# Rapid DNA Signature Discovery Using A Novel Parallel Algorithm

Hsiao Ping Lee*†, Yen-Hsuan Huang‡ and Tzu-Fang Sheu§
*Department of Applied Information Sciences, Chung Shan Medical University, Taichung, Taiwan, 40201 ROC
Email: ping@csmu.edu.tw
†Department of Medical Research, Chung Shan Medical University Hospital, Taichung, Taiwan, 40201 ROC
‡Department of Applied Information Sciences, Chung Shan Medical University, Taichung, Taiwan, 40201 ROC
Email: kevin656504@hotmail.com
§Department of Computer Science and Communication Engineering, Providence University, Taichung, Taiwan, 43301 ROC
Email: fang@pu.edu.tw (corresponding author)

***Abstract*—DNA signatures provide valuable information that can be used in various applications in bioinformatics, for example the identification of different species. Rapid signature discovery algorithms are required by biologists to discover signatures. Since more and more computers are equipped with a CPU of many processing cores, parallelism becomes a feasible solution to accelerate the discovery. However, most of the existing signature discovery algorithms are sequential algorithms. Parallel signature discovery algorithms are rare. In this paper, a parallel signature discovery algorithm is proposed. The algorithm discovers hamming-distance-based signatures from DNA databases. The proposed algorithm is a parallel enhancement of an existing discovery algorithm. Through parallel computing, the algorithm accelerates the process of signature discovery. In the experiment on a human chromosome EST database of 88M bases, the proposed algorithm has up to 73.28% less processing time than the existing discovery algorithm when 4 processors are used.**

***Index Terms*—DNA signature, human chromosome EST database, parallel algorithm, unique signature discovery.**

## I. INTRODUCTION

Based on the assumptions of the theories of evolution and natural selection, almost all species shared a common ancestor at a point in time. Random mutations in DNAs sometimes lead to differently structured proteins. If such changes give rise to advantages in survival, the DNAs is prevailed in the gene pool. The advantageous mutations are one of the ways that genomes diverge from one another. The result of the evolution is that the different species might own some unique patterns in their DNA sequences, and the species can be identified by the unique patterns. For example, specific oligonucleotides have already been used in a polymerase chain reaction (PCR) method for the identification of 14 human pathogenic yeast species [1].

DNA patterns are referred to as unique signatures if they appear in a DNA database only once, and have some minimum mutation distance from all other patterns in the database. The unique signatures are used in several bioinformatics researches. For example, unique signatures are used to identify HIV-1 subtypes and 28S rDNA sequences from more than 400 organisms [2]; the selected signature probes with microarray analysis are used to identify bacteria [3].

Unique signature discovery is to find all unique signatures in a DNA database. The methods of unique signature discovery have been widely studied, and many related algorithms, tools and applications have been developed [2]–[14]. For example, insignia [6] is a web application for rapidly identifying unique DNA signatures. Zheng's algorithm [12] is a hamming-distance-based unique signature discovery algorithm. The algorithm deals with DNA databases, and discovers unique signatures from the databases. CMD [13] is an algorithm designed to discover all implicit signatures from DNA databases under a discovery condition, where the implicit signatures are the patterns that satisfy the discovery conditions looser than the given discovery condition.

The internal-memory-based unique signature Discovery (IMUS) algorithm [14] improves upon the Zheng's algorithm. The IMUS algorithm deals with DNA databases. The algorithm discovers hamming-distance-based unique signatures. Let $l$ and $d$ be two positive integers, where $d \leq l$. An $l$-pattern is a string of $l$ characters in the alphabet set {A, C, G, T}. A pattern $P$ is $(l, d)$-mismatched to a pattern $Q$ if the length of $P$ and $Q$ is $l$ and the hamming distance, which is the number of mismatches, between $P$ and $Q$ does not exceed $d$. A pattern $P$ is referred to as a unique signature under the discovery condition $(l, d)$ if and only if no other pattern $Q$ exists in the given DNA database such that $P$ and $Q$ are $(l, d)$-mismatched. The IMUS algorithm is designed for efficiently discovering the unique signatures under the discovery conditions of signature length $l$ and mismatch tolerance $d$.

