UASMAs (Universal Automated SNP Mapping Algorithms): a set of algorithms to instantaneously map SNPs in real time to aid functional SNP discovery

James T.L. Mah
Data Mining Department, Institute for Infocomm Research (I2R), A*STAR, Singapore
tlmah@i2r.a-star.edu.sg

Danny C.C. Poo
Department of Information Systems, National University of Singapore
dannypoo@nus.edu.sg

Shaojiang Cai
Department of Information Systems, National University of Singapore
caishaojiang@gmail.com

ABSTRACT

Currently, submission of new SNP entries into SNP repositories such as dbSNP by NCBI is done by manual curation. This gives rise to errors and ambiguities in SNP data entries. Due to the exponential increase in SNP discovery, there is a necessity to create algorithms to accurately and rapidly map SNPs as they are discovered in real time and depositing these entries automatically into a central SNP database. UASMAs are a set of algorithms to instantaneously map SNPs efficiently and accurately by their unique chromosome position in real time. It is the result of integration of structures and algorithms in state of the art alignment methods MAQ, BWT-SW, Bowtie, SOAP2 and BWA.

Using BLAST employed by NCBI as benchmark where recall was at most 91%, recall performance of components Bowtie and BWA were much better at up to 99% for longer reads. Similarly, Bowtie and BWA performed better in terms of precision at greater than 91 % whereas BLAST was only 78 – 88%. BLAST performed poorly in terms of recall and precision for longer reads. Bowtie and BWA algorithms in UASMAs were superior in terms of performances in alignment of longer sequences and locating the precise chromosome position of any SNP with respect to the NCBI reference assembly. Results obtained are fast, instantaneous and accurate.

Using UASMAs prove to be fast and optimal in mapping new variants onto the genome in view of depositing these entries accurately into a central database. Because it is done in real-time and with increased accuracy, recall and precision, the database created will be complete, up-to-date and devoid of ambiguities and redundancies.

Proceedings of the VLDB Endowment, Vol. 3, No. 2

1. INTRODUCTION

Single-Nucleotide Polymorphisms (SNPs) are the most prevalent type of DNA sequence variation and of research interest because of their applications in personalized medicines and genome wide association (GWA) studies. Currently, submission of new SNP entries into SNP repositories such as dbSNP by NCBI [20] is done by manual curation. It is not real time nor automated. As a result, redundant, incomplete and inaccurate SNP records abound, without an efficient feedback system to detect, correct and prevent this from occurring. Our preliminary experiments on dbSNP data verified the inadequacy (see Section 3.2). These errors and ambiguities in SNP data entries, inaccuracies and redundancies in dbSNP are difficult to resolve afterwards due to the magnitude of the number of stored SNPs. Due to the exponential increase in the discovery of SNPs, there is an increasing necessity to create algorithms to accurately and rapidly map SNPs as they are discovered in real time and depositing these entries automatically into a central SNP database. The resulting SNP database created will be complete, up-to-date and without redundancies or ambiguities.

UASMAs (Universal Automated SNP Mapping Algorithms) are a set of algorithms developed to instantaneously map SNPs efficiently and accurately by their unique chromosome positions in real time. It is the result of integration of structures and algorithms in state of the art alignment methods MAQ, BWT-SW, Bowtie, SOAP2 and BWA.

2. METHODS

Our proposed theory assumes that any SNP sequence can be given an identity instantaneously. This is because the identifier of a SNP is the characteristic of the SNP itself, which can be easily found by performing local alignment of the SNP sequence against the chromosome the SNP resides in, thus giving us the absolute chromosome position of the SNP. Local alignment of a SNP against the NCBI reference assembly can be achieved efficiently with programs such as BWT-SW by Lam et al.[10], and Bowtie by Langmead et al.[11].

We assume that the identifier for a single-base polymorphism would be unique because it can only be found in a specific base position in a chromosome. To locate the chromosome position of a SNP, we integrated state-of-art data structures and algorithms inspired by MAQ [12], BWT-SW [10], Bowtie [11], SOAP2 [14]

and BWA [13]. We provide an efficient system for finding accurate alignments. In the pipeline is a front end product which can easily be presented in a web browser in the form of a user-friendly tool, from which a user is able to submit newly discovered SNP sequences. The resultant tool will then automatically and accurately map and immediately deposit the submitted SNPs into a central SNP database or repository. This would also work for SNPs submitted online by the scientific community, as it does not require an independent administrator to curate and manually allocate SNP RefSeq numbers present in dbSNP which could lead indirectly to errors, artefacts and redundant entries.

