Efficient Methods for Generating Optimal Single and Multiple Spaced Seeds

I-Hsuan Yang, Sheng-Ho Wang, Yang-Ho Chen and Pao-Hsian Huang
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106

Liang Ye and Xiaoqiu Huang
Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA

Kun-Mao Chao
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106

Abstract

Biologists highly rely on good algorithms for finding homologous regions in bimolecular sequences. An advanced homology search program named PatternHunter has recently been developed. Unlike the well-known program BLAST using a consecutive model, it utilizes a spaced seed model to attain higher sensitivity. We have developed a new program, which extends PatternHunter from a single spaced model to a multiple spaced model. In this paper, we describe methods for finding optimal single and multiple spaced models.

Keywords: *Sequence Alignment; Database Search.*

1. Introduction

BLAST family [1, 2, 10] find short continuous word matches called "seeds" between sequences and extend seeds into highly homologous regions. However, they miss homologous regions that do not contain any seed. An advanced homology search program named PatternHunter [8, 7] has recently been developed. The PatternHunter program finds short word matches under a spaced model. A spaced model is represented as a binary string of *length* l, where a 1 bit at a position means that a base match is required at the position and a 0 bit at a position means that either a base match or mismatch is acceptable at the position. The number of 1 bits in a spaced model is the *weight* of the model. For examples, the two words ACGTC and ATGAC form a word match under the spaced model 10101 of length 5 and weight 3. Because PatternHunter allows base mismatches in seeds, it is able to find more homologous regions than BLAST, which finds word matches under a consecutive model. A new version of BLASTZ also adapts the clever idea of spaced models [9].

A measure called *min_value*, to be defined in the next section, is used to evaluate a model. For a given pair of length l and weight w, say (l, w), a model of length l and weight w with the minimum *min_value* is an *optimal* model under (l, w).

Choi and Zhang [5] performed a sensitivity analysis of optimal spaced models and designed efficient heuristic algorithms for finding optimal spaced models. Brejova $et\ al.$ [3] studied the problem of computing optimal spaced models for detecting sequences generated by a Hidden Markov model. They also proposed an extension to spaced models that achieves substantial improvements in sensitivity and specificity [4]. In this paper, we present a new dynamic programming algorithm for finding an optimal model. Moreover, by analyzing optimal models under different pairs of length and weight, we observe a correlation between *min_value* and sensitivity.

Similar evaluation criteria are used to construct a set of spaced seeds, called *a multiple model*. A measure called *cross_min_value* is used to represent the *min_value* between any two models within a model set. The size of the potential search space for finding an optimal 2-model, denoted by (l, w)-(l, w), is $C_w^l \times C_w^l$ combinations. In general, the

Proceedings of the Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'04)
0-7695-2173-8/04 $ 20.00 © 2004 IEEE

size of the potential search space for finding an optimal k-model, denoted by $(l, w)^k$, is $\left(C_w^l\right)^k$ combinations. We present dynamic programming and hash-table preprocessing techniques to reduce the size of the search space.

2. Generating a good single model

2.1. Defining a good single model

The key point that makes PatternHunter more sensitive than BLAST is the model it uses. PatternHunter uses a spaced model, whereas BLAST uses a consecutive model. The spaced model shares fewer bases with any of its shifted copies than the consecutive one. We define the number of non-overlapping 1s between a model and its shifted copy as the number of 1s in the shifted copy that correspond to 0s in the model. If there are more non-overlapping 1s between the model and its shifted copy, then the probability of useless hit is lower[1], resulting in higher sensitivity (for detail descriptions, please refer to [6]). Thus, a sensitive model has a low sum of hit probabilities between the model and each of its shifted copies.

One might consider that collisions should be counted for C_2^l pairs among all shifted copies. However, a pair consisting of the first and the second copies, say 1-2, is equivalent to 2-3, 3-4, 4-5... And so as 1-3 is equivalent to 2-4, 3-5, 4-6... As a result, the hit probabilities of only $l-1$ pairs have to be computed. Therefore, for a model of length l, we define *min_value* as:

$$min\_value = \sum_{i=1}^{l} p^{N_i} \tag{1}$$

where N_i denotes the number of non-overlapping 1s between the model and its ith shifted copy.

The sensitivity of a model is related to its *min_values*. Observation 1 shows that the spaced model has a smaller *min_value* than the consecutive one. Observation 2 [6] shows that a sensitive model can be found by selecting a model of low *min_value*. Based on Observations 1 and 2, a model with the minimum *min_value* has the highest level of sensitivity in computing homologous regions.

