On Half Gapped Seed

Wei Chen **Wing-kin Sung**
chenwei1@comp.nus.edu.sg ksung@comp.nus.edu.sg
School of Computing, National University of Singapore, Singapore, 117543

Abstract

In this paper, we proposed a new type of seed for Blast-like homology search tools called "half seed". This new seed is better than the "consecutive seed" used by the original Blast tools in both sensitivity and efficiency. When compared with the "gapped seed", which is proposed together with a new Blast-like searching tool, PatternHunter, this new seed offers a much wider range of choices for performing tradeoff between sensitivity and efficiency. This property is especially useful when some searching applications want to get more precise results with limitation on hardware resources, or vice versa.

Keywords: 'half match' position, half gapped seed

1 Introduction

Homology search is the problem of locating the approximate matches within one DNA sequence or between two sequences. This problem has a lot of applications in biology. Finding faster and more sensitive methods for homology search has attracted a lot of research works.

The first solution to the homology search problem is contributed by Smith and Waterman [11]. Their method is dynamic programming in nature and compares every base in the first sequence with every base in the other sequence to generate a precise local alignment. Although this method gives the most sensitive solution, it is also the slowest one. In order to improve the efficiency, without too much loss in sensitivity, many ideas are presented. Among them, FASTA [9], SIM [6], the Blast family (Altschul [1]; Gish, [5]; Altschul [2]; Zhang [15]; Tatusova and Madden, [13]), Blat [7], SENSEI [12], MUMmer [4], QUASAR [3], REPuter [8] and PatternHunter [10] are the most famous ones. All of these methods can be divided into two major tracks.

The first track is represented by MUMmer [4], QUSAR [3] and REPuter [8], which use suffix trees [14]. Two major problems make them less popular. First, although suffix tree is good in dealing with exact matches, it is not good for finding approximate matches. Therefore, methods based on suffix tree normally can only find matches with high homology. Second, suffix tree is very big and methods based on suffix tree suffer from the storage limitation.

The second track is represented by Blast, which is probably the most widely used approach now. Their basic idea is to finds short exact matches(hits) in the whole sequence first, which are then extended into longer alignments through dynamic programming process. FASTA [9], SIM [6], Blastn [13], WU-Blast [5], and Psi-Blast [2] encounter space and efficiency problem when they are used to compare relatively long sequences. SENSEI [12] is much faster and cost less working space, though it is incapable to allow gapped alignments. Blat [7] is a Blast-like homology searching tool, which is very fast to get results while it is limited by the high similarity requirements. MegaBlast [15] is the most efficient among Blast family, while its output is also rough.

Blast type methods all face an inevitable dilemma caused by the length of the exact match hit, that is, longer exact match hit increases the efficiency but reduces the accuracy; while shorter one gives better sensitivity but prolongs the executing time.

Ma et al. proposed the PatternHunter [10] to solve the awkward dilemma. They introduce the new idea, **gapped seed**, which is used to seek nonconsecutive short matches. The total number of nonconsecutive matches is called **weight** for their seeds. Once these matches are found, they are extended to longer alignments by dynamic programming. According to their experimental results, "gapped seeds" can reach both higher efficiency and better sensitivity than Blast's original consecutive seeds.

Depending on applications, we sometime require better sensitivity while we can tolerant a little decrease in efficiency. "Gapped seeds" allows us to perform such tuning only by changing its weight. More precisely, reducing the weight of the "gapped seed" brings better sensitivity while we should sacrifice a lot in efficiency. In other words, the "gapped seeds" are incapable of providing finely flexible tradeoff choices. For example, when we reduce the weight from 7 to 6, the sensitivity can be improved from 0.8 to 0.9 when two sequence have 0.6 similarity. But at the same time, the searching time is prolonged by 4 times! Such kind of tuning is too rough for many applications. Therefore, we would like to ask if we can give a better solution to solve the problem of tradeoff between the sensitivity and the efficiency.