The underlying idea of the IMUS algorithm is that the unique signatures appear after all of the patterns that are not unique are discarded. Instead of finding unique patterns, the IMUS algorithm focuses on finding non-unique patterns. The IMUS algorithm is based on the observation that if two patterns $P$ and $Q$ are $(l, d)$-mismatched, then at least one of the two halves of $P$ is $(l/2, \lfloor d/2 \rfloor)$-mismatched to the corresponding part of $Q$. The IMUS algorithm is a two-phase algorithm. In the first phase, the algorithm divides DNA sequences into patterns of length $l$. Each $l$-pattern consists of two consecutive $l/2$-patterns. An index system is built based

 ISBN: 978-1-61208-202-8 

$S \leftarrow$ divide all of the DNA sequences in the input database into $l$-patterns which comprise two consecutive $l/2$-patterns
$\sigma \leftarrow$ construct an index of $4^{l/2}$ entries, which is based on the $l/2$-patterns as index keys
**for** an entry $E$ in $\sigma$ **do**
  **for** a pattern $P$ in $E$ **do**
    **for** an entry $E'$ in $\sigma$, whose key is $(l/2, \lfloor d/2 \rfloor)$-mismatched to $E$'s key **do**
      compare $P$ to all patterns in $E'$
      **if** $P$ is $(l, d)$-mismatched to any of the compared patterns **then**
        discard $P$
      **end if**
    **end for**
  **end for**
**end for**
the remaining patterns is the unique signatures of $(l, d)$

Fig. 1. The IMUS algorithm.

on the $l/2$-patterns as index keys, in which $l$-patterns that contain same index keys are gathered in a single index entry. Assume that $E$ is an entry and its key is $K_E$. $P$ is an $l$-pattern in $E$. Based on the IMUS observation, all of the $l$-patterns that are $(l, d)$-mismatched to $P$ are in the entries whose keys are $(l/2, \lfloor d/2 \rfloor)$-mismatched to $K_E$. In the second phase, $P$ is compared to the patterns that are possibly $(l, d)$-mismatched to it. $P$ is discarded if it is $(l, d)$-mismatched to any of the compared patterns. The IMUS algorithm is presented in Figure 1.

Nowadays, CPUs of many processing cores are commonplace, and the prices of the CPUs are in an acceptable range. For example, the price of an Intel Core i7 870 quad-core CPU is around 300 US dollars in November, 2011. Parallel computing technology has been used in several bioinformatics research areas, such as sequence alignment and analyses [15], protein structure prediction [16], [17], and motif finding [18]. Based on our experiments made on a computer with an Intel 2.93GHz CPU, the IMUS algorithm spent about 12.5 hours to discover unique signatures from a database of 88M bases under the discovery condition of signature length 24 and mismatch tolerance 4. However, the IMUS algorithm is a sequential algorithm. The increasing number of processing cores in a CPU would not increase the discovery efficiency of the IMUS algorithm. Therefore, upgrading the IMUS algorithm to a parallel algorithm would further accelerate the signature discovery processes.

In this work, an algorithm that is called parallel internal-memory-based unique signature discovery (PIMUS) algorithm is proposed. The PIMUS algorithm is a parallel enhancement of the typical IMUS algorithm. To improve discovery efficiency, the PIMUS algorithm uses an efficient scheduling heuristic proposed in [13] to generate a reordered processing list. The processing list helps to reduce discovery time to approaching the optimal discovery time for a multi-processor platform. Based on the results from the experiments on human chromosome EST databases of 88.0 and 36.4M bases, the PIMUS algorithm respectively spent about 3.5 hours and 0.62 hours to discover signatures from the EST databases under the discovery condition (24,4) when four processing cores are used. the PIMUS algorithm has up to 71.35% and 72.06% less processing time than the typical IMUS algorithm in the signature discoveries.

TABLE I
A LIST OF 6 PATTERN ENTRIES AND THEIR PROCESSING TIME.

| ID | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Time | 5 | 8 | 36 | 4 | 13 | 4 |

The rest of the paper is organized as follows. The PIMUS algorithm is presented in Section II. The time complexity of the algorithm is analysed in Section III. The results of the performance evaluation about the proposed algorithm are presented in Section IV. Finally, the conclusions of this work are given in Section V.

## II. METHODS

The proposed parallel internal-memory-based unique signature discovery (PIMUS) algorithm discovers signatures efficiently from a DNA database that can be entirely loaded into main memory under a certain discovery condition. The PIMUS algorithm improves upon the IMUS algorithm, and accelerates signature discovery by using parallel computing.