Observation 1 *The spaced model has a lower min_value than the consecutive one.*

Observation 2 *The expected number of hits E of a model M, with length l and weight w, in a region R of length n and similarity p ,$0 \leq p \leq 1$, is the product of the probability of M hitting in R, called sensitivity S, and the min_value of M, named E'. That is, $E = S \times E'$ and $S = E/E'$.*

1 Under the condition that the original model hits, if any of its shifted copies also hits, the latter is a useless hit. This is because hits generated by the former and the latter are overlapped.

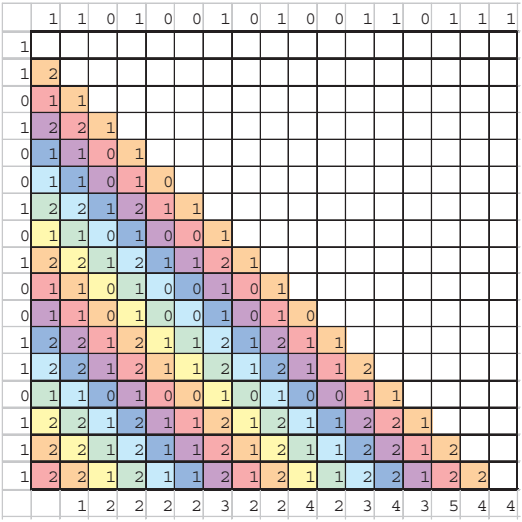

Figure 1. Overlapping 1s counting table. Take the uppermost diagonal for example. In that diagonal, there are four overlapping 1s, which are recorded as 2s in the table. The number of 2s is given in the last row of that diagonal.

2.2. Speeding up the computation

A model of length l has $l-1$ shifted copies. An overlapping 1s happens when a pair of copies has a 1 in the same column. In order to calculate the *min_value* of a model, we need to count the number of non-overlapping 1s between the model and each of its shifted copies, which can be computed by subtracting the number of overlapping 1s from the weight w.

Figure 1 uses the model 11010010100110111 to illustrate a straightforward way for counting the numbers of overlapping 1s between the model and each of its shifted copies. Take the uppermost diagonal for example. That diagonal corresponds to the copy that shifts one position to the right, i.e., 1st shifted copy. If a position has both 1s in its corresponding row and column, we record the number 2 which means there's an overlapping 1 in this position. The last row, denoted as $C_1, C_2, ..., C_l$, records the numbers of 2 in that diagonal. In other words, C_i is the number of overlapping 1s in the ith shifted copy. Therefore, *min_value* can be computed by the following equation.

$$min\_value = \sum_{i=1}^{l} p^{w-C_i} \tag{2}$$

A naïve method takes $O(l^2)$ time to fill in the table for one model. By observing that only $O(l)$ entries differ for two models with only one pair of 0 and 1 swapped, we

develop an $O(l)$ time algorithm for updating the table for a swapped model. Specifically, our method enumerates all models by swapping one pair of 0 and 1 at a time to keep most entries of the table unchanged, thus speed up the calculation. This method only needs to modify two rows and two columns of the table at a time. It reduces the time for a model's table calculation from $O(l^2)$ to $O(l)$.

Even though the method still has to exhaust all possible models, it is good enough for current practical applications. It generates an optimal (18,11) model in less than one second on a PC. It also generates an optimal model of length more than 30 in just a few minutes.

Besides the exact solution given above, we have also developed a heuristic method that takes polynomial time to find "good" models directly from shorter optimal models. It will be useful when a very long spaced seed is required.

3. Generating a good multiple model

Recently, we have developed a program that utilizes a set of single models, called *multiple models.* In a multiple model, the model hits if any single model in the set generates a "hit." This strategy can increase the sensitivity, however it increases the probability of random hits as well. In order to avoid random hits, weights of the models are increased. On the other hand, increasing the weights might decrease the sensitivity. Thus, the combination of models should be carefully selected. A set with good combination can increase the sensitivity and reduce random hits at the same time.

3.1. Defining a good multiple model

Basically, the principle of multiple models follows that of a single model — "The fewer bases shared by a model and any of its shifted copies, the higher its sensitivity is." In addition to *min_value*s stated above, we propose *cross_min_value*, which means the *min_value* between a pair of models. The best combination of models is the one with the minimum sum of all *min_value*s and all *cross_min_value*s within the set. In the following, this sum is called *total_min_value*, and the term *original_min_value* refers to the *min_value* of a single model.

3.1.1. 2-models Let us consider a 2-model first. For single models M_1 and M_2, there are two *cross_min_value*s and two *original_min_value*s. The *original_min_value* of the model M_1 is denoted by $M_1 \rightarrow M_1$, and the *cross_min_value* from M_1 to M_2 is denoted by $M_1 \rightarrow M_2$.