Given a variant sequence, we want to map the SNP by finding its chromosome position and all its local alignment and choosing the one with the best similarity score. The local alignment problem is as follows. Let T be the entire text of the chromosome sequence of length n, and P be the pattern of the SNP sequence of length m, and $m < n$. We want to find a region of text within T to align with P such that it maximizes the alignment score. A popular dynamic-programming algorithm for finding optimal local alignment is the Smith-Waterman algorithm [21], which takes $O(nm)$ time and space complexity for aligning the entire text T against the pattern P. This would be unfeasible for finding local alignments on large texts such as the human genome, which contains approximately 3 billion bases [8]. In such a case, the amount of memory required for such an alignment would be:

$$(3*10^9 \text{ bases}) * (2 \text{ bits per base}) * m \approx 750m\text{MB}$$

For practical purposes, we exploited currently popular alignment tools MAQ, BWT-SW, Bowtie, SOAP2 and BWA. Among them MAQ and SOAP2 mainly make use of hashing, while BWT-SW, Bowtie and BWA are based on Burrows-Wheeler Transform (BWT) [4]. Since BWT has been proved to be efficient for short read alignment [5, 6, 10, 11, 13, 14], here we introduce relevant data structures and local alignment algorithms in detail.

2.1 Suffix Trie

Because T is large, there are potentially many identical substrings that we want to avoid aligning repeatedly with P. We can treat these substrings as a common set of sub-problems which we only want to solve once. This can be done with the use of a search tree data structure such as the suffix tree suggested by McCreight [17]. Also, as a suffix tree is a compact representation of a suffix trie where all nodes with one child are merged with their parents, we can easily obtain its suffix trie.

Let Σ be the alphabet of characters in T. Each edge of the suffix trie is labeled with a character z from Σ and the concatenation of the characters on a path from the root to a node represents a unique suffix of T. Each leaf node stores the starting position of the corresponding suffix it represents. The preorder traversal of this suffix trie will generate all suffices of T.

Suffix trees have been used to solve a variety of problems in biological sequence analysis. Suffix trees can be constructed in time and space linear in the sequence length. However, it suffers seriously from its inefficiency of space usage. The best known implementation of a suffix tree on human genome requires $12.5n$ bytes for storage, which is approximately 40GB [15]. The figure far exceeds the 4G capacity of a standard PC nowadays. Disk based implementations have been introduced to deal with large suffix trees [1, 3], but they slow down the alignment process by orders of magnitude.

2.2 Dynamic Programming: Smith-Waterman

The Smith-Waterman algorithm solves the local alignment problem using dynamic programming [21]. Dynamic programming is performed at each node of the suffix trie to compute the best possible alignment score corresponding to the suffix represented by the path from the root to the node. Each node of the suffix trie will keep a set of dynamic programming tables defined by its recurrence relation for alignment of a pattern P against the suffix string formed by traversing from root to the node.

Let u be a node on this suffix trie and X be the suffix of length d represented by the path from the root to the node u, and we want to align a pattern P of length m with X. Also, let the character in the strings be specified by $X[i]$ and $P[j]$ such that $1 \le i \le d$ and $1 \le j \le m$. We define a table M such that every cell $M(i, j)$ has the maximum score for alignment of $X[1, \ldots, i]$ with $P[1, \ldots, j]$. For local alignments, the scoring scheme is the as follows:

(1) $X[i] = P[i]$; Matched pair has a score of a.

(2) $X[i] \neq P[i]$; Mismatched pair has a score of b.

Either $X[i, \ldots, j]$ or $P[i, \ldots, j]$ is a gap of length r, and has a score of $g + s * r$, where $a = 1$, $b = -3$, $g = -5$ and $s = -2$.

Any cell $M(i, j)$ is the maximum of the follow three cases of alignment M_1, M_2 and M_3.

(1) Let M_1 be the case where $X[i]$ is aligned $P[j]$.

(2) Let M_2 be the case where $X[i]$ is aligned with a gap.

(3) Let M_3 be the case where $P[i]$ is aligned with a gap.

M, M_1, M_2 and M_3 can be represented as memorized functions such that each function stores its results in a dynamic programming table. We can then express the above as a recurrence relation as follows.

(1) $M_1(i,j) = M(i-1, j-1) + S(X[i], P[j])$

(2) $M_2(i,j) = \max \{ M_2(i-1, j) + s, M(i-1, j) + g + s \}$

(3) $M_3(i,j) = \max \{ M_3(i, j-1) + s, M(i, j\text{-}1) + g + s \}$

(4) $$M(i,j) = \begin{cases} -\infty & \text{If } M_1, M_2 \text{ and } M_3 \le 0 \\ \max(M_1, M_2, M_3, 0) & \text{otherwise} \end{cases}$$

Where $S(X[i], P[j]) = a$ if $X[i] = P[j]$, or b otherwise.

The best alignment score at node u is given by the cell $M[i, j]$ with the highest alignment score and tracing backwards to the maximum of $M[i-1, j-1]$, $M[i-1, j]$ and $M[i, j-1]$.

Also, we consider a child node v of node u, such that the substring of v is $X[1, \ldots, d]c$, where c is the edge label between u and v. A dynamic programming table of a node and its child differs only by a single row. To compute the dynamic programming table of child node v, the corresponding table of node u is extended by a new row for aligning character c against T. Thus only the new row has to be computed. Similarly, to traverse up the suffix trie to a parent node, we delete the last row from the dynamic programming table of the child node.

Lam et al.[10] proposes a pruning strategy to stop traversing a path when the dynamic programming tables of a node u shows that

a positive similarity score cannot be obtained for the path and pattern being aligned. If the rows of all the dynamic programming tables do not have a positive entry, the values in the remaining rows will also be negative. Thus the entire subtree of node *u* can be pruned away.

