GHOSTX: An Improved Sequence Homology Search Algorithm Using a Query Suffix Array and a Database Suffix Array

Shuji Suzuki, Masanori Kakuta, Takashi Ishida, Yutaka Akiyama*

Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Meguro-ku, Tokyo, Japan

Abstract

DNA sequences are translated into protein coding sequences and then further assigned to protein families in metagenomic analyses, because of the need for sensitivity. However, huge amounts of sequence data create the problem that even general homology search analyses using BLAST become difficult in terms of computational cost. We designed a new homology search algorithm that finds seeds sequences based on the suffix arrays of a query and a database, and have implemented it as GHOSTX. GHOSTX achieved approximately 131–165 times acceleration over a BLASTX search at similar levels of sensitivity. GHOSTX is distributed under the BSD 2-clause license and is available for download at http://www.bi.cs.titech.ac.jp/ghostx/. Currently, sequencing technology continues to improve, and sequences are increasingly producing larger and larger quantities of data. This explosion of sequence data makes computational analysis with contemporary tools more difficult. We offer this tool as a potential solution to this problem.

Citation: Suzuki S, Kakuta M, Ishida T, Akiyama Y (2014) GHOSTX: An Improved Sequence Homology Search Algorithm Using a Query Suffix Array and a Database Suffix Array. PLoS ONE 9(8): e103833. doi:10.1371/journal.pone.0103833

Editor: Ahmed Moustafa, American University in Cairo, Egypt

Received March 11, 2014; Accepted July 7, 2014; Published August 6, 2014

Copyright: © 2014 Suzuki et al. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability: The authors confirm that all data underlying the findings are fully available without restriction. SRR444039 is available from the Data Analysis and Coordination Center for Human Microbiome Project Web site (http://www.hmpdacc.org/). SRX444039 is available from the Sequence Read Archive.

Materials and Methods

Overview of the GHOSTX algorithm

GHOSTX consists of three main steps: a seed search, an ungapped extension, and a gapped extension. The flow of GHOSTX is shown in Figure 1. Initially, GHOSTX finds seeds that are substrings of database sequences similar to the substrings of a query sequence. Next, GHOSTX makes alignments by extending those seeds without gaps, and then similar, nearby seeds are brought together by a chain filter. Finally, GHOSTX makes alignments from seeds with gaps. The gapped extension step
requirements heavy calculation, but the BLAST algorithm efficiently decreases the number of gapped extension candidates through its seed search and ungapped extension steps. As a result, the seed search and the ungapped extension steps are the most computationally intensive parts of BLAST. The seed search and the ungapped extension steps consume approximately 75% of the computation time of BLAST, while approximately 20% of the time is spent on the gapped extension [9]. Thus, reducing the computation time for the seed search and ungapped extension steps is effective for achieving acceleration. To accelerate the search seed step, GHOSTX uses suffix arrays for both the query and the database sequences.

Figure 1. The flow of GHOSTX.

Our seed search method using a suffix array effectively reduces the computation time of the seed search step. As a result, the ungapped extension step then becomes the bottleneck. Thus, for further acceleration, we have to decrease the number of ungapped extension candidates by using longer seeds. However, if this is done, significant matches can be missed, and search sensitivity becomes lower. Consequently, a sophisticated method is required for accelerating search speed, while still maintaining search sensitivity. Therefore, GHOSTX does not fix the length of a seed in the seed search step, but rather it extends the length until the matching score exceeds a given threshold. In comparison, BLAST searches with seeds of fixed lengths, and if one seed is discovered near another, BLAST performs ungapped extensions around it. BLAST seed hits with low matching scores using fixed length seeds, such as an exact match of "AAA," whose score is only 12 based on the BLOSUM62 score matrix, are treated equally with seed hits with high matching scores, such as an exact match of "WWW," whose score is 33. However, hits with lower scores tend to be false. Consequently, GHOSTX extends such seeds to check whether they are reliable, thus GHOSTX can use a higher score threshold than BLAST, without losing its search sensitivity. As a result, GHOSTX can reduce the number of ungapped extensions and gapped extensions needed, thereby reducing computation time after the initial seed search step.