In Figure 2, the triangles represent the overlapping 1s counting tables as depicted in Figure 1. M_1's and M_2's *original_min_value*s are the lower triangles in the left two tables. By placing different models in both sides of the rightmost table, we have two *cross_min_value*s: $M_1 \rightarrow M_2$ in the upper triangle and $M_2 \rightarrow M_1$ in the lower triangle. The one with the minimum sum of *total_min_value*s, i.e., $((M_1 \rightarrow M_1)+(M_1 \rightarrow M_2)+(M_2 \rightarrow M_1)+(M_2 \rightarrow M_2))$ is considered as an optimal combination of a 2-model.

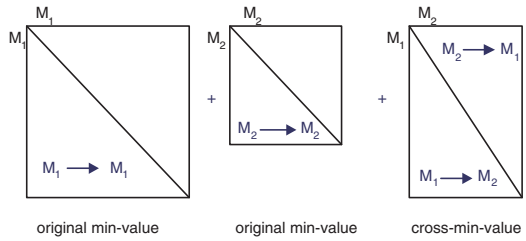

Figure 2. *total_min_value* = *original_min_value*s + *cross_min_value*s, the rectangles are the dynamic programming tables in Figure 1. Notice that M_1 and M_2 might be of different lengths.

3.1.2. k-models If there are k models in a set, there will be k *original_min_value*s and $2 \times C_2^k$ *cross_min_value*s. The optimal combination is the one with the minimum value defined in the function below.

$$total\_min\_values = \sum_{i=1}^{k}[(\sum_{j=i+1}^{k} M_i \leftrightarrow M_j) + M_i \rightarrow M_j] \quad (3)$$

The term $M_1 \leftrightarrow M_2$ denotes $(M_1 \rightarrow M_2) + (M_2 \rightarrow M_1)$, which again can be derived by dynamic programming. Note that the proportion of *original_min_value* over *total_min_value*s in k-model is $1/k$ which decreases as the number of models increases in a set. This observation implies that the cooperations among single models within a set become more important as the number of models increases.

3.2. Speeding up the computation

In finding a good multiple model, the computational complexity of exhausting all possible combinations is extremely high. Here we propose some strategies to speed up the computation.

3.2.1. Speeding up the computation for multiple models of the same length Let the symbol $(l, w)^k$ denotes a k-model where each individual single model is a (l, w) model. There are $(C_w^l)^k$ combinations, and each combination has k^2 *original_min_value*s and *cross_min_value*s, where each takes $O(l)$ time. Totally, it takes $O((C_w^l)^k \times k^2 \times l)$ time. By storing the information which have been computed in

advance, there are at most $C_k^{C_w^l} \times k$ steps to exhaust all possible combinations. Moreover, because all models have the same length in the set, using a preprocessing table that contains all of the *cross_min_value*s and optimizing caching speedups each step dramatically. Therefore, the time complexity can be reduced from $O((C_w^l)^k \times k^2 \times l)$ to $O(C_k^{C_w^l} \times k)$.

3.2.2. A heuristic method An alternative heuristic approach is merging a k_1-model and a k_2-model to form a (k_1+k_2)-model. In each merging step, we simplify the computation by setting a threshold which selects only good submodels.

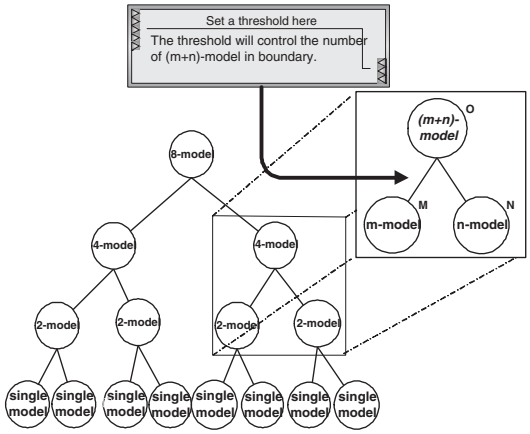

Figure 3. The merging process for an 8-model.

The process of merging k models is guided by a tree in a bottom-up manner, and yields the root as the resulting k-model set. The tree topology determines the overall merging process. Our tests show that higher trees tend to result in better models.

As Figure 3 shows, in the merging process, if node M contains *i* m-models, and node N contains *j* n-models, and the threshold is 1%, then there are $0.01 \times i \times j$ models in the node O.

