A Sensitive Sequence Comparison Method

I-Hsuan Yang[1], Sheng-Ho Wang[1], Yang-Ho Chen[1], Pao-Hsian Huang[1], Liang Ye[2], Kun-Mao Chao[1], Xiaoqiu Huang[2,3]

[1] Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan

[2] Department of Computer Science
Iowa State University
Ames, IA 50011-1040, USA

[3] Corresponding Author:

Xiaoqiu Huang
Department of Computer Science
Iowa State University
226 Atanasoff Hall
Ames, IA 50011-1040, USA
Email: xqhuang@cs.iastate.edu
Fax: (515) 294-0258

Key Words: Sequence Alignment, Database Search

ABSTRACT

Biologists highly rely on good algorithms to find the homologous regions in bimolecular sequences. One advanced homology search program, PatternHunter, has been developed in 2002. Unlike the well-known program Blast using a consecutive model, it benefited from gapped (nonconsecutive) model to work better. By observing and analyzing some significant properties of gapped-models, we propose a new program, which extends from using single gapped-model to multiple gapped models.

INTRODUCTION

Blast family [1, 2, 7],the well-known sequence analysis programs, use short continuous matches as "seeds" to extend and "grow" into highly homologous regions. However, they miss homogenous regions which have no successive matches longer than the model (seed) length. This is considered as lower sensitive.

PatternHunter [5], an advanced homology search program, has been developed in 2002. Instead of using a consecutive model, it benefits from a gapped model[1], which lets genome-sized sequence comparison on PC become possible.

In order to find the optimal model for a given pair of length l and weight w, say (l, w), the brute-force method is to exhaust all possible C_w^l combinations. And each combination takes $O(C_2^l \times l)$ to evaluate its *min_value* [2]. In this paper, we propose a new dynamic programming algorithm to speed up the computation of evaluation, which uses only $O(l)$ time. Using *min_value* as a benchmark, the optimal model of a fixed length and weight could be found. Moreover, by analyzing optimal models under different pairs of length and weight, some rules about *min_value* and sensitivity can be ovserved and are surely very helpful for improving the optimal models searching.

We also examined our novel idea—finding homologous patterns with multiple models. Similar benchmarks and rules are used to choose models to form a set of multiple models. Besides, another measurement, *cross_min_value*, is used to compute the *min_value* between any two models within a model set. In order to find the optimal 2-model, say (l, w)-(l, w), there will be $C_w^l \times C_w^l$ combinations to form a set. Also, if we want to find the optimal *k*-model, denoted by $(l, w)^k$, there will be $(C_w^l)^k$ combinations. In order to overcome this extra-high time complexity, some dynamic programming skills and preprocessing hash tables are used to speed up the computation.

METHODS

USE OF WORD MODELS

We describe an algorithm for computing high-scoring segment pairs (HSPs) between two files of sequences under a set of word models. The sequences in one file are called query

[1]Gapped models are represented as a binary string, where '1' means a match is required and '0' means don't care. The number of '1's is the *weight* of the gapped model.

[2]*Min_value* is the benchmark to evaluate a model. The smaller *min_value* means the better model. The definition will be given below.

sequences and those in the other file are called database sequences. The query and database sequences are concatenated together with a special boundary character inserted at each sequence boundary, where the query sequences are placed before the database sequences. The resulting sequence is called the combined sequence. Assume that the first position of the combined sequence starts at 1 instead of 0. Any word of the combined sequence with the special boundary character is ignored in the following description. Two positions of the combined sequence are equivalent under a word model of length k if the words of length k starting at the two positions form a match under the model. Two positions p_1 and p_2 of the combined sequence are equivalent if there is a word model in the set such that the two positions are equivalent under the model, or there is a position p_3 of the combined sequence such that p_1 and p_3 are equivalent and p_3 and p_2 are equivalent. Assume that each position is equivalent to itself.

The algorithm for finding HSPs between query and database sequences under the set of word models consists of two major steps. In step 1, compute sets of equivalent positions. In step 2, for each pair of equivalent query and database positions, if the pair of positions is not covered by any saved HSP, then extend a pair of words starting at the pair of positions into an HSP and save the HSP if its score is greater than a cutoff. Below we describe each step in detail.