Suffix Array

A suffix array is the list of indexes of all suffixes of a string in a lexicographically sorted order. A suffix array can be constructed in linear time. A text $T[0, n] = t_0 ... t_n$ is a sequence of symbols and the length of $T$ is $|T| = n$. Each symbol is an element of an alphabet $\Sigma$ (e.g., a protein is 20). $T[i]$ is $t_i$ and $T[i, i+j] = t_i ... t_{i+j-1}$ are substrings. The suffix array of $T$ is $S_A$, that is, an array of pointers to all the suffixes of $T$ in lexicographical order. Therefore, if $i < j$, then $T[S_A[i]] < T[S_A[j]]$. An exact search based on a binary search for pattern, whose length is $m$, can be performed as $O(\log n)$ with the suffix array of $T$.

Seed Search

For two suffix arrays, we can find all the local matches using dynamic programming [10]. However, calculating all alignments using dynamic programming requires a huge amount of computation time. In GHOSTX, therefore, we introduce two methods to prune the search space.

Here, the sequences $S_0, S_1, ..., S_N$ in a database are connected with inserting delimiters to transform them into a long single sequence $S_{db} = S_0\#S_1\#...\#S_N$ (marked by the special symbol #). $S_A_{db}$ is the suffix array of $S_{db}$, and $S_A_q$ is the sequence of query $S_q$.

The pair of substrings $S_{db}i$ and $S_{q}j$, $S_{db}[i, i+l]$, and $S_{q}[j, j+l]$ is the seed. Here, we want to find a seed whose score is more than the threshold $T_{seed}$ based on these two suffix arrays. Figure 2 shows the pseudo-code of the seed search method, and Figure 3 shows a pseudo-code for the search method of one character using a suffix array. In Figure 2, $q_p$, $e_p$, $q_a$, and $e_a$ are positions on $S_A_q$ and $S_A_{db}$, and GHOSTX gets the positions of substrings from suffix arrays by using these positions. If the score of a pair of substrings $(X_{db}, X_q)$ exceeds threshold $T_{seed}$, GHOSTX keeps the pair as a seed (line 22 in Figure 2); otherwise, GHOSTX checks all pairs of extended substrings $(X_{db}c, X_qc^')$ (c and $c'$ are members of $\Sigma$) (line 25 in Figure 2). Thus, the maximum number of new pairs of substrings is $|\Sigma|$. Using the suffix arrays of a query and a database, GHOSTX can find a substring efficiently. Figure 4

Table 1. Computation time with SRS011098 and KEGG GENES (3.9 GB).

| Program          | Computation time (sec.) | Acceleration ratio |
|------------------|-------------------------|--------------------|
| GHOSTX           | 401.9                   | 152.6              |
| RAPSearch        | 649.5                   | 94.4               |
| RAPSearch in fast mode | 91.2                   | 672.2              |
| BLAST            | 1409.7                  | 43.5               |
| BLAST            | 61314.1                 | 1.0                |

The first, second, and third columns show the name of each program, the computation time, and the acceleration in processing speed relative to BLASTX using 1 thread, respectively.

doi:10.1371/journal.pone.0103833.t001
shows the example for the seed search. If \{A, A\} is found, GHOSTX searches the query sequence and the sequences in a database for extended substrings AA, AR, ..., AV. And then, GHOSTX checks all pairs of extended substrings that are found \{AA, AA\}, \{AA, AR\}, ..., \{AV, AV\}. GHOSTX repeats this step. However, the search takes a long time if the max seed length \(\text{length}_{\text{max}}\) is large, because the size of the seed search space is \(O(2^{\text{length}_{\text{max}}})\). Thus, the search space must be pruned.

GHOSTX uses two methods to prune the search space (line 24 in Figure 2). First, let \(\text{score}_{\text{max}}\) be the sum of the exact match score of all query substring characters (line 16 in Figure 2), \(\text{score}\) be the score of the pair of the query and database substring (line 20 in Figure 2), and \(D\) be the upper limit of \(\text{score}_{\text{max}} - \text{score}\). If \(\text{score} \leq \text{score}_{\text{max}} - D\), GHOSTX does not extend the substring in the pair. For example, if GHOSTX checks \{AA, AR\} and uses BLOSUM62 score matrix, \(\text{score}_{\text{max}}\) of this pair is \(4 + 5 = 9\) and \(\text{score}\) of this pair is \(4 - 1 = 3\). If \(D = 4\). In this case, GHOSTX does not extend the substrings in this pair. Second, if the score of a substring pair is not more than 0, GHOSTX does not extend it.

