bucket is sent to refinement.

Figure 2: Theoretical comparison of PROJECTION and AGGREGATION with varying thresholds. The curves sweep down and to the right as the thresholds increase. This graph suggests that AGGREGATION does not need to refine as many buckets as PROJECTION.



Because of the simplifications made in our analysis and the fact that AGGREGATION will be slower per iteration than PROJECTION, we cannot consider this to be any kind of proof that AGGREGATION is the superior method. Fortunately we see similar results in our experiments.

## 6 Experimentation

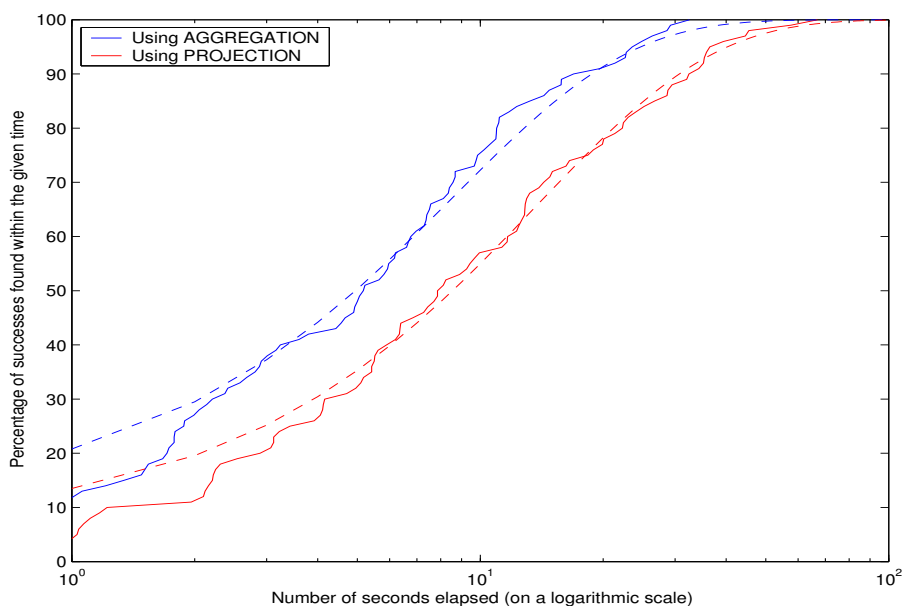
As stated before, for any fixed input we can consider one iteration of PROJECTION or AGGREGATION to be a Poisson trial. Therefore, in comparing the performance of PROJECTION to that of AGGREGATION for a fixed input, the only two things that need to be measured are the probabilities of the Poisson trials and the average iteration runtimes. The probabilities can be approximated by running the algorithms for a large value of  $m$  and, for each algorithm, dividing the number of successful iterations by  $m$ . The average iteration times, of course, can be determined by dividing the total runtimes of the two methods by  $m$ . Combining these two values we can obtain the expected number of successes per second.

Doing this in our experimentation gives us a fair, linear measurement of quality in terms of successes per second. That is, for each method we can determine the expected number of times the consensus motif will be recovered in one second of CPU time. Of course, this is all for a fixed set of input sequences. To measure the quality over a number of different inputs we must do this for each input and take the average of the measurements. For synthetic data this can be done quite

easily since we can sample our inputs uniformly at random from whatever problem space we want.

We should note that, since a single iteration is a Poisson trial, the length of time for which the algorithm runs until the first success follows an exponential distribution. We simply want the Poisson trials of AGGREGATION to have a greater success probability than those of PROJECTION. For a given input we can analyse these running times and manually fit an exponential distribution curve to our experimental data to approximate the success probabilities of our Poisson trials. Figure 3 gives a typical example of such data with the fitted cumulative exponential distribution curves.

Figure 3: The exponential distributions exhibited by the runtimes of AGGREGATION and PROJECTION on the same fixed input (when we stop at the first success). Dashed lines are the manually fitted curves of true exponential distributions. Distributions shown are cumulative and a log scale is used on the  $x$ -axis.