Initially, each position of the combined sequence is a set by itself. Every word model wm in the set is considered to merge sets with equivalent positions under model wm as follows. Let w and k be the weight and length of the current model wm. The positions of the combined sequences are considered one at a time. Let p be the current position of the combined sequence and let v be the value of a word of length k starting at position p under model wm. We say that position p is of value v under model wm. To locate a set that contains previous positions of value v under wm, a word table T of size 4^w is constructed such that $T[v]$ is a position in a set that contains all previous positions of value v under wm. Initially, table T is set to zero at every entry to indicate that the set is empty. The following steps are performed for position p. If $T[v] = 0$, then set $T[v]$ to p. Otherwise, if the set containing position $T[v]$ is different from the set containing position p, then merge the two sets and set $T[v]$ to a position in the resulting set.

The sets of positions are kept in a data structure that supports Union-Find operations [6]. A set of positions is represented by a tree of nodes with each node corresponding to a position. Every child node points to its parent node. A position corresponding to the root node of the tree is used as the name of the set. The trees are implemented by an array Set of the size of the combined sequence. The array Set is set up as follows. Initially, for each position p, set $Set[p]$ to p. The initial setting indicates that each position is a set by

itself. To merge (or produce a union of) two sets with names s_1 and s_2, set $Set[s_1]$ to s_2. The name of a set containing a position p is found by computing a position s_i such that $s_1 = Set[p], s_2 = Set[s_1], ..., s_i = Set[s_{i-1}]$, and $s_i = Set[s_i]$. Two positions p_1 and p_2 are in the same set if and only if the name of a set containing a position p_1 is identical to the name of a set containing a position p_2. See ref [3] for more details on the Union-Find data structure.

A position array Pos of the size of the combined sequence is constructed to facilitate the generation of pairs of equivalent query and database positions. Every set of equivalent positions is processed as follows. Let $d_1, d_2, ..., d_i$ be an ordered list of database positions in the current set. Then set $Pos[d_1] = d_2$, $Pos[d_2] = d_3$, ..., $Pos[d_{i-1}] = d_i$, $Pos[d_i] = 0$. For each query position q in the current set, set $Pos[q] = d_1$. The array Pos is used to obtain a list of database positions equivalent to a given query position. The number of pairs of equivalent positions produced for a query position is the number of database positions in the set.

It is necessary to limit the number of database positions in every set for efficiency considerations. The construction of sets of equivalent positions is revised as follows. Let t be a cutoff on the number of database positions in a set for model wm. Let p be a position of value v under the current model wm. An additional requirement is placed on the merging of the set containing position $T[v]$ and the set containing position p. If the number of database positions in each set is at most t, then merge the two sets and set $T[v]$ to a position in the resulting set. Otherwise, if the number of database positions in the set containing position p is at most t, then set $T[v]$ to p.

SELECTING A GOOD SINGLE MODEL

How to find a good model

The key point that makes PatternHunter superior to Blast is the model it uses. PatternHunter uses nonconsecutive (gapped) models whereas Blast uses consecutive models. The gapped model shares fewer bases with any of its shifted copies than the consecutive one. As Table 1 shows, if there are more non-overlap 1s, then there will be less probability of useless hit[3], which will cause higher sensitivity (for detail descriptions, please refer to [4]). Thus, the method to enhance a model's sensitivity is to minimize the hit probability between all its

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

shifted copies.

One might consider that collisions should be counted for C_2^l pairs amount all shifted copies. However, the pair consists 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, hit probabilities of only $l-1$ pairs are needed to be counted. Therefore, we can define *min_value* as follows:

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

Assume N_i denotes the non overlap 1s of the ith shifted copy, and l is the model length.

Table 1: model's property

a model and any its shifted copies	non overlap 1s (N_i)	hit probability
1110100101		1
1110100101	4	p^4
1110100101	3	p^3
1110100101	4	p^4
1110100101	5	p^5
.....	...	...

The value of p represents the hit probability of every base *i.e.*, similarity. Under the condition of the first comparison occurring hit, the hit probability of the following comparisons is shown as the hit probability column above.

Speed up finding good models

To find the optimal model, the *min_value* of all candidate models with different length l and weight w should be evaluated. A model with length l has $l-1$ shifted copies. A collision happens when any one pair of copies have '1' in the same column (Table 1). In order to calculate the *min_value*s, collisions between two copies of a model should be counted.