This paper gives a positive answer to this question. We propose a new type of seed called "half seed". This new type of seed is a generalization of the gapped seed, which will be defined in detail in Section 2. Similar to the gapped seed, the half seeds are better than the existing consecutive seeds in both sensitivity and efficiency. Moreover, the half seeds provide a more flexible tradeoff between speed and sensitivity. Especially for the cases where we cannot afford to have a big jump in both efficiency and sensitivity, the half seeds are particularly useful.

The paper is organized as follows. Section 2 gives all the necessary and useful definitions for fully understanding what is a half seed. We also give a convenient notation to represent the different classes of seeds, which is used throughout this paper. Section 3 compares the half seeds with the gapped seeds in term of both sensitivity and efficiency by performing a series of experiments. The results show that the half seeds can really offer flexible choices of tradeoff than gapped seed between sensitivity and efficiency. In Section 4, we mention the impacts on sensitivity and efficiency when parameters are changed in our new seeds. From those results, we can have a fundamental idea of how to tune the tradeoff for "half seeds".

2 What is a Half Seed?

Before describing our new seeds, let's first have a brief review of the seeds used in Blast family and PatternHunter. These seeds can be represented using some $0-1$ strings of length L. What's the meaning for these 0 and 1? They represent two important definitions, 'match' positions and 'don't care' positions.

Definition 1 *Consider two length L substrings S and S' from the query sequence and the database sequence, respectively. Suppose position i of the seed is* 1*, which is denoted as the 'match' position. Then, (S, S') is said to have a match at position i, if $S[i] = S'[i]$.*

Definition 2 *Consider two length L substrings S and S' from the query sequence and the database sequence, respectively. Suppose position i of the seed is* 0*, which is denoted as the 'don't care' position. (S, S') is said to have a match at position i, no matter $S[i] = S'[i]$ or not.*

Definition 3 *For a length L seed, we say there is a hit when two length L substrings from query and database sequence match at all the corresponding positions in the seed.*

Definition 4 *We denote the seed which only contains 'match' positions "consecutive seed". We denote the seed which contains both 'match' positions and 'don't care' positions "gapped seed".*

For Blast, they use the "consecutive seed" 11111111111, which means every pair of length 11 substrings from query and database sequence should be identical at all these 11 'match' positions to get a hit. For PatternHunter, they use the "gapped seed" 110100110010101111, which means there is a hit for a pair of length 18 substrings from query and database sequence when they are identical at the 11 'match' positions regardless of those characters at the 7 'don't care' positions.

After we have an idea of the seeds used in Blast and PatternHunter, we will introduce our new seeds as follow. First of all, there is a fundamental definition called 'neighbor nucleotide'.

Definition 5 *Recall that every DNA sequence is composed of a set of* 4 *different nucleotides,* $N = \{A, C, G, T\}$*. For every nucleotide* $x \in N$*,* $neig\{x\}$ *is a predefined subset of* $N - \{x\}$*, which represents the set of neighbor nucleotides of* x*. When* $|neig\{x\}| = 2$*, we call it 'two neighbor' definition, and when* $|neig\{x\}| = 1$*, we call it 'one neighbor' definition.*

To generalize the gapped seeds to the half gapped seeds, apart from 'match' positions and 'don't care' positions, we need to introduce a new kind of positions known as 'half match' positions, which are defined as follows.

Definition 6 *Consider two length* L *substrings* S *and* S' *of the query sequence and the database sequence, respectively. Suppose position* i *of the seed is* 0.5*, which is denoted as the 'half match' position.* (S, S') *is said to have a match at position* i *, if* $S[i] = S'[i]$ *or* $S[i] \in neig\{S'[i]\}$*.*

Now, we are ready to define the "half seed" and the "half gapped seed".

Definition 7 *We call the seed which contains 'match' positions and 'half match' positions "half seed". We call the seed which contains 'match' positions, 'don't care' positions and 'half match' positions "half gapped seed".*

For example, 1 0.5 1 0 0 0.5 0 1 is a "half gapped seed" of length 8 with 3 match positions, 2 half match positions, and 3 don't care positions. This seed implies that there is a hit between two length 8 substrings from query and database sequence, S and S' respectively, when they are identical at all the 3 'match' positions, and $(S[2] \in neig\{S'[2]\}) \bigcap (S[6] \in neig\{S'[6]\})$, regardless of those characters at 'don't care' positions.