An intuitive way to apply parallel computing to the IMUS algorithm is to assign randomly an available processor to process a pattern entries in sequential order. For example, a computer with $m$ processors is used to handle $n$ pattern entries. Initially, processor 1 can be assigned to entry 1, ...,and processor $m$ can be assigned to entry $m$. Assume that processor 3 is the first to complete its task; the processor is immediately assigned to the next entry, entry $m + 1$. The next available processor is similarly assigned to the next entry until all of the $n$ pattern entries are completed. The optimal processing time when $m$ processors are used is $1/m$ of the processing time of a single-processor computer.

Table I shows the processing time of six pattern entries. The entries can be treated in 70 time units by a single-processor computer. The optimal processing time is therefore 70/2=35 time units for a two-processor computer. However, in the case of the assignment in sequential order, processor 1 is assigned to entries A and C, and processor 2 is assigned to entries B, D, E and F. The assignment of the entries is presented in Figure 2. The processing time is 41 and 29 time units respectively. Since the processor that takes longest dominates the overall processing time, the overall processing time is 41 time units in this case, which exceeds the optimal processing time.

The order of pattern entries in the processing list influences the overall processing time for parallel discovery. An efficient scheduling heuristic, called the parallel entry list (PEL), is used in the CMD algorithm [13]. Figure 3 presents the PEL

 ISBN: 978-1-61208-202-8

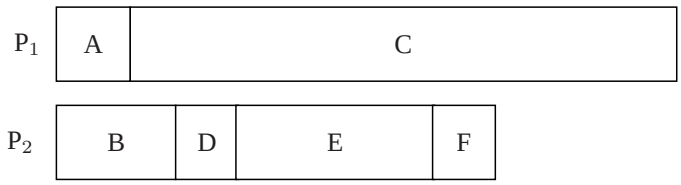


Fig. 2. The assignment of the entries in Table I in sequential order for a two-processor computer.

TABLE II
THE PROCESSING LIST GENERATED BY THE PEL HEURISTIC FOR PROCESSING THE ENTRIES IN TABLE I.

| ID | $C_1$ | $C_2$ | E | B | D | A | F |
|---|---|---|---|---|---|---|---|
| Time | 18 | 18 | 13 | 8 | 4 | 5 | 4 |

heuristic. The PEL heuristic yields a processing order list for pattern entries in which the entries that involve more patterns are before those that involve fewer. The PEL heuristic is similar to a partial quicksort. Unlike quicksort, the PEL heuristic is iterative, and only operates on the left part of a list in each iteration. The PEL heuristic focuses on all entries in the entry list initially. Assume $g$ is the average number of patterns in each entry within the focused part. The PEL heuristic moves the entries that contain more than $g$ patterns forward to the left part of the entry list in each iteration. In the next iteration, the heuristic focuses on the left part of the entry list, that consists of the entries that contain more than $g$ patterns. The time complexity of the PEL heuristic is $O(n)$, where $n$ is the number of pattern entries in the entry list. The processing list generated by the PEL heuristic for processing the entries in Table I is presented in Table II. Figure 4 presents the assignment of the entries in the processing list in sequential order for a two-processor computer. The overall processing time is 35 time units in this case, which equals the optimal processing time.

Figure 5 presents the PIMUS algorithm. Let $l$ be the desired signature length and $d$ be the mismatch tolerance. The PIMUS algorithm uses a DNA database as an input, and discovers all unique patterns from the database under the discovery condition $(l, d)$. The PIMUS algorithm uses the PEL heuristic to generate a processing order list, and applies parallel computing techniques to process multiple pattern entries simultaneously to accelerate signature discovery processes.

The PIMUS algorithm divides all of the DNA sequences in the input database into $l$-patterns. Each of the $l$-patterns comprises two consecutive $l/2$-patterns. An index of $4^{l/2}$ entries is built based on the $l/2$-patterns as entry keys. A multi-level index can be adopted if the index is too large to be fit in main memory. The $l$-patterns that contain one same key are collected in a single entry. A processing order list of the entries in the index is generated by the PEL heuristic. The reordered entry list makes the number of patterns treated by each of the processors approximately equal. It reduces the overall discovery time to approaching optimal processing time