Figure 1 is designed for counting the collision numbers between different copies. Take the model 11010010100110111 for example. Since collisions happen when there are '1's in the same column, we use '2' to represent it and modify the table. The last row (denoted as $C_1, C_2, ..., C_l$) which records the numbers of '2' in that diagonal is used to calculate the

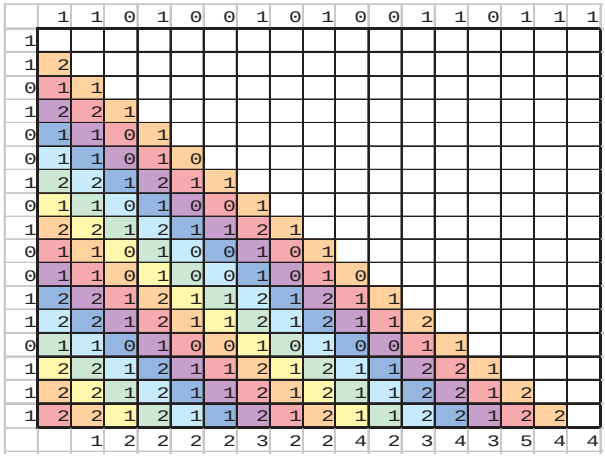

Figure 1: collision counting table

*min_value*s. The function below is the *min_value*, and *p* means the similarity.

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

Filling the lower triangular table, the collisions of all pairs of shifted copies are done. Naive method may need $O(l^2)$ time to fill a table for one model evaluation. However, we carefully enumerate all models by swapping one pair of 0 and 1 at a time to keep most of the table used unchanged, therefore simplify the calculation. This method needs only modifying two rows and two columns of the table at a time. It reduces the table filling of one model from $O(l^2)$ into $O(l)$, which speed up the scanning of all model considerably.

Even though the method still has to exhaust all possible models, this graceful brute-force algorithm is good enough for the practical usage. It could generate the optimal (18,11) model mentioned in PatternHunter paper less than one second on modern PC. It could also generate optimal model with length more than 30 in few minutes. Because of the hash table size (w^4 of DNA sequence) limitation, those models are large enough in practice.

There is also a heuristic method which takes polynomial time to find ”good” models directly from shorter optimal models. Within a practically countable range, this method generates models more efficiently and precisely.

SELECTING A GOOD MULTIPLE MODELS SET

Besides single models which PatternHunter uses, our program uses a set of models, called *multiple models*. Within multiple models, if any one model in the set generates a “hit”,

then say that the set hits. This strategy can increase the sensitivity, however it increases the probability of random hits as well. In order to avoid random hits, weight of the models are increased. On the other hand, increasing the weight 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.

How to find a good multiple models set

Basically, the rules of multiple models come from that of single model including the main idea — "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 proposed *cross_min_value*, which means the *min_value* between two models. The best combination of models can be chosen by the one which has minimum summation of all *min_value*s and all *cross_min_value*s within the set. In this paper, this summation will be called *total_min_value*s and *min_value*s will be called *original_min_value*s to distinguish from *cross_min_value*s.

2-model

Using models M_1 and M_2 as a 2-model, there will be two *cross_min_value*s and two *original_min_value*s. The symbol $M_1 \rightarrow M_2$ denotes the *cross_min_value* M_1 to M_2(directive).

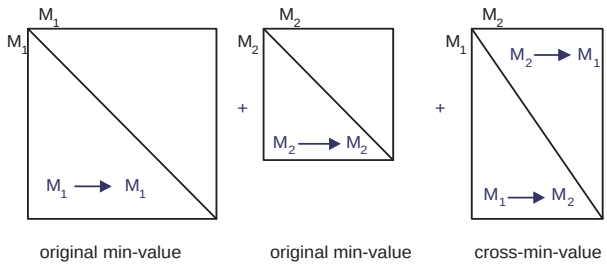

Figure 2: *original_min_value*s and *cross_min_value*s, the rectangles are the DP table of Figure 1. M_1 and M_2 might be of different length.

As Figure 2 shows, M_1's and M_2's *original_min_value*s come from the lower triangular dynamic programming table. By placing different models at each side of this table, say model M_1 and model M_2, there will be two *cross_min_value*s: $M_1 \rightarrow M_2$ at the upper triangular matrix and $M_2 \rightarrow M_1$ at the lower triangular matrix. Therefore, the one with minimum summation of *total_min_value*s ($M_1 \rightarrow M_1 + M_1 \rightarrow M_2 + M_2 \rightarrow M_1 + M_2 \rightarrow M_2$) is the optimal combination of 2-model.

k-model

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 comes from the one which has minimal 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 symbol $M_1 \leftrightarrow M_2$ means $(M_1 \rightarrow M_2) + (M_2 \rightarrow M_1)$, which can be derived from a single dynamic programming table simultaneously (The third table of Figure 2). 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 importance of cooperation between models within a set grows as the size of model set increases.