Despite of above strategies, SW algorithm consumes too much memory for maintaining the dynamic tables. Especially for large genomes like human genome, its inefficiency becomes bottleneck.

2.3 Backward Search on BWT

In this section we describe the algorithm for string matching on Burrows-Wheeler Transform (BWT) [4]. At first we will introduce suffix array (SA) [16], which is another famous data structure for efficient string matching. Then we will describe FM-index [5], an algorithm to simulate a variant of SA based on BWT to further boost the performance. We illustrate the backward search process using an example through this section.

Let T be a string of length n that is comprised of characters from a set of alphabets Σ. We assume T ends with a special character '$' and that '$' is lexicographically smaller than all other characters in Σ. In a typical DNA context, Σ would be the set '$', 'A', 'C', 'G', 'T', sorted in ascending lexical order. A suffix of a sequence is a substring that begins at any position of the sequence and extends to the end of the zero sequence. For example, if T = "ACAACG$", then all the possible suffices of T sorted in ascending order and in 0-based indexing will be as follows:

Table 1. Example of sorted suffices of *T*

0-based index	Sorted suffices	Starting position of suffix in *T*
0	$	6
1	AACG$	2
2	ACAACG$	0
3	ACG$	3
4	CAACG$	1
5	CG$	4
6	G$	5

Table of suffices of T = 'ACAACG$' and their starting positions, sorted by lexical value. Note that the index starts counting from zero.

Next, we define the suffix array SA[0 … n - 1] of T as an array, where $SA[i]$ contains the starting position for the i^{th} smallest suffix of T. For example, referring to Table 1 above, the 5^{th} smallest suffix of T is 'CG$' and its starting position in T is index 4. The rest of the suffix array values of T follow in Table 2.

Table 2. Example of suffix array values of T

0-based Index	Values of suffix array SA[]
0	$SA[0] = 6$
1	$SA[1] = 2$
2	$SA[2] = 0$
3	$SA[3] = 3$
4	$SA[4] = 1$
5	$SA[5] = 4$
6	$SA[6] = 5$

The resulting suffix array is the positions of the suffices concatenated together, which is "6203145".

The resulting suffix array of T is simply the starting positions of the sorted suffixes in T concatenated together. Continuing from the previous example in Table 1, the suffix array of T is $SA[]$ = "6203145".

Now that we have seen the suffix array of T, we shall define the BWT of T. The BWT of T is the set of all cyclic shifts of T sorted in ascending lexical order, and then taking the last column of characters from the set gives us its BWT. Using the previous example T = "ACAACG$", we illustrate the BWT of T in Table 3.

Table 3. Example of the Burrow-Wheeler Transform of T

0-based index	Sorted cyclic shifts of T	Last column of characters
0	$ACAACG	G
1	AACG$AC	C
2	ACAACG$	$
3	ACG$ACA	A
4	CAACG$A	A
5	CG$ACAA	A
6	G$ACAAC	C

Table of BWT of T is BWT(T) = 'GC$AAAC'. Note that the cyclic shifts of T are the same as its sorted suffixes after wrapping around.

From our example in Table 3, the BWT of T is 'GC$AAAC'. The resulting BWT of T is the concatenation of the last column of characters. Notice that the cyclic shifts of T in Table 3 is the sorted suffixes of T in Table 2 after wrapping around. This establishes the relationship between BWT and suffix arrays, and which can be expressed as $BWT[i] = T[SA[i] - 1]$.

We can now perform the backward search technique as follows ([5, 6, 9]). Given a string X, let the SA range of X be $[i, j]$ such that $SA[i]$ and $SA[j]$ are the smallest and largest suffices of T respectively that have X as the prefix. Given the SA range of X, we can obtain the SA range $[p, q]$ of zX for any character z in Σ by using the backward search algorithm. The backward search algorithm uses the following definitions:

(1) Let $C(z)$ be the total number of characters in T that are smaller than z.

(2) Let $Occ(z, i)$ be the total number of character z in BWT$[0, \ldots, i]$.

(3) Given the SA range $[i, j]$ of X, the SA range $[p, q]$ of zX can be found with:

$$p = C(z) + Occ(z, i - 1)$$
$$q = C(z) + Occ(z, j) - 1$$

Referring to the previous example in Table 3, if X is the prefix 'AC', then the SA range of X would be [2, 3]. Now let us assume z = 'A', then the string zX = 'AAC'. Then, from the state of variables shown in Table 4, we find the SA range $[p, q]$of zX to be [1, 1].

Table 4. Variable states for finding SA range of zX = 'AAC'.

Variables	Values	Description
X	"AC"	Given string.
i	2	Start of SA range of X
j	3	End of SA range of X
BWT[0, …, n]	'GC$AAAC'	BWT of T.
z	"A"	X is extended by character z
$C(z)$	1	Number of characters in Σ smaller than z.
$Occ(z, i - 1)$	0	Number of z in BWT[0, …, 1]
$Occ(z, j)$	1	Number of z in BWT[0, …, 3]
$p = C(z) + Occ(z, i - 1)$	1	Start of SA range of zX
$q = C(z) + Occ(z, j) - 1$	1	End of SA range of zX

Table of variable states used to find p and q, which is the SA range of zX.