Based on the definition for 'neighbor nucleotides', we know that the probability of having a match at 'half match' positions depends on the definition of the 'neighbor nucleotide'. Such probability is $\frac{3}{4}$ if we use the 'two neighbor' definition and is $\frac{1}{2}$ if we use the 'one neighbor' definition.

To ease the description of the seed, we name the seeds according to their composition of match positions, half match positions and don't care positions. More precisely, if a seed has s_1 match positions, s_2 half match positions in 'one neighbor' definition, s_3 half match positions in 'two neighbor' definition, and s_4 don't care positions, then we denote the seed as a (s_1, s_2, s_3, s_4) seed. For example, $(6, 0, 0, 4)$ represents a weight 6 and length 10 gapped seed, and $(6, 2, 0, 1)$ represents a length 9 half gapped seed with 6 match positions and 2 half match positions in 'one neighbor' definition.

Before we move ahead into further discussion, we give another three important definitions based on [10], which are related to evaluating seeds in later comparisons.

Similarity: Consider a pair of fixed length sequences, the pair is said to be of similarity α if they have $\alpha\%$ of the positions share the same characters.

Sensitivity: The sensitivity of a seed is the probability of getting at least one hit in a pair of fixed length sequences of a certain similarity.

Efficiency: The efficiency of a seed is represented by the expected number of hits in a fixed length region.

3 Half Gapped Seeds vs Gapped Seeds

As stated in Section 1, one major problem of using "gapped seed" is the inflexibility in making tradeoff between sensitivity and efficiency. Consider the scenario where we cannot stand severe decrease in the efficiency, but meanwhile, we still want to get more sensitive outputs. Then, we will be in an awkward situation by using "gapped seeds". That is, if we decrease the weight of "gapped seed" to get better sensitivity, the large amount of loss in efficiency is unaffordable; on the other hand, if we keep its weight to guarantee the speed, it is impossible to increase the sensitivity.

Can we avoid such awkward situation by using "half gapped seeds"? By comparing the tradeoff abilities between "half gapped seeds" and "gapped seeds", this section gives a positive answer to the above question by comparing the sensitivities and the efficiencies of them. As for efficiency, the expected number of hits for gapped seeds can be computed based on Lemma 1 of [10]. For half gapped seeds, the expected number of hits can be computed using the following lemma.

Lemma 1 *Given a length M "half gapped seed" with W_1 half positions and W_2 match positions within a length L regions of similarity $0 \leq p \leq 1$, for 1 neighbor definition, the expected number of hits is $(L-M+1)(\frac{1}{3}(1-p))^{W_1}p^{W_2}$; for 2 neighbor definition, the expected number of hits is $(L-M+1)(\frac{2}{3}(1-p))^{W_1}p^{W_2}$.*

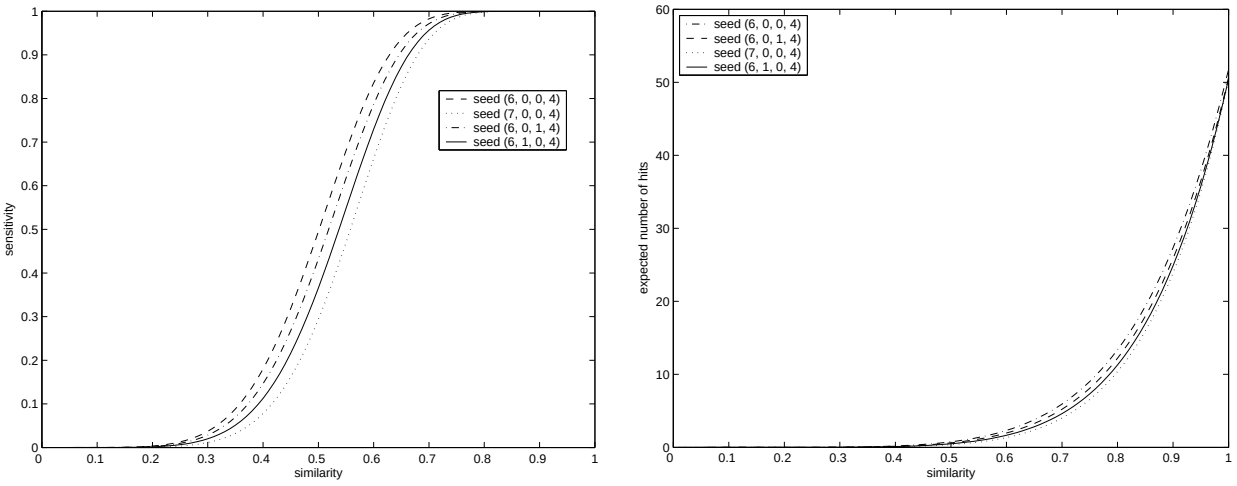
Proof: The expected number of hits in the sum of possibility that the seed fits substring in the region over $(L-M+1)$ possible positions. The possibility for every successful alignment is $(\frac{1}{3}(1-p))^{W_1}p^{W_2}$ for 1 *neighbor* definition and $(\frac{2}{3}(1-p))^{W_1}p^{W_2}$ for 2 *neighbor* definition. ■