Speed up finding good multiple models sets

In finding a good set of multiple models, the computation load of exhausting all possible combinations is extremely high. There are some tricks to speed up the computation.

Speed up with a same length in a set Let the symbol $(l, w)^k$ denotes taking k (l, w) models as a set. There will be $(C_w^l)^k$ combinations, and each combination has k^2 *original_min_value*s and *cross_min_value*s, with each takes $O(l)$ time to be obtained from DP table. Totally, it takes $O((C_w^l)^k \times k^2 \times l)$. By using another dynamic programming skill to store the information which have been computed in advance, there will be utmost $C_k^{C_w^l} \times k$ steps to exhaust all possible combinations. Moreover, because all models have the same length in the set, using a big preprocessing table that contains all of the *cross_min_value*s and optimizing caching will efficiently speedup each step. Therefore, the time complexity can be reduced from $O((C_w^l)^k \times k^2 \times l)$ to $O(C_k^{C_w^l} \times k)$. However, this time complexity is still in exponential time.

Heuristic method Due to time limitation, it is impossible to exhaust all combinations for 8-model with different lengths. One way to approximate is by merging one k_1-model and one k_2-model to form a $(k_1 + k_2)$-model. In each step of merging, there will be a threshold which will select only good sub-models, thus reduces our computation and keeps good results.

The process of merging k models will construct a tree by bottom-up approach and makes the root to be *k*-model set. Also, any tree that contains k leaves can decide a process of merging k models to be a *k*-model set. Different trees structures may cause different

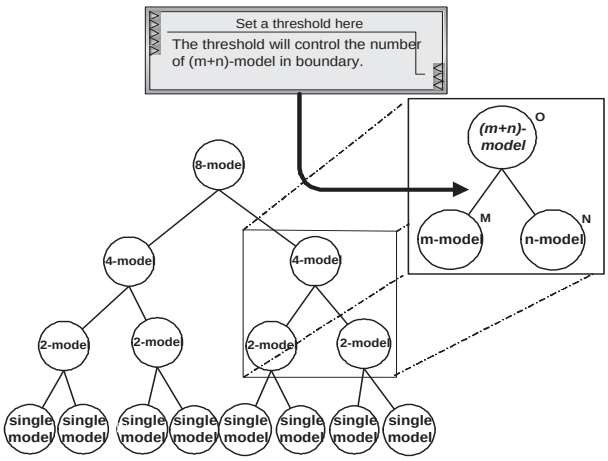

Figure 3: The merging process of 8-model

Table 2: 8-model with length between 16 and 19, weigth=12

k	*k*-models	Total *min_value*s
8	1111010101110111	94.250019
	1111011010101111	
	11111001011010111	
	11101100110110111	
	1110101110001101111	
	1110101100110011111	
	11110011010010011111	
	11101100011001011111	

results of the root. By our experience, the higher the tree is, the better models can be obtained.

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

After merging, the models in the root can also be adjusted by a greedy algorithm which will replace the models to become a local optimal. Finally, our program uses the best local optimal model set as the word models.

RESULTS

Table 3: 8-model with length=27 weight=12

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

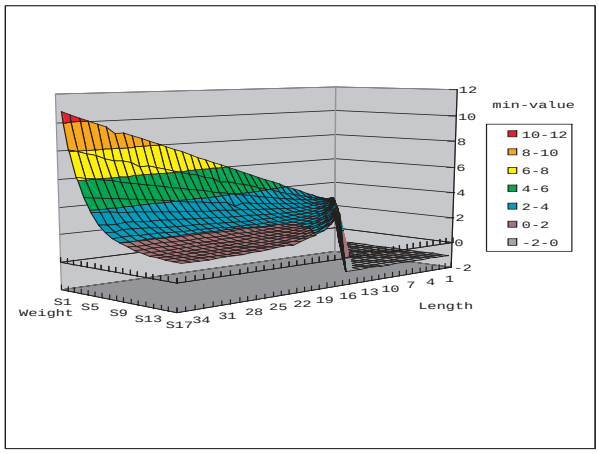

Figure 4: the *min_value*s of models with different length and weight.