$L \leftarrow$ generate a processing order list that consists of all pattern entries in arbitrary order
$m \leftarrow$ the number of available processors
$n \leftarrow$ the number of entries in $L$
$g, h \leftarrow$ the average number of patterns in each entry in $L$
**while** $g < mh$ **do**
  $s \leftarrow 0$, $k \leftarrow 1$
  **while** $n > k$ **do**
    **while** $|L_n| \leq g$ **do**
      $n \leftarrow n - 1$
    **end while**
    **while** $|L_k| > g$ **do**
      $s \leftarrow s + |L_k|$
      $k \leftarrow k + 1$
    **end while**
    **if** $n > k$ **then**
      exchange $L_k$ and $L_n$
      $s \leftarrow s + |L_k|$
    **end if**
  **end while**
  $g \leftarrow s/n$
**end while**
**for** $i \leftarrow 1$ to $n$ **do**
  $Y \leftarrow L_i$
  divide $Y$ into $m$ partitions $Y_1, Y_2, \ldots, Y_m$
  remove $L_i$ from $L$
  put $Y_1, Y_2, \ldots, Y_m$ into $L$
**end for**
**return** $L$

Fig. 3. The PEL heuristic. $L_i$ is the $i$-th entry in $L$. $|L_i|$ is the number of patterns in $L_i$.

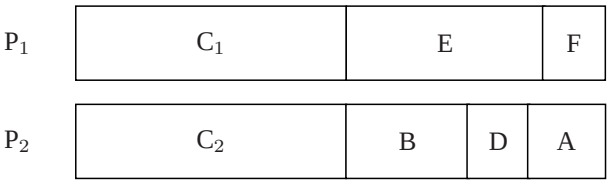


Fig. 4. The assignment of the entries in the processing list in Table II in sequential order for a two-processor computer.

when parallel computing is used.

**Observation 1 (IMUS Observation).** If two patterns $P$ and $Q$ are $(l, d)$-mismatched, then at least one of the two halves of $P$ is $(l/2, \lfloor d/2 \rfloor)$-mismatched to the corresponding part of $Q$.

An available processor is assigned to handle the next untreated entry in the processing order list. Two index entries are called similar entries if the number of mismatches between the keys of the two entries is less than or equal to $\lfloor d/2 \rfloor$. Assume that $E$ is an index entry, and $P$ is an $l$-pattern listed in $E$. Based on the IMUS Observation, if a pattern $Q$ is $(l, d)$-mismatched to $P$, then $Q$ must be in one of the entries

$S \leftarrow$ divide all of the DNA sequences in the database into $l$-patterns
$\Gamma \leftarrow$ construct the index of $4^{l/2}$ entries based on $S$
$L \leftarrow$ generate a processing order list of the entries in $\Gamma$ by using the PEL heuristic
**for** an entry $E$ in $L$ **do**
  assign an available processor to handle $E$
  **for** a pattern $P$ in $E$ **do**
    **for** an entry $E'$ in $\Gamma$, which is similar to $E$ **do**
      compare $P$ to all $Q$s, where $Q \in E'$
      **if** $P$ is $(l, d)$-mismatched to any of the compared $Q$s **then**
        set the duplication flag of $P$ to true
      **end if**
    **end for**
  **end for**
**end for**
discard all of the non-unique $l$-patterns
**return** the remaining $l$-patterns, which are the unique signatures of $(l, d)$ in the database

Fig. 5. The PIMUS algorithm.

similar to $E$. To check the uniqueness of $P$, $P$ is compared to all patterns in the entries which are similar to $E$. In each of the comparisons, $l/2$ characters, excluding the key region, are compared. $P$ is not unique if it is $(l, d)$-mismatched to any of the compared patterns. After all of the entries in the index are treated, the non-unique patterns are discarded. The remaining patterns are the unique signatures under the discovery condition $(l, d)$ in the input database.

## III. Mathematical Analysis

Let $l$ be the signature length and $d$ be the mismatch tolerance. The time complexity of the PIMUS algorithm under the discovery condition $(l, d)$ when $m$ processors are used is analyzed.

Assume $D$ is the input database, and $|D|$ denotes the size of the database. $\Gamma$ denotes the index used in the PIMUS algorithm. $\Gamma$ consists of $4^\alpha$ pattern entries under the discovery condition $(l, d)$, where $\alpha = l/2$ is the length of the entry keys. Let $\Gamma_i$ denote the $i$-th entry in $\Gamma$, where $1 \leq i \leq 4^\alpha$. $|\Gamma_i|$ denotes the number of patterns in $\Gamma_i$. The relationship between $|D|$ and $|\Gamma_i|$ is:

$$\sum_{i=1}^{4^\alpha} |\Gamma_i| = 2|D|$$

Assume that $E$ and $E'$ are two entries in $\Gamma$. HD$(E, E')$ denotes the hamming distance between $E$ and $E'$, which is defined as the hamming distance between the entry keys of $E$ and $E'$. A pattern $P$ in an entry $\Gamma_i \in \Gamma$ requires $\sum_j |\Gamma_j|$ string comparisons to check if the patterns that are $(l, d)$-mismatched to it exist, where $\Gamma_j \in \Gamma$ such that HD$(\Gamma_i, \Gamma_j) \leq \lfloor d/2 \rfloor$. All characters in the $l$-pattern $P$, excluding the entry key region, are compared in each of the string comparisons, yielding $l - \alpha = l/2$ character comparisons. Therefore, the number of character comparisons that is used to process all patterns in $\Gamma_i$ is $(l/2)|\Gamma_i| \sum_j |\Gamma_j|$.

The total amount of character comparisons used in the discovery under the discovery condition $(l, d)$, denoted as $M_{l,d}$, is:

$$M_{l,d} = \sum_{i=1}^{4^\alpha} (l/2)|\Gamma_i| \sum_j |\Gamma_j|$$

where $\alpha = l/2$ and $\Gamma_j \in \Gamma$ such that HD$(\Gamma_i, \Gamma_j) \leq \lfloor d/2 \rfloor$.

Assume the input database $D$ is in uniform distribution. In this case, each entry $\Gamma_i \in \Gamma$ should contain $|\Gamma_i| \approx 2|D|/4^\alpha$ patterns because of the assumption of uniform distribution. the amount of character comparisons used in the discovery under the discovery condition $(l, d)$, denoted as $\overline{M}_{l,d}$, is:

$$\begin{aligned} \overline{M}_{l,d} &= \sum_{i=1}^{4^\alpha} (l/2)|\Gamma_i| \sum_j |\Gamma_j| \\ &= \sum_{i=1}^{4^\alpha} (l/2)(2|D|/4^\alpha)\kappa(2|D|/4^\alpha) \\ &= 4^\alpha (l/2)\kappa(2|D|/4^\alpha)^2 \\ &= 2l\kappa|D|^2/4^\alpha \end{aligned}$$

where $\alpha = l/2$. $\Gamma_j \in \Gamma$ such that HD$(\Gamma_i, \Gamma_j) \leq \lfloor d/2 \rfloor$. $\kappa = \sum_{k=0}^{\lfloor d/2 \rfloor} 3^k \binom{\alpha}{k}$ is the number of all possible permutations that the number of changes does not exceed $\lfloor d/2 \rfloor$ bases in a string of length $\alpha$.

The time complexity of the PIMUS algorithm when $m$ processors are used, denoted as $\overline{M}^m_{l,d}$, is:

$$\begin{aligned} \overline{M}^m_{l,d} &= \overline{M}_{l,d}/m \\ &= 2l\kappa|D|^2/(4^\alpha m) \end{aligned}$$

## IV. Experimental Results

The platform that was adopted in the experiments was a personal computer with an Intel Core i7 870 2.93GHz quad-core CPU, 16GB RAM and 1.5TB disk space. The operating system was CentOS release 5.5. The algorithms were implemented in JAVA language, and the programs were compiled by JDK 1.6. The DNA data that were used in the experiments were from the human chromosome 4 and 13 EST databases. The experimental data were denoted as $D_4$ (human chromosome 4 EST database) and $D_{13}$ (human chromosome 13 EST database) respectively, and their corresponding sizes were approximately 88.0M and 36.4M bases. Before the experiments, the remarks in the databases were removed; all of the universal characters, such as 'don't care', were replaced with 'A', and DNA sequences that were shorter than 36 bases were discarded. The experiments in this section focused on discovering signatures of length between 24 and 30 with mismatch tolerances of two and four.

 ISBN: 978-1-61208-202-8 

TABLE III
THE PERFORMANCE OF THE PIMUS ALGORITHM WHEN USING 4 PROCESSING CORES. THE TIME UNIT IS A SECOND.