We did extensive experiments to compare "half gapped seed" and "gapped seed". Our experiment is as follows. For all (s_1, s_2, s_3, s_4) seeds, that is, for all half gapped seeds with s_1 match positions, s_2 half match positions (one neighbor definition), s_3 half match positions (two neighbor definition), and s_4 don't care positions, we compute their sensitivity based on dynamic programming. By comparing their goodness, we can get the optimal (s_1, s_2, s_3, s_4) seed among all (s_1, s_2, s_3, s_4) seeds. For efficiency, according to Lemma 1, all (s_1, s_2, s_3, s_4) seed have the same efficiency and its value can be computed using Lemma 1.

We demonstrate that half gapped seeds can give a more flexible tradeoff between sensitivity and efficiency by Figures 1(a) and 1(b). The two graphs show the sensitivity and the efficiency of the optimal weight 6 gapped seed (optimal $(6,0,0,4)$ seed), the optimal $(6,0,1,4)$ seed, the optimal $(6,1,0,4)$ seed, and the optimal weight 7 gapped seed (optimal $(7,0,0,4)$ seed). Figure 1(a) shows that there is a gradually increase in sensitive for the four seeds in order and Figure 1(b) reveals their loss in efficiency in terms of expected number of hits accordingly. We also observe that there exists a big empty space between the optimal weight 6 and the optimal weight 7 gapped seeds for both sensitivity and efficiency. This means that gapped seeds give a big jump for both sensitivity and efficiency. Moreover, by having one (one-neighbor or two-neighbor) half gapped seed, we can already fill up the empty space between the two gapped seeds.

Based on the analysis of both "half gapped seeds" and "gapped seeds", we know that one can benefit from using "half gapped seeds" as they offer more flexible abilities in performing tradeoff between sensitivity and efficiency. "Half gapped seeds" are really useful when one want to increase the precision of the searching results while the hardware capacity cannot afford too much loss of efficiency.

In the next section, we will study some key parameters in "half gapped seeds" to show their effect on the sensitivity and efficiency tradeoff.



(a) Sensitivity

(b) Efficiency

Figure 1: Comparison on sensitivities and efficiencies between optimal weight 6,7 gapped seeds, optimal $(6, 1, 0, 4)$ seed, and optimal $(6, 0, 1, 4)$ seed.

4 Further Study of Half Gapped Seeds

Previous sections reveal the fact that "half gapped seeds" are more flexible than "gapped seeds" when performing tradeoff between sensitivity and efficiency. This section describes the key parameters in the "half gapped seeds" that affect the tradeoff. The study helps to give a fundamental idea of how to tune the tradeoff for the half gapped seed to suit the user requirement.

4.1 The Number of 'Half Match' Positions

If we fixed the number of match positions and 'don't care' positions for "half gapped seeds", what will happen if we change the number of half match positions? According to our experimental results, if we only change the number of half match positions, the more the half match positions are, the less sensitive the seed will be, and the more efficient it will become.