## 7 Benchmarking

To evaluate the efficacy of the AGGREGATION enhancement to the PROJECTION algorithm, both algorithms were tested on randomly generated synthetic data. AGGREGATION was modified from the original PROJECTION algorithm only to implement the hash-based random access data structure to allow for looking at all the neighbours of an arbitrary bucket. The parameters ( $k = 7, s = 4, m$ ) chosen for PROJECTION were the same as in [1], and the parameters for AGGREGATION were set to ( $k = 7, s = 28$ ) (see Figure 1 to see why this threshold is reasonable). The number of iterations  $m$  for AGGREGATION was set to twice the value of  $m$  selected for PROJECTION to ensure that the two methods were given comparable amounts of CPU time without AGGREGATION

being given an advantage (actual runtimes are shown in Table 1). For each of the  $(l, d)$  combinations, 100 randomly generated sets of input sequences ( $t = 20, n = 600$ ) were generated (as in [1]), and PROJECTION and AGGREGATION were both run once on each of these sequences. The confidence intervals for the means are 95% intervals; this confidence is over all tested instances from the respective test sets.

Table 1: Experimental setup

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

Comparison of the mean performance coefficients obtained by PROJECTION and AGGREGATION is given in Table 2. The performance coefficient is defined in [8] as follows. Let  $K$  be the set of  $l \cdot t$  base positions in the  $t$  planted motif instances and let  $P$  be the set of  $l \cdot t$  base positions in the motif instances predicted by the algorithm. The performance coefficient is equal to  $|K \cap P|/|K \cup P|$ . AGGREGATION slightly beats out PROJECTION in terms of this metric, though a more meaningful comparison can be seen in Table 3. As the actual motif becomes occluded by equally well preserved spurious motifs in the harder problem instances, we compare the number of times AGGREGATION and PROJECTION successfully find a motif, either correct or spurious, that occurs in all 20 sequences, as done by Keich and Pevzner in [4].

Table 2: Mean performance coefficient comparison

l	d	PROJECTION	AGGREGATION
		Performance	Performance
9	2	$0.08 \pm 0.03$	$0.16 \pm 0.05$
11	3	$0.04 \pm 0.02$	$0.08 \pm 0.03$
13	4	$0.05 \pm 0.03$	$0.05 \pm 0.02$
14	4	$0.74 \pm 0.04$	$0.78 \pm 0.03$
15	4	$0.934 \pm 0.015$	$0.93 \pm 0.02$
16	5	$0.59 \pm 0.08$	$0.72 \pm 0.06$
18	6	$0.68 \pm 0.07$	$0.70 \pm 0.07$

As can be seen in Table 4, for all problem instances AGGREGATION discovers the consensus at at least twice the rate of PROJECTION. Higher discovery rates imply that AGGREGATION is able to determine a solution in less time than PROJECTION and is therefore more sensitive to the presence of projected motif buckets.

We can also compare the quality of the AGGREGATION thresholding process by comparing the number of refinements which yield the consensus to the total number of refinements. As can



Table 3: Proportion of runs in which the recovered consensus had planted instances in all 20 sequences.

l	d	PROJECTION Proportion	AGGREGATION Proportion
9	2	$0.47 \pm 0.05$	$0.71 \pm 0.04$
11	3	$0.80 \pm 0.04$	$0.86 \pm 0.03$
13	4	$0.83 \pm 0.03$	$0.89 \pm 0.03$
14	4	$0.90 \pm 0.03$	$0.970 \pm 0.017$
15	4	1	1
16	5	$0.68 \pm 0.05$	$0.85 \pm 0.04$
18	6	$0.74 \pm 0.04$	$0.79 \pm 0.04$

Table 4: Mean solution speed comparison

l	d	PROJECTION Successes per $10^3$ Seconds	AGGREGATION Successes per $10^3$ Seconds	Ratio of Rates AGGREGATION : PROJECTION
9	2	$43 \pm 18$	$200 \pm 60$	$5 \pm 2$
11	3	$18 \pm 5$	$51 \pm 15$	$2.9 \pm 1.2$
13	4	$3.5 \pm 1.4$	$10 \pm 6$	$3 \pm 2$
14	4	$10 \pm 2$	$26 \pm 5$	$2.7 \pm 0.8$
15	4	$162 \pm 13$	$400 \pm 40$	$2.5 \pm 0.3$
16	5	$2.4 \pm 0.8$	$6 \pm 2$	$2.5 \pm 1.2$
18	6	$0.9 \pm 0.2$	$2.2 \pm 0.6$	$2.4 \pm 0.9$

be seen in Table 5, the primary performance gain from AGGREGATION is due to the significant reduction in unnecessary refinements. The ratios correspond directly to those in Table 4, as the bulk of the computation time is spent performing refinement.