Table 2. Computation time with SRR444039 and KEGG GENES (3.9 GB).

|                       | Computation time (sec.) | Acceleration ratio |
|-----------------------|-------------------------|--------------------|
| GHOSTX                | 362.7                   | 151.8              |
| RAPSearch             | 553.2                   | 99.5               |
| RAPSearch in fast mode| 64.8                    | 849.6              |
| BLAT                  | 1265.3                  | 43.5               |
| BLAST                 | 55045.0                 | 1.0                |

The first, second, and third columns show the name of each program, the computation time, and the acceleration in processing speed relative to BLASTX using 1 thread, respectively.

doi:10.1371/journal.pone.0103833.t002

Figure 2. Search algorithm using suffix arrays.
doi:10.1371/journal.pone.0103833.g002

Algorithm 2 Search by using suffix array

```plaintext
SASearchNextCharacter(T; SA, sp, ep, c, length)
1: \(p_{\text{tmp}} \leftarrow sp - 1\)
2: \(c_{\text{tmp}} \leftarrow c\)
3: while \(p_{\text{tmp}} < ep\) do
4: \(m \leftarrow \frac{\text{score}_{\text{max}} - \text{score}}{2}\)
5: if \(|T| - \text{length} + \text{SA}[m] < c\) then
6: \(p_{\text{tmp}} \leftarrow m\)
7: else
8: \(p_{\text{tmp}} \leftarrow m\)
9: end if
10: end while
11: if \(T[|T| - \text{length} + \text{SA}[p_{\text{tmp}}]] \neq c\) then
12: \(p_{\text{tmp}} \leftarrow 1\)
13: \(c_{\text{tmp}} \leftarrow 0\)
14: return \(p_{\text{tmp}}, c_{\text{tmp}}\)
15: end if
16: \(sp \leftarrow p_{\text{tmp}}\)
17: \(c_{\text{tmp}} \leftarrow c\)
18: while \(c_{\text{tmp}} < c\) do
19: \(m \leftarrow \frac{\text{score}_{\text{max}} - \text{score}}{2}\)
20: if \(|T| - \text{length} + \text{SA}[m] > c\) then
21: \(c_{\text{tmp}} \leftarrow m\)
22: end if
23: \(c_{\text{tmp}} \leftarrow m\)
24: end if
25: end while
26: if \(T[|T| - \text{length} + \text{SA}[p_{\text{tmp}}]] \neq c\) then
27: \(p_{\text{tmp}} \leftarrow 1\)
28: \(c_{\text{tmp}} \leftarrow 0\)
29: return \(p_{\text{tmp}}, c_{\text{tmp}}\)
30: end if
31: \(ep \leftarrow p_{\text{tmp}}\)
32: return \(sp, ep\)
```

Figure 3. Search algorithm using a suffix array.
doi:10.1371/journal.pone.0103833.g003
x<y<z, the score of the substring pair \( \{S_{db}[i, i+y], S_q[j, j+y]\} \) is less than 0, and the score of the substring pair \( \{S_{db}[i, i+z], S_q[j, j+z]\} \) exceeds the threshold \( T_{seed} \). Then GHOSTX finds another pair \( \{S_{db}[i, i+x], S_q[j, j+y]\} \) whose score exceeds \( T_{seed} \). Therefore, GHOSTX examines only those pairs with scores greater than 0. The number of search candidate substrings drastically decreases as they become longer. For example, if GHOSTX checks \( \{A, R\} \) and uses the BLOSUM62 score matrix, the score of this pair is −1. Therefore, GHOSTX does not extend the substrings in this pair. Consequently, GHOSTX can find long seeds quickly using these pruning methods. In addition, GHOSTX uses a depth-first search for the implementation of this algorithm to save memory. With a breadth-first search, the depth of the recursion in a seed search is proportional to the exponential of \( \text{length}_{\text{max}} \), and thus it is difficult to check all pairs of substrings. However, the depth of recursion in SeedSearchCore is \( O(\text{length}_{\text{max}} \times \Sigma^2) \) based on a depth-first search. Therefore, using this depth first search strategy can save memory.