Ferragina and Manzini [5] also introduced an auxiliary data structure for precomputing $Occ(z, i)$ in constant time. $C(z)$ can also be precomputed and found in constant time. This implies that $[p, q]$ can be obtained from $[i, j]$ in constant time, and we can quickly check for the existence of a suffix zX from X in constant time by substituting z with the four possible nucleotide bases A, C, T, G. Since we the reverse of T is used instead, we instead check for the existence of zX^{-1} in T^{-1}. This can be done by finding the SA range $[p, q]$ of zX^{-1} using the BWT of T^{-1}. This edge exists if $p \leq q$.

According to Ferragina and Manzini [5], in FM-indexing suffix array is not needed to be stored explicitly for backward search. Only some auxiliary arrays are needed, which can be accessed in constant time. That overcomes largest problem of SA, which is eager consuming of memory for large genomes. FM-index consumes only $3n$ bits, i.e., $0.375n$ bytes in practice. For human genome, it is less than 1G. It is an ideal utility for our application, in which we would like to keep the whole index in main memory of PCs, meanwhile performing batch alignments efficiently. In addition, for matching a pattern P with length m, the time complexity is $O(m+occ\log^{\varepsilon}u)$, where occ is occurrence times in T and ε is a fixed constant between 0 and 1. This performance is competitive against all other approaches.

2.4 Backward Search with Substitution

Backward search on BWT introduced in Section 2.2 is just fit for exact string matching, which is insufficient for short read alignment. Instead of SW algorithm, Bowtie employs substitution during backward search process to allow mismatches. Each character in the read is assigned a numeric quality value, with lower value indicating higher error probability. It prefers alignments where the sum of the quality values at all mismatched positions are low. Similar to exact matching, it calculates SA ranges for successively longer query suffixes. In exact matching, the process stops whenever the SA range becomes empty. However, to allow inexact alignment, the algorithm would substitute current unmatched base with a different base, introducing one mismatch. If finally there are multiple alignments satisfying the alignment policy, Bowtie will pick the one with lowest quality value.

Bowtie introduces 'double indexing' to avoid excessive backtracking, which happens when the aligner spends useless effort on the positions close to the 3' end of the query. In addition to the forward index, Bowtie maintains 'mirror' index as well, which is the BWT of the reversed sequence. Then the read is split into halves, each searched with the forward and mirror indexes respectively. However, it works only when single mismatch is allowed, if there are multiple mismatches, Bowtie sets a threshold to stop excessive backtracking.

3. EXPERIMENTS

To determine the performance of the components in UASMAs, we compare alignment tools BWT-SW (bwtsw-x64-linux-20070916), Bowtie (Bowtie 0.12.1), MAQ (Maq 0.6.6), SOAP2 (SOAP2 2.20) and BWA (Bwa-0.5.5) in terms of parameters flexibility, functionality, memory usage, alignment time and alignment accuracy based on real SNPs data from largest public database dbSNP by NCBI Sherry et al.[20]. Different parameters and read lengths were tried to evaluate their performance. Meanwhile, BLAST [2], the tool currently employed by NCBI, acted as a benchmark.

All our experiments were conducted on a desktop with Duo Core 2.53Hz, 4G memory and 64-bit Linux Fedora 11 installed.

3.1 Data Source

Our experiments are based on real data, which are SNPs on chromosomes Y and 1 from dbSNP [20]. The unprocessed flat files acting as data source can easily be downloaded from the following three public databases:

(1) The UCSC reference genome (HG18) [1].

The database files are named "chr*.fa.gz" and they are available publicly. This database is stable and is not expected to change. The original human genome is 2.9G, including 24 chromosomes.

(2) dbSNP (build 130) SNPChrPosOnRef database[2].

The chromosome positions of SNPs relative to the reference genome are available at FTP of NCBI. This database is expected to change quarter-yearly due to clustering by dbSNP.

(3) SNPs in dbSNP in FASTA format[3].

The FASTA files include the RefSeq number, length, position, orientation and the sequence of SNPs, etc. We here choose SNPs on chromosomes 1 and Y, which contain 1393418 and 38340 SNPs respectively.

The basic procedures of our experiments are as follows: take the SNPs sequences as input, align them against reference genome, and check the returned positions against recorded positions in SNPChrPosOnRef.

3.2 Pre-processing

One important thing to note is that there are noises in SNPChrPosOnRef database. Firstly, position values of some SNPs were zero. We referred to the dbSNP online and found that the recorded position had two numbers in the form of <pos, pos+1>. Secondly, ac-

[1] http://hgdownload.cse.ucsc.edu/goldenPath/hg18/chromosomes/

[2] ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/database/organism_data/b130_SNPChrPosOnRef_36_3.bcp.gz

[3] ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/rs_fasta/

cording to the strict definition of single nucleotide polymorphism, only *SINGLE* mutations (one-base substitution / insertion / deletion) will be counted. We filtered out these noises in the experiments. The breakdown is shown in Table 5.