| (A) database = $D_4$ | | | |
|---|---|---|---|
| $(l, d)$ | IMUS | PIMUS | Saving(%) |
| (30,2) | 4172.08 | 1150.29 | 72.43 |
| (28,2) | 6305.20 | 1710.57 | 72.87 |
| (26,2) | 7324.49 | 2054.43 | 71.95 |
| (24,2) | 9523.10 | 2627.35 | 72.41 |
| (30,4) | 8389.32 | 2252.30 | 73.15 |
| (28,4) | 14113.54 | 3941.78 | 72.07 |
| (26,4) | 23998.85 | 6413.65 | 73.28 |
| (24,4) | 44951.49 | 12878.85 | 71.35 |

| (B) database = $D_{13}$ | | | |
|---|---|---|---|
| $(l, d)$ | IMUS | PIMUS | Saving(%) |
| (30,2) | 1048.50 | 223.12 | 78.72 |
| (28,2) | 945.93 | 261.86 | 72.32 |
| (26,2) | 1184.62 | 325.79 | 72.50 |
| (24,2) | 1753.88 | 458.23 | 73.87 |
| (30,4) | 2333.16 | 553.16 | 76.29 |
| (28,4) | 2532.26 | 720.57 | 71.54 |
| (26,4) | 4046.71 | 1139.96 | 71.83 |
| (24,4) | 7959.96 | 2224.20 | 72.06 |

For reasons of performance and memory consumption, a two-level index was used in the implementation of the IMUS and PIMUS algorithms. The first level of the index comprised $4^{10}$ direct-accessible entries, and a binary search was used to locate a specified entry in the second level. Since the purpose of our experiments was to evaluate the improvements provided by parallel computing, additional filters, such as the frequency filter that was used in the IMUS algorithm, was excluded from the implementation of the algorithms.

The improvements in discovery performance delivered by the PIMUS algorithm were examined. For 4 processing cores, the performance of the PIMUS algorithm was evaluated by using the algorithm to discover signatures from the experimental databases, $D_4$ and $D_{13}$. The percentage time saved is used to evaluate the improvements in the processing time of signature discovery. The time saving is defined as (1-(processing time of the PIMUS algorithm)/(processing time of the IMUS algorithm))*100%. A larger 'saving' means a greater improvement by the PIMUS algorithm. Table III presents the processing time that for the IMUS and PIMUS algorithms under various discovery conditions. The table also presents the time savings delivered by the PIMUS algorithm. The experimental results reveal that the PIMUS algorithm with 4 processing cores requires up to 78.72% less processing time to discover all signatures from $D_{13}$ than the IMUS algorithm under the discovery condition (30,2). Moreover, more than 71.35% of the processing time is saved under every discovery condition in the experiment. Restated, the proposed PIMUS algorithm performs at least 3.49 times faster than the IMUS algorithm when 4 processing cores are used.

To elucidate the benefits of parallel computing for signature discovery, various number of processing cores were used and the PIMUS algorithm was used to discover the signatures of ($l = 30, d = 4$) from $D_4$ and $D_{13}$. Table IV shows the experimental results. The acceleration is the processing time normalized to the processing time when one processor is used. The acceleration values of the PIMUS algorithm increase with the number of processing cores used in the experiment. For example, to discover signatures from $D_{13}$, the discovery processes that use 2, 3 and 4 processing cores are approximately 2.23, 3.13 and 4.22 times faster than those that use a single processing core respectively.

TABLE IV
THE BENEFITS OF PARALLEL COMPUTING FOR SIGNATURE DISCOVERY. THE TIME UNIT IS A SECOND.

| (A) database = $D_4$ | | |
|---|---|---|
| CPUs | Time | Acceleration |
| 1 | 8389.32 | 1.00 |
| 2 | 4239.86 | 1.98 |
| 3 | 3021.27 | 2.78 |
| 4 | 2252.30 | 3.72 |

| (B) database = $D_{13}$ | | |
|---|---|---|
| CPUs | Time | Acceleration |
| 1 | 2333.16 | 1.00 |
| 2 | 1048.36 | 2.23 |
| 3 | 744.46 | 3.13 |
| 4 | 553.16 | 4.22 |

The PIMUS algorithm treats pattern entries based on their order in a processing list. The influence on discovery efficiency made by the processing list was examined. The PIMUS algorithm that uses the processing list generated by the PEL heuristic is denoted as $\text{PIMUS}_\text{P}$ and that uses the processing list of pattern entries in the original order in index is denoted as $\text{PIMUS}_\text{N}$. To evaluate the improvements in the discovery efficiency of the PIMUS algorithm provided by the PEL heuristic, $\text{PIMUS}_\text{P}$ and $\text{PIMUS}_\text{N}$ were respectively used to discover signatures from $D_4$ and $D_{13}$ in this experiment. Table V presents the improvements in the discovery efficiency of the PIMUS algorithm delivered by the PEL heuristic when 4 processing cores were used. The percentage time saved is used to evaluate the benefits to the PIMUS algorithm made by the PEL heuristic. The time saving is defined as (1-(processing time of the $\text{PIMUS}_\text{P}$ algorithm) / (processing time of the $\text{PIMUS}_\text{N}$ algorithm))*100%. A larger 'saving' means a greater improvement delivered by the PEL heuristic. The experimental results reveal that the PIMUS algorithm that uses the processing list generated by the PEL heuristic saves up to 41.35% overall processing time than that uses the processing list of pattern entries in the original order in index. The amount of time used by the PEL heuristic to reorder the processing list is presented in Table VI. All of the processing time used by the PEL heuristic to reorder the processing list are less than 1.03 seconds in the experiment. Compared with the discovery time, the generation time of the reordered processing lists is negligible.