Even when using a binary search, this seed search approach was originally a bottleneck in GHOSTX. To accelerate the process GHOSTX searches parts of seeds using an auxiliary data structure. GHOSTX stores the search results for all substrings whose length is less than 6 on a table before the database search. This process is performed only once, similar to the construction of the database index. In the seed search, GHOSTX can find the search result for a substring without performing a binary search on the suffix array of a database, if the length of the substring is shorter than 6. If we store the search results for longer substrings, we can make the process more efficient. However, the table requires more memory depending on the length of the substring. If the length \( \text{length}_{\text{substring}} \) of a substring is extended by 1, the size of the table increases by \( O(\text{length}_{\text{substring}}) \). Thus, GHOSTX only stores the search results for the substring whose length is less than 6.

**Table 3.** Computation time with SRS011098 and NCBI nr (14.8 GB).

| Program       | Computation time (sec.) | Acceleration ratio |
|---------------|-------------------------|--------------------|
| GHOSTX        | 1020.1                  | 165.2              |
| RAPSearch     | 1564.4                  | 107.7              |
| RAPSearch in fast mode | 223.8                  | 752.8              |
| BLAT          | N/A                     | N/A                |
| BLAST         | 168488.0                | 1.0                |

The first, second, and third columns show the name of each program, the computation time, and the acceleration in processing speed relative to BLASTX using 1 thread, respectively.

GHOSTX: An Improved Sequence Homology Search Algorithm

### Ungapped Extension and Chain Filtering

Decreasing the number of seeds is critical for the acceleration of a search. However, higher \( T_{seed} \) values cause an increase in the number of significant hits missed, so it is difficult to use high \( T_{seed} \) values without sacrificing sensitivity. Therefore, GHOSTX performs an ungapped extension, which extends seeds without any gaps and excludes low-score extended seeds, after the seed search step, as in BLAST. In the ungapped extension step, GHOSTX uses dropoff termination [4].

Some seeds may overlap with others after the seed search and the ungapped extension step. In particular, if there is a sequence highly similar to a query in the database, many seeds that overlap with others are found, and almost identical alignments are often obtained from these overlapped seeds. Thus, it is necessary to merge such overlapped seeds to reduce the number of gapped extensions. Therefore, GHOSTX uses a chain filtering technique. There are two cases in which the seeds are filtered out, as shown in Figure 5. First, if two seeds \( \{S_{db}[i, i+x], S_q[k, k+x]\} \) and \( \{S_{db}[j, j+y], S_q[l, l+y]\} \) overlap as shown in Figure 5A, GHOSTX combines these overlapped seeds together into one. Second, if two seeds \( \{S_{db}[i, i+x], S_q[k, k+x]\} \) and \( \{S_{db}[j, j+y], S_q[l, l+y]\} \) do not overlap but the score exceeds the dropoff parameter used for the ungapped extension step, as shown in Figure 5B, GHOSTX also merges the overlapped seeds.

### Gapped Extension

Those seeds judged as meaningful by the chain filter are extended with gaps. In the gapped extension, GHOSTX employs dynamic programming and the same heuristics as BLAST. In BLAST gapped extension, the process stops if the score is much lower than the best score, which saves computation time. GHOSTX also employs this technique and uses the same cutoff parameter.
**Database Division**

GHOSTX requires a large amount of memory in its homology search. Memory size depends on database size. However, computing systems generally have relatively small memory sizes compared with current database sizes. Therefore, GHOSTX divides a database into several chunks, each of whose size is \( l_{db} \), before it constructs its indexes. GHOSTX sequentially searches each database chunk, and merges its results with the results of previous chunk searches, when this chunk division is performed before the construction of its database indexes. GHOSTX dramatically reduces working memory requirements using this approach.

**Multithreading Implementation**

GHOSTX can be run in multithreading mode. Each query is searched independently, and GHOSTX divides query sequences into several parts. Therefore, each thread independently searches different parts of the query sequences. GHOSTX uses OpenMP for running in multithread mode.