Table 5. Breakdown of Raw SNPs Data.

Chr	Raw #	Pos 0	Non SNPs	SNPs Counted
1	1393418	26988	169134	1199566
Y	38340	5324	2284	31198

SNPs on chromosomes 1 and Y. Note that there are some records those are non SNPs as well as having position 0.

In order to investigate the sensitivity to query length, we conducted separate experiments with various maximum lengths. They are 36, 50, 63, 76, 128, 256, 512 and 1024bp. If the original SNP sequence is longer than length considered, we truncate it to maximum allowed size, meanwhile of course, reserving the mutation position of the SNP. The metadata position and query length are also modified accordingly. Table 6 illustrates the distribution of different lengths of counted SNPs. We can see that around 94% (97% for Chr1) of the query sequences have length longer than 256bp, meaning that successively increasing the considered lengths does make sense.

We first perform the alignment using BLAST as the benchmark. For SNPs on chromosome Y, in best case (length 128) 91% are aligned successfully, with loose precision 97.7%. The result supports our opinion that current SNP entries in dbSNP could be inaccurate and confusing. Actually, some SNPs have been removed or renamed in dbSNP, for example, rs3893 [18]. But these names could still be cited in published papers and lead to confusion and ambiguities.

Table 6. Lengths of SNP Sequences

	<256	[256, 512)	[512, 1024)	>=1024
Chr Y	2.98%	26.44%	57.58%	12.99%
Chr 1	5.23%	34.36%	55.17%	5.25%

3.3 Indexing

Compressed indexing is able to reduce memory requirements for the human genome such that it is small enough to fit into primary memory limitations. Lam et al.[10] reports that among FM-index [6], CSA (Compressed Suffix Array) [19, 7] and Burrow-Wheeler transform (BWT), BWT is the most efficient compression algorithm and it is possible to reduce memory requirements to approximately *0.25n* bytes. This means that using BWT, the human genome can be compressed to fit into 1GB of memory which easily fits within the 4GB memory limitation in commodity computers. The indexing results of different tools are shown in Figure 1.

We find that algorithms those based on BWT, including BWT-SW, Bowtie, SOAP2 and BWA, create indexes in more than one hour, meanwhile occupying much more virtual memory (more than 3G). However, the final sizes of the indexes are quite close, which are slightly larger than original genome chromosomes, except for SOAP2, which is roughly twice. But indexing process runs only once, so the time complexity is not very important.

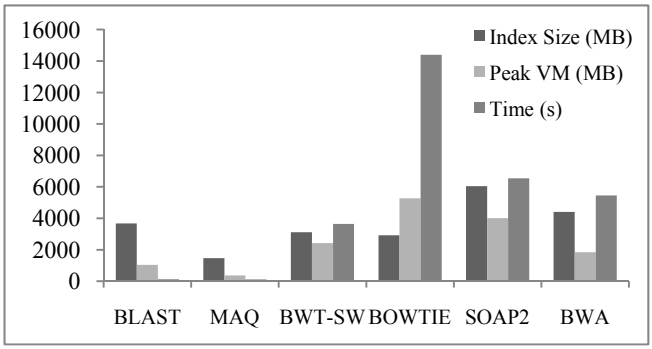

Figure 1. Indexing[4].

3.4 Parameter Flexibility

UASMAs have the ability to provide online SNP mapping service in real-time. Such a service must satisfy the following requirements to be efficient. It has to be responsive, flexible parameters and able to perform batch processing. However, our experiments found that none of the five components are perfect in every requirement. Table 7 looks into the parameter flexibility in each of them.

Table 7. Parameter Flexibility of components of UASMAs.

Tool	Latest Version	Server Process	# of mismatch	# of alignment	Gap	Paired-end	Multi-format	Multi-thread
MAQ	04/08	-	+	+	-	+	-	-
BWT-SW	09/07	+	-	-	+	-	+	-
Bowtie	01/10	-	+	+	-	+	+	+
SOAP2	08/09	-	+	-	-	+	-	+
BWA	11/09	-	+	+	+	+	-	+

BWT-SW will not be updated any more. Column "server process" indicates a process running in background, loading the index in memory already.

The server process is critical, since it will save much time spent in loading indexes into memory. BWT-SW supports server/client model, with server process performing alignment tasks and client process interacting with users. However, the server process will keep searching alignments even if the client side has aborted the query. When it is stuck, the server process does not response anymore until killed. In the experiments, BWT-SW hanged when processing certain queries. We doubt that some bugs exist. In addition, BWT-SW has stopped updating, with latest version released two years ago.

Currently the best performers seem to be Bowtie and BWA. Both Bowtie and BWA support flexible parameters for usage, although they do not support server process. Bowtie does not implemented gapped alignment, which is a disadvantage compared with BWA. But BWA outputs the alignment in SAM form [14].