## V. CONCLUSIONS

This work proposes a parallel unique signature discovery algorithm called parallel internal-memory-based unique signature discovery (PIMUS) algorithm. The PIMUS algorithm is a parallel enhancement of the existing IMUS algorithm. The proposed PIMUS algorithm discovers hamming-distance-based unique signatures under a certain discovery condition efficiently. For example, when 4 processing cores are used, the PIMUS algorithm can discover the unique signatures of length 30 and mismatch tolerance 2 in 1150 seconds from

 ISBN: 978-1-61208-202-8

TABLE V
THE BENEFITS TO THE PIMUS ALGORITHM MADE BY THE PEL HEURISTIC WHEN 4 PROCESSING CORES WERE USED. THE TIME UNIT IS A SECOND.

| (A) database = $D_4$ | | | |
|---|---|---|---|
| $(l, d)$ | $\text{PIMUS}_\text{N}$ | $\text{PIMUS}_\text{P}$ | Saving(%) |
| (30,2) | 1359.70 | 1150.29 | 15.40 |
| (28,2) | 2194.71 | 1710.57 | 22.06 |
| (26,2) | 2585.03 | 2054.43 | 20.53 |
| (24,2) | 3439.14 | 2627.35 | 23.60 |
| (30,4) | 2523.20 | 2252.30 | 10.74 |
| (28,4) | 4554.72 | 3941.78 | 13.46 |
| (26,4) | 7331.77 | 6413.65 | 12.52 |
| (24,4) | 14253.49 | 12878.85 | 9.64 |
| (B) database = $D_{13}$ | | | |
| $(l, d)$ | $\text{PIMUS}_\text{N}$ | $\text{PIMUS}_\text{P}$ | Saving(%) |
| (30,2) | 360.42 | 223.12 | 38.09 |
| (28,2) | 435.27 | 261.86 | 39.84 |
| (26,2) | 555.50 | 325.79 | 41.35 |
| (24,2) | 760.05 | 458.23 | 39.71 |
| (30,4) | 737.69 | 553.16 | 25.01 |
| (28,4) | 813.03 | 720.57 | 11.37 |
| (26,4) | 1315.04 | 1139.96 | 13.31 |
| (24,4) | 2987.61 | 2224.20 | 25.55 |

TABLE VI
THE PROCESSING TIME USED BY THE PEL HEURISTIC TO GENERATE THE REORDERED PROCESSING LIST. THE TIME UNIT IS A SECOND.

| (A) database = $D_4$ | | (B) database = $D_{13}$ | |
|---|---|---|---|
| $(l, d)$ | Time | $(l, d)$ | Time |
| (30,2) | 1.03 | (30,2) | 0.72 |
| (28,2) | 0.92 | (28,2) | 0.65 |
| (26,2) | 0.80 | (26,2) | 0.60 |
| (24,2) | 0.51 | (24,2) | 0.45 |
| (30,4) | 1.03 | (30,4) | 0.71 |
| (28,4) | 0.92 | (28,4) | 0.65 |
| (26,4) | 0.76 | (26,4) | 0.60 |
| (24,4) | 0.54 | (24,4) | 0.46 |

an EST database of 88M bases. Compared with the typical IMUS algorithm, it saves more than 72% of the discovery time. The PIMUS algorithm can be used to rapidly discover signature data for further analysis, for example finding implicit signatures.

## ACKNOWLEDGMENT

The authors would like to thank the National Science Council of the Republic of China, Taiwan, for financially supporting this research under Grants 100-2218-E-040-001.

## REFERENCES

[1] B. M. Kiryu and C. P. Kiryu., "Rapid identification of candida albicans and other human pathogenic yeasts by using oligonucleotides in a pcr." *J. Clin. Microbiol.*, vol. 73, pp. 1634–1641, 1998.