Table 1: The top three sensitivities for "half gapped seeds" differing only in the number of 'half match' positions when the similarity between query and database sequences is 0.6.

(6,0,0,1)	(6,0,1,1)	(6,0,2,1)	(6,0,3,1)
0.796263	0.782873	0.730733	0.679517
0.796263	0.782873	0.729445	0.679517
0.789812	0.782001	0.729445	0.677894

Table 1 shows that if we only increase the number of half match positions, the sensitivity will decrease in a certain degree. In this sense, we sacrifice the sensitivity of the seeds, so we should get some benefit in the efficiency. Let's see what happens to the expected number of hits for these seeds to verify this assumption.

Figure 3 plots the equations in Lemma 1 for the three half gapped seeds we used in Table 1. We find that, as we increase the number of half match positions, the efficiency of these seeds improve.

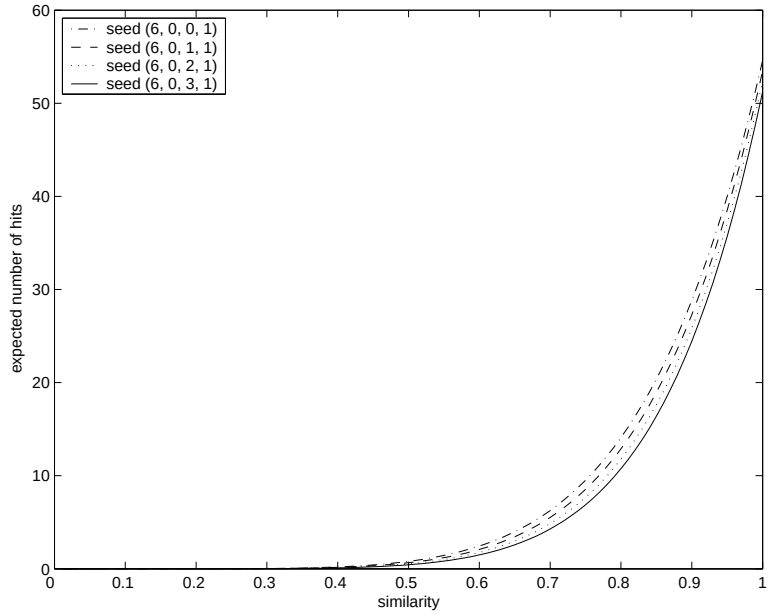


Figure 2: Comparison on the expected number of hits between "half gapped seeds" differing only in the number of 'half match' positions on 64-bits regions.

By Table 1 and Figure 2, it is clear that more half match positions make "half seeds" less sensitive but more efficient; while less half match positions make them more sensitive but more efficient.

4.2 The Definition of Neighbor Nucleotides

As we mentioned in Section 2, there are two different definitions for neighbor nucleotides in "half seeds": 'one neighbor' definition and 'two neighbor' definition. If we compare the "half seeds" that have the same number of half match positions, match positions and 'don't care' positions while use different neighbor nucleotide definitions, we will find they also vary on both sensitivity and efficiency. This property of "half seeds" shows another way of performing various tradeoffs between efficiency and sensitivity.

We conduct the experiments between the $(6, 0, 1, 1)$ half seed and the $(6, 1, 0, 1)$ half seed. Since one neighbor definition is more restricted, it is quite obvious that the two-neighbor definition one has better sensitivity, while the one-neighbor definition has higher efficiency. The experimental result agrees with our intuition.

Below table lists the top three most sensitive seeds for the above two seeds. Figure 3 shows the difference in their expected number of hits.

Table 2: The top three sensitivities for "half gapped seeds" using different neighbor nucleotides definition when the similarity of query and database sequence is 0.6.