**Results and Discussion**

**Datasets and Conditions**

To evaluate the performance of our tool, we compared its search sensitivity and computation time to National Center for Biotechnology Information (NCBI) BLASTX (version 2.2.28+), BLAT (version 34 standalone) and RAPSearch (version 2.12). We used the binaries of BLASTX and BLAT downloaded from Web sites. We used RAPSearch compiled with GCC (version 4.3.4) and the -O3 optimizing option. We also compiled GHOSTX using GCC with the -O3 optimizing option and –fopenmp, because GHOSTX can use OpenMP for multithreading. We used a database obtained from KEGG GENES [11], [12] protein sequences as of May 2013. This database contained approximately 10 million protein sequences, with a total size of approximately 3.6 billion residues (3.9 GB). We also used another database obtained from NCBI non-redundant protein sequences (nr) that contained 25 million sequences, approximately 8.6 billion residues (14.8 GB), to check our algorithm’s dependency on database size. For the query sequences, we used 2 query sets: one from human microbiome metagenomic sequences (SRS011098), and the other of soil microbiome metagenomic sequences (SRR011098), and the other of soil microbiome metagenomic sequences (SRR444039). SRS011098 was obtained from the Data Analysis and Coordination Center for Human Microbiome Project [13] Web site (http://www.hmpdacc.org/). We used the whole metagenomic shotgun sequencing data from SRS011098. SRR444039 was obtained from the Sequence Read Archive. 10 thousand randomly selected DNA short reads were used from both sets, SRS011098 and SRR444039. We also used 100 thousand randomly selected high quality DNA short reads from SRS011098 to measure multithreading computation time. We performed the analyses on a workstation with two 2.93 GHz Intel Xeon 5670 processors for a total of 12 CPU cores and 54 GB of memory.

**Relationship between GHOSTX Parameters and Sensitivity and Computation Time**

GHOSTX has two parameters for its seed search, threshold of the seed search \( T_{seed} \), and an upper mismatch score \( D \). These parameters affect the performance of GHOSTX. Therefore, we first searched for optimal parameters. To determine the best parameters, we used \( T_{seed} = 22, 24, 26, 28, 30 \) and \( D = 1, 4, 7 \). To evaluate search sensitivity, we used the search results obtained using Smith-Waterman local alignment by SSEARCH [14] as the

| Computation time (sec.) | Acceleration ratio |
|-------------------------|--------------------|
| GHOSTX                  | 1003.5             | 130.8              |
| RAPSearch               | 1404.1             | 93.4               |
| RAPSearch in fast mode  | 223.8              | 586.2              |
| BLAT                    | N/A                | N/A                |
| BLAST                   | 131213.3           | 1.0                |

The first, second, and third columns show the name of each program, the computation time, and the acceleration in processing speed relative to BLASTX using 1 thread, respectively. doi:10.1371/journal.pone.0103833.t004

**Figure 6. Search sensitivity of each tool with KEGG GENES.** The vertical axis shows the percentage of correct answers that correspond to the correct answers for each method. The horizontal axis shows the E-value of the alignments. doi:10.1371/journal.pone.0103833.g006

**Figure 7. Computation times with multithreading.** doi:10.1371/journal.pone.0103833.g007
The search space in the seed search is large and the number of seeds is small. This is because the Smith-Waterman algorithm is based on the dynamic programming algorithm and does not use any heuristics, it returns an optimal local alignment. We analyzed the performance of the particular parameter in terms of the fraction of its results that corresponded to the correct answers. When the subject sequences that had the highest score by SSEARCH and each particular method corresponded on each query, the query was deemed correct. Table S1 shows the sensitivity and computation time of each different parameter. As shown in the table, when size is large or $D$ is small, the sensitivity of GHOSTX is low and its computing speed is fast. This is because the search space in the seed search is small and the number of seeds is small. However, when size is small or $D$ is large, the sensitivity of GHOSTX is high and its computing speed is slow. This is because the search space in the seed search is large and the number of seeds is large. We selected size = 30 and $D = 4$ as default parameters that have a good balance between sensitivity and computation time. We used those parameters in the following evaluations.