After the merging process, the models in the root are further adjusted by a greedy algorithm which optimizes the models locally. Finally, our program uses the best local model as the final multiple model. Two 8-models are given in Tables 1 and 2.

3.2.3. An incremental approach Another approach is to grow a multiple model from an optimal single model. We start with an optimal 1-model of a given length and weight. In the i^{th} iteration, the single model which is the best complement to the current i-model is selected to form an $(i+1)$-model.

k	k-models	Total *min_value*s
8	1111010101110111 1111011010101111 1111100101101011 1 1110110011011011 1 1110101110001101 11 1110101100110011 11 1111001101001001111 1110110001100101111	94.250019

Table 1. An 8-model with lengths between 16 and 19, and weigth=12

k	k-models	Total *min_value*s
8	111001001010000100010110011 110100110000100100100010111 111010001001001010001000111 110011001010100000100101011 111010010000100011010001101 111000100100010100001101011 110101010000011001000011011 110100100011000001010100111	78.333077

Table 2. An 8-model with length=27 and weight=12.

Table 3 gives an incremental 8-model with length=19 and weight=12. Notice that the first row of this 8-model, i.e. 1111010011010100111, is an optimal single model.

4. Results

The *min_value*s of optimal models with different lengths and weights are showed in Figure 4. Models with length larger than 35 and weight above 19 are infeasible to exhaust. In that case, a heuristic method is applied to complete the statistics. Given a fixed weight, this figure is useful in determining which length yields the lowest *min_value*. Conversely, for a fixed length, it helps in determining which weight yields the lowest *min_value*.

1111010011010100111
1110101100010110111
1101100101011001111
1101110010100011111
1110011001101011011
1111000110100110111
1110101010011101011
1111001011001010111

Table 3. An incremental 8-model with length=19 and weight=12.

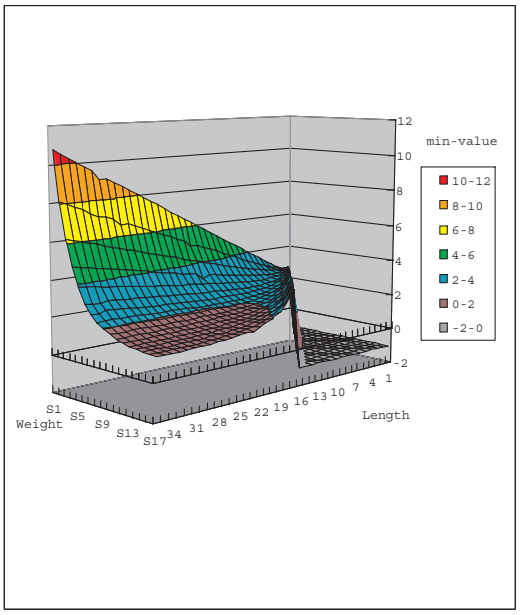

Figure 4. The *min_values* of models with different lengths and weights.

4.1. Simulation of single models

In order to evaluate the models found by the *min_value* rules, a simulation of sequence comparison by these models is done as follows. We randomly generate a region of length 64 composed of 0s and 1s with 70% similarity. The probability of each base to be 1 is 70%. Then we compare each optimal 1-model with the generated 64-length region for hits. Repeatedly doing this simulation a million times gives the statistic results given in Figure 5. The x coordinate is the length of every model and the y coordinate represents the model's sensitivity in this 70% similarity region. It implies that every model of different weight has its optimal length which is at the acme of the curve. Take weight=11 for example. Its optimal length is 18 and the corresponding optimal model is 111010010100110111, which is exactly the optimal model of weight 11 proposed by PatternHunter [8]. The optimal model of weight 12 in Figure 5 is the one of length 18, whereas by the *min_value* rules the optimal one's length is 22. This difference seems to be conflict at the first glance. However, this is due to the length of the simulated region is fixed at 64. By extending this region's length gradually, the optimal model's length of weight 12 in simulation approaches 22.

We have also examined some models under 65% and 75% similarity, and the results are very similar to the 70% similarity case.

4.2. Simulation of multiple models

The simulation of multiple models shows they also have an optimal length for each weight. For weigth=12, the optimal 8-model's sensitivity in the region of length 64 with 70% similarity is 0.723327, which is much higher than that of an optimal 1-model whose sensitivity is 0.357021.