(6,0,1,1)	(6,1,0,1)
0.782873	0.715385
0.782873	0.715385
0.782001	0.714139

These results imply that the 'two neighbor' definition can help the "half gapped seeds" to get

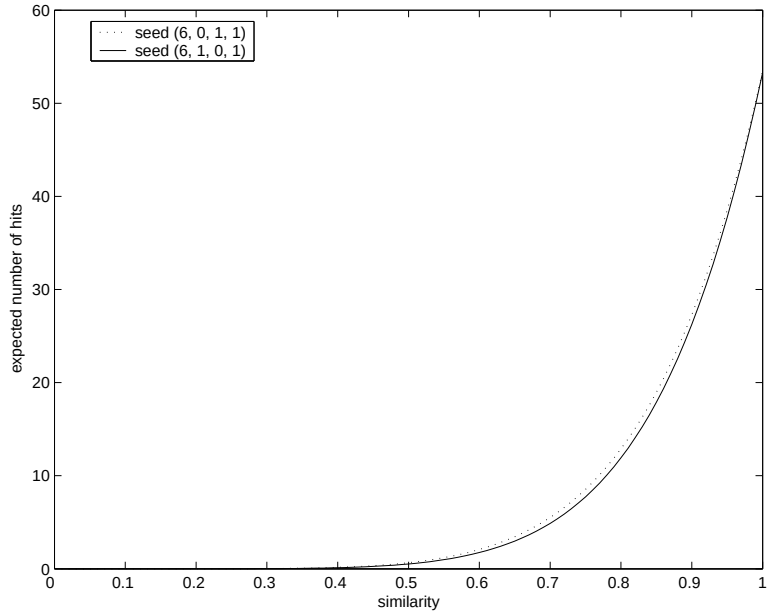


Figure 3: Comparison on the expected numbers of hits between "half gapped seeds" differing only in neighbor nucleotides definition on 64-bits regions.

better sensitivity, but it also reduce their efficiency; 'one neighbor' definition decreases the sensitivity of the "half gapped seeds", but it can improve their efficiency.

4.3 The Number of 'Don't Care' Positions

Besides the above two parameters, the number of 'don't care' positions in the "half gapped seed" also affects the result. In general, assume the parameters remain unchanged, when we increase the number of 'don't care' positions, the sensitivity of the seed will first increase to a maximum value, then the sensitivity decreases with the increasing of the number of 'don't care' positions. To analyze this parameter, we conduct some experiments on the "half gapped seeds" with the same number of half match positions and match positions, and the same neighbor nucleotides definition, but different number of 'don't care' positions. The result is as follow.

Table 3: The top three sensitivities for "half gapped seeds" having different number of 'don't care' positions when the similarity between query and database sequence is 0.6.

(6, 0, 1, 0)	(6, 0, 1, 1)	(6, 0, 1, 2)	(6, 0, 1, 3)	(6, 0, 1, 4)	(6, 0, 1, 5)
0.747137	0.782873	0.78908	0.794778	0.793832	0.791674
0.747137	0.782873	0.78908	0.794778	0.793832	0.791674
0.745968	0.782001	0.787612	0.793739	0.792491	0.791669

For efficiency, based on Lemma 1, when two seeds have the same number of match positions and half match positions, the efficiency improves as the number of 'don't care' positions in the seed increases.

In Table 3, we find that with the increase of 'don't care' positions from 0 to 5, the sensitivity of the "half gapped seeds" will first increase until it reaches the maximal value, and then it keeps decreasing. Hence, there exists a threshold, says α, so that when the number of 'don't care' positions

is smaller than α, the sensitivity of the "half gapped seed" always increases. After that, the sensitivity will decrease gradually. On the other hand, the efficiency of the "half gapped seeds" always get better and better with more 'don't care' positions. So, until the number of 'don't care' positions is bigger than α, this parameter takes effect in the tradeoff ability for the "half gapped seed", that is, increasing the number of 'don't care' positions can improve the efficiency while sacrificing on the sensitivity, and vice versa.

4.4 The Usage of the 3 Key Parameters

Till now we have analyzed three key parameters in the composition of "half gapped seeds", and have a clear picture of the effects when they are changed separately. But in real situations, different "half gapped seeds" are usually different in more than one parameters. What will happen when more than one of them change simultaneously? The results of the above analysis give us some hints that we can find some finer tuning for the tradeoff by change these three parameters together carefully. After thoroughly studying such situations, we are glad to find there exists a series of "half seeds" that can provide finer levels of tradeoff between two successive weighted optimal "gapped seeds".