Evaluation of Search Sensitivity

To evaluate search sensitivity, we used sensitivity the same way as we evaluated the relationship between GHOSTX seed search parameters and their sensitivity and computation time. To evaluate the software, we executed the BLASTX program with the command line options “–outfmt 6 -comp_based_stats 0”, which instructed the program to output in tabular format, without using composition-based statistics [15], because composition-based statistics are not available in SSEARCH. We used default parameters for the other options. The BLAT program does not include a function to translate DNA reads to protein sequences. Therefore, we translated the DNA reads into protein sequences based on all six potential frames using a standard codon table before executing BLAT. We executed the BLAT program with the command line option “q = prot -t = prot -out = blast6”, which instructed the program to run the queries and database as protein sequences, and to output data in the BLAST tabular format. We could not execute BLAT when we used nr as a database because our machine has insufficient memory for the execution. Therefore, we only executed BLAT with KEGG GENES. We executed the RAPSearch program with 2 cases. One case used the default options and the other used the command line option “–a T”, which instructed the program to perform a fast mode search. For GHOSTX, we used the following parameters: threshold of the seed search $T_{seed}$ = 30, upper mismatch score $D = 4$, and size of the database chunk $l_{db} = 2$ GB. The other parameters used are the same as BLAST defaults. In Figure 6, GHOSTX shows lower sensitivity than BLASTX, especially for those hits with E-values above $10^{-3}$. However, alignments with such high E-values are not normally used in most practical analyses anyway, because it is difficult to judge whether the results are merely because of chance. In fact, most research have ignored those hits with such high E-values [1], [2]. Therefore, we think GHOSTX has sufficient search sensitivity for most practical analyses. The sensitivity of GHOSTX is clearly better than that of BLAT and RAPSearch in fast mode, and almost equal to, or better than that of RAPSearch.

Evaluation of Computation Time

We ran each method with the same commands as for the evaluation of search sensitivity to measure computation time. We used 2 query sets, 10 thousand randomly selected DNA short reads from SRS011098 and from SRR444039, and we used KEGG GENES as our database. Table 1 and Table 2 show the computation time for each program. As shown with each query set, GHOSTX showed accelerations of approximately 153 and 152 times with respect to BLASTX, and approximately 3.5 and 3.5 times with respect to BLAT. Additionally, GHOSTX was approximately 1.6 and 1.5 times faster than RAPSearch, even though GHOSTX showed better search sensitivity than RAPSearch at E-values above $10^{-3}$. GHOSTX outperforms BLASTX in reducing computation time. The processing time acceleration is caused by the use of a suffix array for its seed search and ungapped extension steps. GHOSTX was slower than RAPSearch in fast mode. However, the sensitivity of RAPSearch in fast mode is clearly lower than GHOSTX.

We also checked the dependency on the database size for each program by using a larger database. Table 3 and Table 4 show the computation times and accelerations for NCBI nr. GHOSTX showed a better acceleration ratio against BLASTX, as compared with the KEGG GENES database (approximately 165 times and

| Chunk size | Memory size for constructing index (GB) | Memory size for homology search (GB) |
|------------|----------------------------------------|-------------------------------------|
| 512 MB     | 4.6                                    | 4.2                                 |
| 1 GB       | 9.2                                    | 7.2                                 |
| 2 GB       | 18.2                                   | 13.3                                |

The first, second, and third columns show the size of the database chunk, the used memory size for constructing index (GB), and the used memory size for homology search (GB).
The first, second, and third columns show the size of the database chunk, the computation time, and the acceleration in processing speed relative to GHOSTX with 2 GB database chunks, respectively.

doi:10.1371/journal.pone.0103833.t007

Conclusions

We have developed an efficient algorithm for performing sequence homology searches, and have implemented it as GHOSTX. GHOSTX has sufficient search sensitivity for practical analyses. It uses an extremely efficient seed search algorithm, employing database and query suffix arrays, to achieve a well over 100 times faster sequence homology search than BLASTX. GHOSTX is also almost 1.4–1.6 times faster than RAPSearch, which is one of the fastest homology search tools available, even though GHOSTX is slightly more accurate. Currently, sequencing technology continues to improve, increasingly producing larger and larger quantities of data. This explosion of sequence data makes computational analysis with contemporary tools more difficult. We offer this tool as a potential solution to the problem.