3.5 Time and Space Complexity

Time and Space efficiency are big concerns. Currently, NCBI provides online alignment service based on BLAST. Processing time

[4] The indexes could be in different formats and the amounts of indexes files vary. The original genome size is 2996MB.

for a single query is about one minute. We claim that the system should response within 10 seconds for a single query, meanwhile occupying memory affordable for desktops nowadays. We can conclude from Table 8 that Bowtie outperforms others by around 50 times faster, meanwhile with the memory usage remaining acceptable.

Table 8. Time and Space Complexity on Chromosome Y.

Tool	Load Index (s)	Total Time (s)	Peak VM (MB)	Reads/sec
MAQ	-	~1200	~330	~60
BWT-SW	~40	-	-	-
Bowtie	~30	~35-80	~2400	~1800-2100
SOAP2	~120	~1000-2000	~5700	~30-70
BWA	-	~100-200	~2300	~500-700

Due to different query lengths, the time could slightly fluctuate. There are 72241 reads after pre-processing.

BWT-SW hangs somehow when performing batch processing. Even if we split the query file to 50 reads each, it is stuck so frequently that we are forced to abandon. In contrast, Bowtie runs amazingly fast, with around 2000 reads per second. Considering that it spends around 30 seconds to load the indexes into memory, we believe Bowtie will meet our requirement if the indexes have been pre-loaded beforehand. BWA ranks second, mainly because that its implementation of gapped alignment (we allow one-base difference here) slows down the speed. There seems no apparent index loading process for BWA. In addition, once the read length reaches 200bp, BWA's performance is degraded [13]. However, BWA is still a competitive candidate considering its speed (~500-700 reads/sec). MAQ and SOAP2 are equally slow, but SOAP2 consumes around 6GB, which is much larger than memory size of mainstream desktops 4G, due to its large-size indexes. MAQ achieves best memory usage, requiring only 330MB. This is mainly because MAQ compresses the indexes and handles the queries well.

We observe that Bowtie and SOAP2 spend much time in loading the indexes into memory, which could be saved if there is a server process running. MAQ's optimal memory usage is an advantage. It starts alignment immediately, making it a competitive candidate.

3.6 Recall and Precision

To achieve optimal performance in alignment, accuracy and reliability are always important. As we have mentioned in Section 3.2, even BLAST achieved 91% recall at most due to existence of outdated data. Figure 2, 3 and 4 are the experimental results of MAQ, Bowtie, SOAP2 and BWA. We ignore MAQ and BLAST here, because MAQ failed to deal with reads with lengths greater than 128bp, and the performance of BLAST drops dramatically (78% for 256bp) as the length of reads becomes large.

Generally, the recall of BLAST increases as the read length becomes larger while the precision decreases. But when it reaches length 256, both recall and precision drop dramatically. This trend shows that BLAST is not suited for long reads.

From Figure 2, we find that in general, at the beginning the recall increases as the length becomes longer, but when it reaches length 256, the value drops sharply. One possible explanation is that longer lengths provide higher confidence, but it can introduce difficulties in alignment. It seems that the lengths 76 and 128 indicate best tradeoff. All three of them are reliable since they can align majority of counted SNP sequences. The figures are close to or even better than BLAST. MAQ gets the worst results, with the value around 1~2 percent less. SOAP2 and Bowtie are winners, achieving slightly better results than BWA at every length except for 36bp.

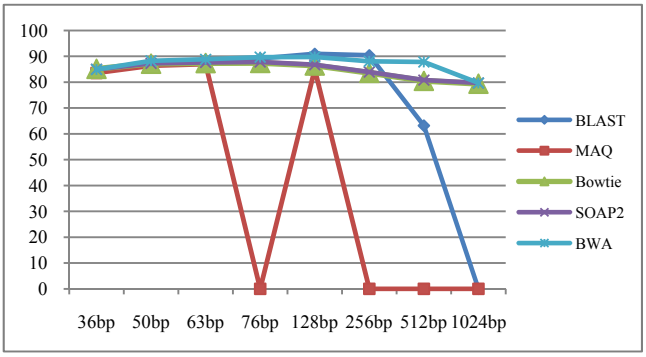

Figure 2. Recall of Alignment on Chromosome Y[5].

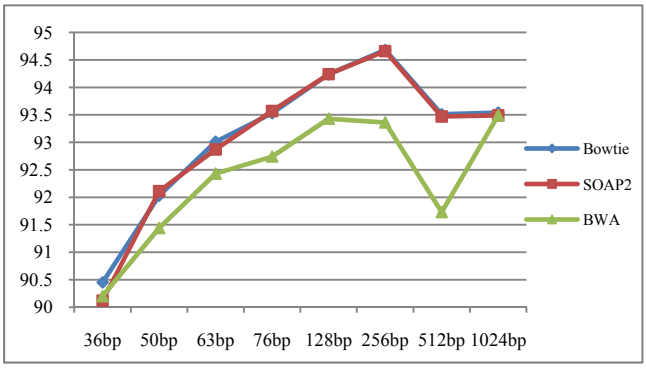

Figure 3. Precision of Exact Alignment on Chromosome Y.