[2] L. Kaderali and A. Schliep., "Selecting signature oligonucleotides to identify organisms using dna arrays." *Bioinformatics*, vol. 18, no. 10, pp. 1340–1349, 2002.

[3] P. Francois, Y. Charbonnier, J. Jacquet, D. Utinger, M. Bento, D. Lew, G. M. Kresbach, M. Ehrat, W. Schlegel, and J. Schrenzel, "Rapid bacterial identification using evanescent-waveguide oligonucleotide microarray classification," *Journal of Microbiological Methods*, vol. 65, no. 3, pp. 390–403, 2006.

[4] A. M. Phillippy, J. A. Mason, K. Ayanbule, D. D. Sommer, E. Taviani, A. Huq, R. R. Colwell, I. T. Knight, and S. L. Salzberg, "Comprehensive dna signature discovery and validation," *PLoS Computational Biology*, vol. 3, no. 5, 2007.

[5] E. K. Nordberg, "Yoda: selecting signature oligonucleotides," *Bioinformatics*, vol. 21, pp. 1365–1370, 2005.

[6] A. M. Phillippy, K. Ayanbule, N. J. Edwards, and S. L. Salzberg, "Insignia: a dna signature search web server for diagnostic assay development." *Nucleic Acids Research*, vol. 37, no. 2, pp. 229–234, 2009.

[7] S. H. Chen, C. Z. Lo, S. Y. Su, B. H. Kuo, C. A. Hsiung, and C. Y. Lin., "Ups 2.0: unique probe selector for probe design and oligonucleotide microarrays at the pangenomic/genomic level." *BMC Genomics*, vol. 4, no. 6, 2010.

[8] R. C. Fry, M. S. DeMott, J. P. Cosgrove, T. J. Begley, L. D. Samson, and P. C. Dedon., "The dna-damage signature in saccharomyces cerevisiae is associated with single-strand breaks in dna." *BMC Genomics*, vol. 7, no. 313, 2006.

[9] M. W. J. van Passel, E. E. Kuramae, A. C. M. Luyf, A. Bart, and T. Boekhout., "The reach of the genome signature in prokaryotes." *BMC Evolutionary Biology*, vol. 6, no. 84, 2006.

[10] M. Nicolau, R. Tibshirani, A.-L. Borresen-Dale, and S. S. Jeffrey., "Disease-specific genomic analysis: identifying the signature of pathologic biology." *Bioinformatics*, vol. 23, pp. 957–965, 2007.

[11] K. C. Bader, C. Grothoff, and H. Meier., "Comprehensive and relaxed search for oligonucleotide signatures in hierarchically-clustered sequence datasets." *Bioinformatics*, vol. 27, pp. 1546–1554, 2011.

[12] T. J. J. Zheng, T. J. Close and S. Lonardi., "Efficient selection of unique and popular oligos for large est databases." *Bioinformatics*, vol. 20, pp. 2101–2112, 2004.

[13] H. P. Lee, T. F. Sheu, and C. Y. Tang, "A parallel and incremental algorithm for efficient unique signature discovery on dna databases." *BMC Bioinformatics*, vol. 11, p. 132, 2010.

[14] H. P. Lee, T. F. Sheu, Y. T. Tsai, C. H. Shih, and C. Y. Tang., "Efficient discovery of unique signatures on whole-genome est databases." in *Proceeding of the 20th annual ACM Symposium on Applied Computing (SAC2005)*, 2005, pp. 100–104.

[15] Y. Chen, A. Wan, and W. Liu., "A fast parallel algorithm for finding the longest common sequence of multiple biosequences." *BMC Bioinformatics*, vol. 7, no. 4, 2006.

[16] W. Sun, S. Al-Haj, and J. He., "Parallel computing in protein structure topology determination." in *Proceedings of 26th Army Science Conference*, 2008.

[17] J. R. Green, M. J. Korenberg, and M. O. Aboul-Magd., "Pci-ss: Miso dynamic nonlinear protein secondary structure prediction." *BMC Bioinformatics*, vol. 10, no. 222, 2009.

[18] W. N. Grundy, T. L. Bailey, and C. P. Elkan., "Parameme: a parallel implementation and a web interface for a dna and protein motif discovery tool." *Bioinformatics*, vol. 12, pp. 303–310, 1999.

 ISBN: 978-1-61208-202-8