Here we give an example. We listed four "half gapped seeds" that can provide different tradeoffs between sensitivity and speed compared with the weight 6 and 7 optimal "gapped seeds". These "half seeds" are limited by two constraints, first, their sensitivity must be better than weight 7 optimal "gapped seed"; second, their efficiency must be better than weight 6 optimal "gapped seeds". And one of the hidden constraint is that they must strictly obey the rule between each other that the more sensitivity it is, the less efficiency it is. We list the four "half gapped seeds" according to their sensitivities in decreasing order:

1. 1 0.5 1 0 0 0 1 1 0 1 1 in two neighbor nucleotides definition;
2. 1 1 0 1 1 0 0.5 0 1 0.5 1 in two neighbor nucleotides definition;
3. 1 1 0 0 1 0 1 0 0.5 1 1 in one neighbor nucleotides definition;
4. 1 1 0.5 1 0 1 0.5 0 0.5 1 1 in two neighbor nucleotides definition.

Besides sensitivities, their efficiencies are in increasing order. Now we present the result of comparison between these four seeds with optimal weight 6 and 7 "gapped seeds".

Figure 4 shows that the sensitivity of these four "half gapped seeds" are in between that of the two "gapped seeds", and Figure 5 shows the efficiencies of these four seeds change according to their sensitivities, that is the more sensitive, the less efficient. We can also find that the curves of each "half gapped seed" is still between the curves of "gapped seeds".

Above comparison reveals that our new seeds can provide higher flexibility in performing tradeoff compared with "gapped seeds". This property is especially useful when different requirements of applications are needed. So we can reach the conclusion that "half gapped seeds" are better than "gapped seeds" in performing tradeoffs between sensitivity and efficiency.

5 Conclusion

In this paper, we present a new seed called "half gapped seed". This seed introduces the novel concept of 'half match' positions into the formation of seeds. We find this new type of seeds can provide more flexible choices of tradeoffs between sensitivity and speed when compared with the "gapped seeds". We illustrate the usage of three key parameters in the composition of "half gapped seed". By changing one or more of them, we can provide finer level of tradeoffs than "gapped seeds". In some fields where tradeoffs are required, the "half gapped seeds" can always perform better than the "gapped seeds".

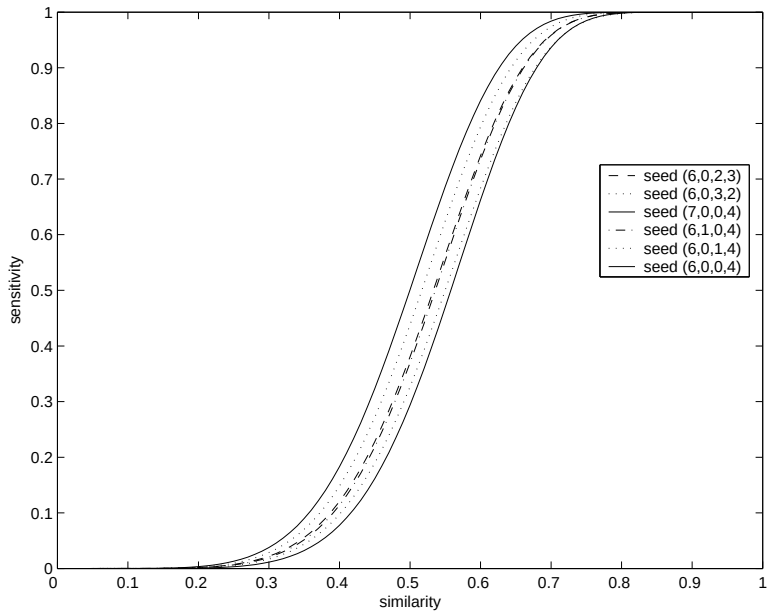


Figure 4: Comparison on sensitivities between the four listed "half gapped seeds" and the optimal weight 6 and 7 "gapped seeds" on 64-bits regions.