Table 5: Solutions per  $10^6$  refinements comparison

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

Our experiments show with a high level of confidence that our AGGREGATION method significantly outperforms the original version of PROJECTION in the basic form of the challenge problem. In the next section we show that AGGREGATION outperforms PROJECTION in different variations of the challenge problem.

Unfortunately we have not yet acquired any biological data. Keich and Pevzner [5] tested their

MULTIPROFILER algorithm on a good assortment of biological data. We would like to test AGGREGATION on the same data (and possibly more). It should be noted that AGGREGATION should succeed in the biological tests in which PROJECTION succeeds.

## 8 Problem Variations

To ensure that AGGREGATION’s speed advantage translates to a wide range of motif finding problem spaces, we performed speed comparisons on different variations of the challenge problem. For each of these tests, we ran AGGREGATION and PROJECTION on 50 random inputs from each problem space. The input variations we consider are

- planted motif instances in background sequences that have non-uniform base distributions;
- planted motif instances in background sequences of increased length;
- motif instances that are not planted in all of the background sequences.

The first two variations were dealt with by Buhler and Tompa [1] for the original version of PROJECTION. Buhler and Tompa did not explicitly handle the third variation in their paper. All tests done in this section are for the  $(15, 4)$  problem, with  $n = 600, t = 20$  unless otherwise stated.

### 8.1 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. However, as can be seen in Table 6, AGGREGATION maintains its speed advantage as  $n$  increases.

Table 6: Solution speed comparison for increased background sequence length

n	PROJECTION Successes per $10^3$ Seconds	AGGREGATION Successes per $10^3$ Seconds	Ratio of Rates AGGREGATION : PROJECTION
600	$152 \pm 23$	$377 \pm 75$	$2.5 \pm 0.6$
800	$37 \pm 5$	$95 \pm 17$	$2.6 \pm 0.6$
1000	$12 \pm 4$	$31 \pm 12$	$2.6 \pm 1.3$

### 8.2 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 need 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. Again, AGGREGATION maintains its speed advantage (Table 7).

Table 7: Solution speed comparison when the motif is planted in fewer than  $t$  input sequences, for  $t = 20$

Plantings	PROJECTION Successes per $10^3$ Seconds	AGGREGATION Successes per $10^3$ Seconds	Ratio of Rates AGGREGATION : PROJECTION
20	$158 \pm 14$	$361 \pm 42$	$2.3 \pm 0.4$
18	$94 \pm 18$	$202 \pm 45$	$2.1 \pm 0.6$
16	$34 \pm 7$	$72 \pm 16$	$2.1 \pm 0.7$
14	$9 \pm 3$	$24 \pm 8$	$2.6 \pm 1.2$
12	$7 \pm 3$	$9 \pm 4$	$1.4 \pm 0.9$

### 8.3 Varying G+C Fraction

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. As in [1], we consider input sequences where the background  $G + C$  fraction is below 50% (by symmetry, this is the same as considering sequences with  $G + C$  fraction above 50%). For each of our synthetic input sequences, each background base is chosen randomly, with

$$\Pr[G] = \Pr[C]$$

$$\Pr[A] = \Pr[T]$$

$$\Pr[G] + \Pr[C] = 1 - \Pr[A] - \Pr[T] = G + C \text{ fraction.}$$

Moving the  $G + C$  fraction away from 50% skews the distribution of background  $l$ -mers. When determining whether a bucket should be sent to refinement, AGGREGATION takes into account the expected number of background  $l$ -mers in that bucket based on its fingerprint. Both PROJECTION and AGGREGATION have always done this, though the expected number of background  $l$ -mers did not previously depend on which bucket was being inspected.

Because PROJECTION uses such low refinement thresholds, no change in threshold is subtle. Therefore PROJECTION cannot subtly set a threshold based on the fingerprint of the bucket being inspected. AGGREGATION can do this because it uses larger thresholds which can be more finely tuned. Essentially, the ‘problem’ of having a non-uniform background distribution is not a problem at all if AGGREGATION uses a higher threshold for buckets that expect more background  $l$ -mers.