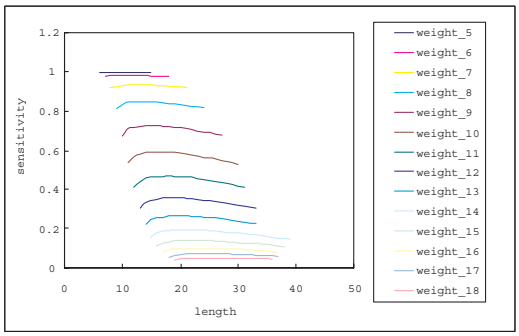

Figure 5. simulation on single models

4.3. A case study of weight 12

Models of smaller weights are not suitable for processing the large files. We have tested the models of smaller weights on the large files. But the results are not as expected. In theory, the models of smaller weights are more sensitive than the models of larger weights and should produce more HSPs. In practice, the program could not afford to look at too many word matches produced by models of smaller weights and hence ignored most of the word matches. Thus, models of smaller weights produce fewer HSPs than models of larger weights on the large files. This means that it is not appropriate to test models of different weights on the same pair of files. We need to use large files for large weights and small files for small weights. Because the program is designed for large files, results for weights 12 and 13 are more relevant in practice. Therefore, we decided to test models of weights 12 and 13 only. The two sequences used in the test are a human sequence (75 Mb), and a mouse sequence (15 Mb).

Figure 6 illustrates the growth curve of the minimum *min_values* for a single model of weight=12. The *min_values* go down till length=27, and then go up again.

On the other hand, Figure 7 gives the growth curve of the numbers of HSPs produced by models of weight=12 and various lengths. The highest peak is located around length 27, which matches the lowest point of Figure 6.

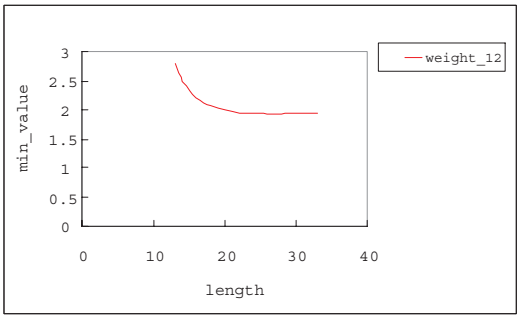

Figure 6. The x-coordinates are sorted by model's lengths, and the y-coordinates are the minimum *min_value* for a single model of weight=12.

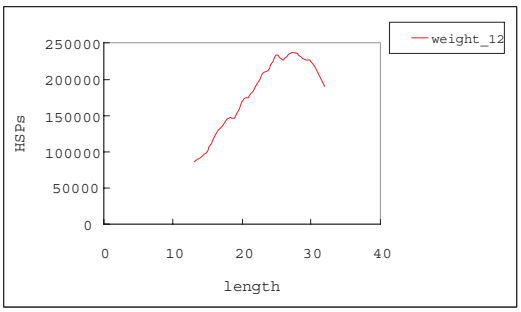

Figure 7. The x-coordinates are sorted by model's lengths, and the y-coordinates are the number of HSPs.

5. Discussion

By the definition of *min_values*, a model having many 1s in its front and rear tends to have a smaller *min_value* than other models with the same weight and length. Besides, a multiple model's *cross_min_value* is likely to be smaller if its individual single models are not similar to each other.

It remains open to find good models in more efficient ways, either in single or multiple models, by adjusting the dynamic programming table and applying the skills of the heuristic methods stated in this paper.

Another open problem is: given a fixed number of 0s and 1s, how to find the combination efficiently such that the resulting binary sequence has the minimum number of 1s collisions between its original sequence and all its shifted copies?

Acknowledgements

X.H. and L.Y. are supported by NIH grants R01 HG01502 and R01 HG01676, USA. K.C. is supported in part by an NSC grant 91-2213-E-002-129, Taiwan.

References

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

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

[3] B. Brejova, D. Brown, and T. Vinar. Optimal spaced seeds for hidden markov models, with application to homologous coding regions. In *The 14th Annual Symposium on Combinatorial Pattern Matching (CPM), 2676 volume of Lecture Notes in Computer Science*, pages 42–54, 2003.

[4] B. Brejova, D. Brown, and T. Vinar. Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. In *The 3rd International Workshop on Algorithms and Bioinformatics (WABI), 2812 volume of Lecture Notes in Bioinformatics*, pages 39–54, 2003.

[5] K. P. Choi and L. Zhang. Sensitivity analysis and efficient method for identifying optimal spaced seeds. *J. Comput. Syst. Sci.*, 2003. (to appear).

[6] M. Li. PatternHunter: Any Genome Anywhere. (Personal homepage).

[7] M. Li, B. Ma, D. Kisman, and J. Tromp. Patternhunter ii: Highly sensitive and fast homology search. *JBCB*, 2004. (to appear).

[8] B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.

[9] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignmenets with blastz. *Genome Research*, 13:103–107, 2003.

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