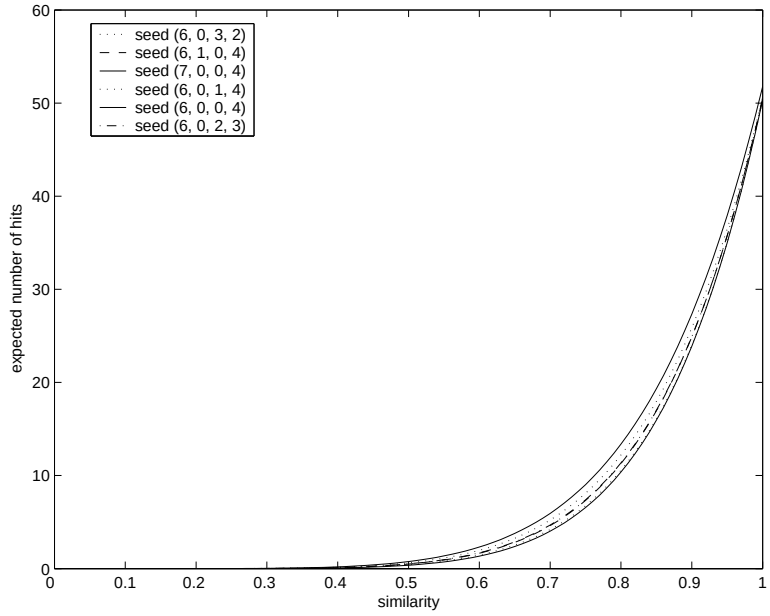


Figure 5: Comparison on efficiencies between the four listed "half gapped seeds" and the optimal weight 6 and 7 "gapped seeds" on 64-bits regions.

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E., and Lipman, D.J., Basic local alignment search tool, *J. Mol. Biol.*, 215:403–410, 1990.

[2] Altschul, S.F., Madden, T.L., Schuaffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J., Gapped blast and psi-blast: a new generation of protein database search programs, *Nucleic Acids Res.*, 25:3389–3402, 1997.

[3] Burkhardt, S., Crauser, A., Lenhof, H.P., Rivals, E., Ferragina, P., and Vingron, M., q-Gram based database searching using a suffix array, *Third Annual International Conference on Computational Molecular Biology*, Lyon, 11–14, 1999.

[4] Delchur, A.L., Kasif, S., Fleischumann, R.D., Peterson, J., White, O., and Salzberg, S.L., Alignment of whole genome, *Nucleic Acids Res.*, 27:2369–2376, 1999.

[5] Gish, W. Wu-Blast 2.0 Website: `http://blast.wustl.edu/`.

[6] Huang, X. and Miller, W., A time-efficient, linear-space local similarity algorithm, *Adv. Appl. Math*, 12:337–257, 1991.

[7] Kent, W.J., BLAT: the BLAST-like alignment tool, *Genome Research*, 12(4):656–664, 2002.

[8] Kurtz, S. and Schleiermacher, C., REPuter–fast computation of maximal repeats in complete genomes, *Bioinformatics*, 15:426–427, 1999.

[9] Lipman, D.J. and Pearson, W.R., Rapid and sensitive protein similarity searchers, *Science*, 227:1435–1441, 1985.

[10] Ma. B., John. T., and Li. M., PatternHunter: faster and more sensitive homology search, *Bioinformatics*, 18:440–445, 2002.

[11] Smith, T.F. and Waterman, M.S., Identification of common molecular subsequence, *J. Mol. Biol.*, 147:195–197, 1981.

[12] States, D. SENSEI website: `http://stateslab.wustl.edu/software/sensei/`.

[13] Tatusova, T.A. and Madden, T.L., Blast 2 sequence–a new tool for comparing protein and nucleotide sequences, *FEMS Microbiol. Lett.*, 174:247–250, 1999.

[14] Weiner, P., Linear pattern matching algorithms, *Switching and Automata Theory*, 1–11, 1973.

[15] Zhang, Z., Schwartz, S., Wagner, L., and Miller,W., A greedy algorithm for aligning DNA sequence, *J. Comput. Biol.*, 7:203–214, 2000.