OBSERVATION IN SINGLE MODELS

The *min_value*s of optimal models with different length and weight are showed in Figure 4. Models with length larger than 35 and weight around 19 are difficult to exhaust where heuristic method is applied to complete this statistic graph. (The relative error is below 0.01.) By observing those optimal *min_value*s of different lengths with the same weight, there exists a lowest *min_value* which length corresponds to our outcome of simulations (will be stated next section). A similar observation can be obtained in the row, too. With the same length, there is also a lowest point in the curve meaning the best weight. These curves imply that a optimal length can be found by given a fixed weight, and vice versa. This important observation is very helpful to extend our models.

OBSERVATION IN MULTIPLE MODELS

Our program will use 8 models to form a set to compare sequences. The lengths of models in a set might be different, but the weight should always keep the same to make hit extension handling easier.

Simulations

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 proceeding sequence comparison between every model and this 64-length region. Repeatedly doing this simulation millions of times can find the statistic results which are plotted 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. Figure 5 displays the relation of model's length and its sensitivity. It implies that every model of different weight has its optimal length which is at the acme of the curve. Taking weight_11 for example, its optimal length is 18 and this optimal model is 111010010100110111. The result is exactly the optimal model of weight 11 proposed in PatternHunter. From Figure 5 and the *min_value* rules referred previously, there is a little difference in determining the optimal model on the fixed weight. The optimal model of weight 12 on Figure 5 is the one of length 18, but 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 compared region is fixed at 64. By extending this region's length gradually, the optimal model's length of weight 12 on simulation would approach 22.

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 exepcted. 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 files used in the test are of 75 Mb and 15 Mb.

Simulation of multiple models

The multiple models simulation, which is similar to that on single model, is provided on multiple models' evaluation. The result shows multiple models also have an optimal length for each weight. Fixing the weight to be 12, the optimal 8-model's sensitivity on the region of length 64 with 70% similarity is 0.723327 which is much higher than that of single one which is 0.357021. And experiments on Mus musculus DNA, more highly similar segments (also called HSPs which is referred in PatternHunter) are found by using these better models.

Therefore, suitability applying these better models on sequenced comparison would make the comparison more efficiently and accurately.

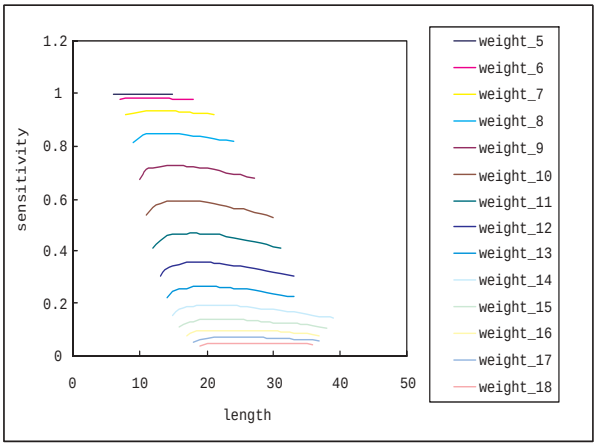

Figure 5: simulation on single models

EXAMINATIONS ON *Mus Musculus* DNA

DISCUSSION

By observing the definition of *min_value*, a model having many 1s at its front and rear part will have a smaller *min_value* than other models with same weight and length. Besides,

cross_min_value will be smaller if those two models are more "dislike". These observations may be helpful to improve the approximation.

There are a few ways to improve the computation of the *min_value*s. However, exhausting all combinations of '0' and '1' in multiple models is still inevitable. There might be a way to find the best model in polynomial time, either in single or multiple models, by adjusting the dynamic programming table and improving the skill of heuristic method stated above.

Another open problem is: given a fixed number of 0s and 1s, how to find the combination such that this binary sequence will have the minimum 1s collision between its original sequence and all its shifted sequence in polynomial time?

In this paper, all of the discussions are based on 70% similarity. We also have examined some models under 65% and 75% similarity, and the results are very similar to the 70% similarity environment. In the next version of our prgram, there might be selective model sets that can suit any similarity by user defined.

ACKNOWLEDGEMENTS

L.Y. and X.H. are supported by NIH grants R01 HG01502 and R01 HG01676. K.M. is supported in part by grant NSC 91-2213-E-002-129 from the National Science Council, 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] H. N. GABOW and R. TARJAN. A linear-time algorithm for a special case of disjoint set union. *J. Compu . Svst. Scl. 30, 2*, April, 1985.

[4] M. Li. PatternHunter: Any Genome Anywhere. website: `http://www.cs.ucsb.edu/~mli/phunter.ppt`.

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

[6] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, v.22 n.2 April, 1975.

[7] 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.

FIGURE LEGENDS