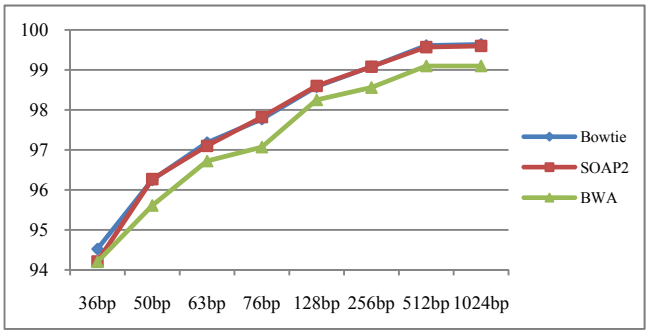

Figure 4. Precision of Loose Alignments on Chromosome Y[6].

The three candidates show consistent trends regarding precision: it rises steadily as the read length increases. This observation is consistent to the claim that longer lengths actually give more confidence to report an alignment. Of course, longer lengths mean the need for more time to process. MAQ outperforms the other two,

[5] MAQ was not able to deal with lengths longer than 128, meanwhile somehow it reported error for length 76. BWT-SW had no results to show.

[6] Loose alignment includes those alignments returning a position within 9 bps apart from the desired position.

although not much better. Bowtie and SOAP2 achieved competitive precision, which is around 1 percent better than BWA for all lengths.

In order to verify that there are no biases towards SNPs on chromosome Y, we conducted similar experiments based on chromosome 1 using Bowtie and BWA. The results are shown in Figure 5 and 6. They show that recall and precision are consistent to the results obtained from chromosome Y.

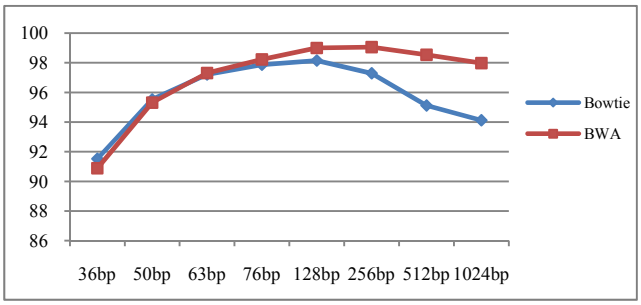

Figure 5. Recall of Bowtie and BWA on Chromosome 1[7].

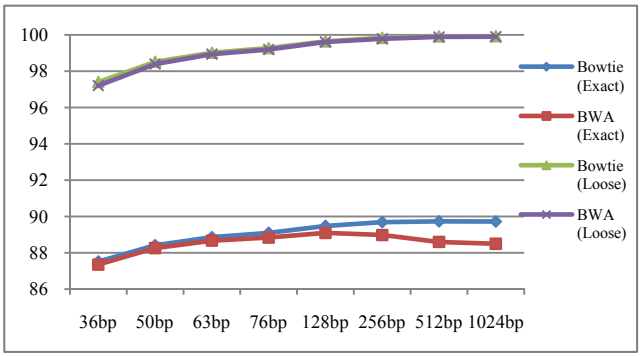

Figure 6. Precision of Bowtie and BWA Based on Chromosome 1.

Consistent to the previous results, for a larger dataset (SNPs on chromosome 1) of around 2.5 million SNPs, BWA achieves higher recall, especially for long lengths above 128bp. However, Bowtie again gets better precision. Recall Figure 3 and 4, we infer that Bowtie performs much better for exact matching. We think the reason for this outcome is that BWA implements gapped alignment, which will increase recall and lower precision.

4. DISCUSSION

Using UASMAs proves to be fast and optimal in mapping new variants onto the genome in view of depositing these entries accurately into a central database. Because it is done in real-time and with increased accuracy, recall and precision, the database created will be complete, up-to-date and devoid of ambiguities and redundancies. In addition, by knowing the exact location in the genome of newly discovered SNPs in real-time computed instantaneously, their proximities to important disease genes and important coding or non-coding regions can be deduced and their respective functionality inferred. High precision and recall makes algorithms used in UASMAs adept for large scale in-silico sequencing experiments to screen for functional SNPs in the genome, which are cheaper and faster than conventional high throughput sequencing technologies employing Illumina/Affimetrix chips.

Algorithms employed in UASMAs require local alignments to be done with high accuracy, speed and flexibility. We conducted experiments to compare the five components of UASMAs in terms of indexing, parameter flexibility, time complexity, space complexity, precision and recall. All have their shortfalls such as MAQ is less competent in dealing with long queries, BWT-SW seems to implement batch processing less well, Bowtie does not support deletion/insertion detection, SOAP2 is slow and occupies too much memory, and BWA may not be flexible enough. Using BLAST employed by NCBI as benchmark where recall was at most 91%, recall performance of components Bowtie and BWA were much better at up to 99% for longer reads. Similarly, Bowtie and BWA performed better in terms of precision at greater than 91 % whereas BLAST was only 78 – 88%. BLAST performed poorly in terms of recall and precision for longer reads. Bowtie and BWA algorithms in our tool were superior in terms of performances in alignment of longer sequences and locating the precise chromosome position of any SNP with respect to the NCBI reference assembly. Results obtained were fast, instantaneous and accurate. Although none of them gave perfect results, but considering that indel (insertion/deletion) of SNPs counts for as low as 1 per-cent, we think Bowtie and BWA gave excellent results. The most impressive advantage of Bowtie was that it was fast and accurate, which is critical for the efficient computational mapping of SNPs.