As the  $G + C$  fraction moves further away from 50%, spurious motifs (mostly made up of the more common bases) are more likely to appear. If anything, AGGREGATION is less likely than PROJECTION to recover spurious motifs because of its higher thresholds for ‘noisy’ buckets. Therefore

solution speed is still a fair metric with which to demonstrate AGGREGATION’s superiority. As can be seen in Table 8, AGGREGATION’s speed advantage is maintained, and even increased, with decreasing  $G + C$  fraction.

Table 8: Solution speed comparison for varying G+C fraction

G+C Fraction	PROJECTION Successes per $10^3$ Seconds	AGGREGATION Successes per $10^3$ Seconds	Ratio of Rates AGGREGATION : PROJECTION
50%	$178 \pm 7$	$316 \pm 10$	$1.8 \pm 0.2$
45%	$120 \pm 6$	$312 \pm 15$	$2.6 \pm 0.2$
40%	$90 \pm 4$	$219 \pm 14$	$2.4 \pm 0.2$
35%	$78 \pm 3$	$530 \pm 21$	$6.76 \pm 0.4$
30%	$128 \pm 5$	$720 \pm 23$	$5.61 \pm 0.3$

The success rate of PROJECTION drops off significantly with decreased  $G + C$  fraction; for the (15, 4) problem, this dropoff occurs for  $G + C$  fraction below 35%. We suspect that AGGREGATION’s flexible thresholding will protect it from this dropoff, though we have yet to prove this experimentally.

## 9 Conclusion

We have developed a modification of Buhler and Tompa’s PROJECTION algorithm [1] for the motif discovery problem. This modification, which we call AGGREGATION, typically runs twice as fast as the original PROJECTION algorithm when solving the challenge problem posed by Pevzner and Sze [8]. This increase in speed is obtained without sacrificing accuracy.

The AGGREGATION method maintains its speed advantage in more difficult variations of the challenge problem, such as when there are more background  $l$ -mers or when the background base distribution is non-uniform. We intend to perform more experiments for these variations to determine whether AGGREGATION can actually solve problems that PROJECTION cannot; this may be the case for problems in which the background base distribution is far from uniform. We also intend to apply AGGREGATION to problems arising in real biological data that were not solved by PROJECTION.

**Acknowledgements.** We gratefully acknowledge Jeremy Buhler for making his PROJECTION source code publicly available and Holger Hoos for his guidance regarding our research.

## References

- [1] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In Thomas Lengauer, David Sankoff, Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors, *Pro-*

- ceedings of the Fith International Conference on Computational Biology (RECOMB-01)*, pages 69–76, New York, April 22–25 2001. ACM Press.
- [2] E. Eskin and P.A. Pevzner. Finding composite regulatory patterns in dna sequences. *Bioinformatics*, 18 Suppl 1:S354–63, Jul 2002.
  - [3] D. GuhaThakurta and G.D. Stormo. Identifying target sites for cooperatively binding factors. *Bioinformatics*, 17(7):608–21, Jul 2001.
  - [4] U. Keich and P.A. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–81, Oct 2002.
  - [5] U. Keich and P.A. Pevzner. Subtle motifs: defining the limits of motif finding algorithms. *Bioinformatics*, 18(10):1382–90, Oct 2002.
  - [6] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, October 1993.
  - [7] 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.
  - [8] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In Russ Altman, L. Bailey, Timothy, Philip Bourne, Michael Gribskov, Thomas Lengauer, and Ilya N. Shindyalov, editors, *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.
  - [9] E. Rocke. Using suffix trees for gapped motif discovery. In R. Giancarlo and D. Sankoff, editors, *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, number 1848 in Lecture Notes in Computer Science, pages 335–349, Montréal, Canada, 2000. Springer-Verlag, Berlin.
  - [10] Saurabh Sinha and Martin Tompa. A statistical method for finding transcription factor binding sites. In Russ Altman, L. Bailey, Timothy, Philip Bourne, Michael Gribskov, Thomas Lengauer, and Ilya N. Shindyalov, editors, *Proceedings of the 8th International Conference on Intelligent Systems for Molecular (ISMB-00)*, pages 344–354, Menlo Park, CA, August 16–23 2000. AAAI Press.