Supporting Information

Table S1 Relationship between GHOSTX parameters and sensitivity and computation time. The first, second, and third columns show the parameter, the sensitivity, and the computation time. The sensitivity is calculated as the ratio of correctly searched queries whose E-values<10^{-3}.

(DOC)

Acknowledgments

The authors thank Prof. Ken Kurokawa and Dr. Takuji Yamada for their helpful discussions of metagenomic analysis.

Author Contributions

Conceived and designed the experiments: SS MK TI YA. Performed the experiments: SS. Analyzed the data: SS. Contributed reagents/materials/analysis tools: SS MK TI. Contributed to the writing of the manuscript: SS MK TI YA.

References

1. Kurokawa K, Itoh T, Kuwahara T, Oshima K, Toh H, et al. (2007) Comparative metagenomics revealed commonly enriched gene sets in human gut microbiomes. DNA Research : an international journal for rapid publication of reports on genes and genomes 14: 169–181. doi:10.1093/dnares/dm018.
2. Turnbaugh PJ, Ley RE, Mahowald MA, Magrini V, Mardis ER, et al. (2006) An obesity-associated gut microbiome with increased capacity for energy harvest. Nature 444: 1027–1031. Available: http://www.ncbi.nlm.nih.gov/pubmed/17183312. Accessed 1 November 2012.

3. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990) Basic local alignment search tool. Journal of Molecular Biology 215: 403–410. Available: http://linkinghub.elsevier.com/retrieve/pii/S0022283605803602. Accessed 24 May 2013.

4. Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, et al. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research 25: 3389–3402. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 146917&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 19 February 2014.

5. Kent WJ (2002) BLAT—The BLAST-Like Alignment Tool. Genome Research 12: 656–664. doi:10.1101/gr.229202.

6. Ye Y, Chee JH, Tang H (2011) RAPSearch: a fast protein similarity search tool for short reads. BMC Bioinformatics 12: 159. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 3113943&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 7 September 2011.

7. Zhao Y, Tang H, Ye Y (2012) RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data. Bioinformatics 29: 125–126. Available: http://bioinformatics.oxfordsjournals.org/content/29/1/125. Accessed 25 May 2013.

8. Manber U, Myers G (1990) Suffix arrays: a new method for on-line string searches. Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics. 319–327. Available: http://portal.acm.org/citation.cfm?id = 320176.320218&amp;coll = portal&amp;dl = ACM&amp;type = seris&amp;csid = SERIES422&amp;part = series&amp;WantType = Proceeding&amp;kmpid = Symposium. Accessed 2010 November 9.

9. Vouzis PD, Sahinidis NV (2011) GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics 27: 182–188. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 3018811&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 013 November 8.

10. Ghodsi M, Pop M (2009) Inexact Local Alignment Search over Suffix Arrays. Proceedings IEEE International Conference on Bioinformatics and Biomedicine 2009: 83–97. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 3029072&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 2013 July 12.

11. Kanehisa M (2009) KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Research 37: 27–30. Available: http://www.nar.oupjournals.org/cgi/doi/10.1093/nar/gtn271.1.29. Accessed 2013 May 27.

12. Kanehisa M, Goto S, Sato Y, Furumichi M, Tanabe M (2012) KEGG for integration and interpretation of large-scale molecular data sets. Nucleic Acids Research 40: D109–14. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 3245020&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 2013 May 22.

13. The Human Microbiome Project Consortium (2012) Structure, function and diversity of the healthy human microbiome. Nature 486: 207–214. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 3564958&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 2014 January 20.

14. Pearson WR (1991) Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. Genomics 11: 635–650. doi:10.1016/0888-7543(91)90074-L.

15. Altschul SF, Wootton JC, Gertz EM, Agarwala R, Margulies A, et al. (2005) Protein database searches using compositionally adjusted substitution matrices. The FEBs journal 272: 5101–5109. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid = 1343503&amp;tool = pmcentrez&amp;rendertype = abstract. Accessed 2011 August 3.