There are several functionalities that can be added to Bowtie to enhance its performance. Firstly, a back-end process can be kept running all the time, maintaining the indexes in main memory and interacting with client processes to avoid loading the indexes repeatedly. Secondly, gapped alignment could be implemented based on the solution of substitution. To allow mismatches, we substitute some bases during backtracking. Similarly, once we encounter a mismatch, we ignore the query base and move forward to consider next base, introducing an insertion. To bring in deletion, we could insert an extra query base into the current position, making SA range not empty in the next iteration. Another option is to integrate SW algorithm into Bowtie, which finds all local alignments satisfying the alignment rules.

5. CONCLUSION

We have proposed an effective and instantaneous method named UASMAs for accurately and efficiently mapping SNPs in real time in view of depositing the entries into a central database. UASMAs employed in the development of a tool in the form of a user-friendly graphical interface is in the pipeline. This tool can aid large scale in-silico screening for functional SNPs in genome wide experiments. The proposed tool can also eliminate all redundancies and inaccuracies whilst ensuring completeness of the included SNP entries. Our method serves to value-add to the existing systems of mapping and if proved robust, may be used as a standardized portal for mapping newly discovered SNPs in the near future.

Our work has compared the power of using the five alignment algorithms to simultaneously map and locate newly discovered SNPs onto the chromosome according to various criteria. A real-time web interface and server for public query submission are in the pipeline. Future work will also include dealing with other complex genomic variants such as tandem repeats, copy number variations, which are presently beyond the scope of this project.

[7] There are 2457326 reads after pre-processing.

6. ACKNOWLEDGEMENT

This work is supported by the Singapore Ministry of Education Academic Research Fund (AcRF) Tier 1 Grant T1 251RES0911. We would also like to acknowledge the contribution of Mr. Situ Yi in reviewing SNP nomenclatures and related algorithms.

7. REFERENCES

[1] Abouelhoda, M. I., Kurtz, S., and Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays. *J. of Discrete Algorithms*, 2(1):53-86.

[2] Altschul,S.F. et al. (1990) Basic local alignment search tool. *J. Mol. Biol*., 215,403–410.

[3] Barsky, M., Stege, U., Thomo, A., and Upton, C. (2008). A new method for indexing genomes using on-disk suffix trees. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 649-658, New York, NY, USA. ACM.

[4] Burrows, M, Wheeler., D. J. (1994). A block-sorting lossless data compression algorithm. *Technical Report 124, Digital Equipment Corporation, California*.

[5] Ferragina, P. and G. Manzini. (2000). Opportunistic data structures with applications. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science, IEEE Computer Society*.

[6] Ferragina, P. and G. Manzini. (2001). An experimental study of an opportunistic index. Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms. Washington, D.C., United States, *Society for Industrial and Applied Mathematics*.

[7] Grossi, R. and J. S. Vitter. (2005). Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *SIAM J. Comput*. 35(2): 378-407.

[8] Guttmacher, A. E. and F. S. Collins. (2002). Genomic medicine--a primer. *N Engl J Med* 347(19): 1512-20.

[9] John Healy, Elizabeth E. Thomas, Jacob T. Schwartz, and Michael Wigler. (2003). Annotating large genomes with exact word matches. *Genome Res*, 13: 2306-2315.

[10] Lam, T. W., W. K. Sung, et al. (2008). Compressed indexing and local alignment of DNA. *Bioinformatics* 24(6): 791-797.

[11] Langmead, B., C. Trapnell, et al. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol* 10(3): R25.

[12] Li, H, Ruan J, et al. (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*., 18, 1851–1858.

[13] Li,H. and Durbin, R. (2009a). Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics (Oxford, England)*, 25(14):1754-1760.

[14] Li et al. (2009b) SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, doi:10.1093/bioinformatics/btp336.

[15] Kurtz, S. and Schleiermacher, C. (1999). Reputer: fast computation of maximal repeats in complete genomes. *Bioinformatics*, 15(5):426-427.

[16] Manber, U. and Myers, G. (1990). Suffix arrays: A new method for on-line string searches. In SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms, pages 319-327, Philadelphia, PA, USA. *Society for Industrial and Applied Mathematics*.

[17] McCreight, E. M. (1976). A Space-Economical Suffix Tree Construction Algorithm. *J. ACM* 23(2): 262-272.

[18] National Center for Biotechnology Information. (2009). dbSNP Home Page. [Online] Available at: http://www.ncbi.nlm.nih.gov/projects/SNP/ [Accessed 10 February 2010].

[19] Sadakane, K. (2003). New text indexing functionalities of the compressed suffix arrays. *J. Algorithms* 48(2): 294-313.

[20] Sherry, S. T., M. H. Ward, et al. (2001). dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res* 29(1): 308-11.

[21] Smith, T. F. and M. S. Waterman. (1981). Identification of common molecular subsequences. *J Mol Biol* 147(1): 195-7.