Improved Gapped Alignment in BLAST

Michael Cameron, Hugh E. Williams, and Adam Cannane

***Abstract*— Homology search is a key tool for understanding the role, structure, and biochemical function of genomic sequences. The most popular technique for rapid homology search is BLAST, which has been in widespread use within universities, research centres, and commercial enterprises since the early 1990s. In this paper, we propose a new step in the BLAST algorithm to reduce the computational cost of searching with negligible effect on accuracy. This new step — *semi-gapped alignment* — compromises between the efficiency of ungapped alignment and the accuracy of gapped alignment, allowing BLAST to accurately filter sequences with lower computational cost. In addition, we propose an heuristic — *restricted insertion alignment* — that avoids unlikely evolutionary paths with the aim of reducing gapped alignment cost with negligible effect on accuracy. Together, after including an optimisation of the local alignment recursion, our two techniques more than double the speed of the gapped alignment stages in BLAST. We conclude that our techniques are an important improvement to the BLAST algorithm. Source code for the alignment algorithms is available for download at** `http://www.bsg.rmit.edu.au/iga/`.

***Index Terms*— Sequence alignment, BLAST, dynamic programming, homology search.**

I. Introduction

BLAST is the most well-known and popular bioinformatics tool. It is used to evaluate over 120,000 homology search queries each day [25] at the popular NCBI website[1], and is installed and maintained in almost all medium- to large-scale molecular biology research facilities. It has also been widely adapted for different platforms, architectures, and tasks.

The popularity of BLAST stems from its speed and accuracy. However, searches with BLAST become slower each year. In a study of the performance of BLAST, we have recently found that a query on the 2003 GenBank data using a 2003 Intel-based server takes an average of around 260 seconds. In 2001, the same task took only 83 seconds on a 2001 GenBank collection and 2001 hardware, and in 1999 only 36 seconds. Indeed, the trend is that BLAST is becoming around 64% slower each year because of the well-known exponential growth in genomic collections, and despite improvements in hardware [13]. It is therefore imperative that the fundamental algorithms in BLAST continue to be improved, and that new heuristics that improve speed without affecting accuracy are discovered.

Since 1997, there have been no fundamental changes to the BLAST algorithm. There has, however, been recent work on adapting BLAST techniques to other problems. For example, *spaced seeds* have been proposed as an improvement to the first stage of BLAST algorithm and included in several BLAST-like tools, such as BLAT [22], PatternHunter [24], and PatternHunter II [23]. However, there has been little focus on improving the final stages of BLAST that compute gapped alignments.

We have found that computing gapped alignments consumes an average of 32% of the total processing time in a BLAST search. Therefore, optimisation and innovation in the final stages of BLAST warrants investigation. This paper proposes two such innovations, *semi-gapped alignment* and *restricted insertion alignment*. Both reduce the computational cost of alignment with no significant effect on accuracy.

Semi-gapped alignment is a fundamental, new step in the BLAST algorithm. This step compromises between ungapped and alignment: insertions and deletions are permitted only every N residues, that is, gaps are allowed but not always at the optimal position. Semi-gapped alignment follows the ungapped alignment stage, aiming to efficiently and accurately reduce the number of gapped alignments required in the stage that follows. When carefully parameterised, this new technique reduces the time taken for the average alignment stage by 40% with no significant effect on accuracy.

Restricted insertion alignment is an heuristic that can be applied to semi-gapped or gapped alignment. We have observed that in optimal alignments, insertions in one sequence are very rarely adjacent to insertions in the other. By disallowing this event, it is possible to make a saving in computation following each alignment between two residues that occurs through insertion or deletion. This reduces the time taken for the gapped alignment stages in BLAST by around 8%. When semi-gapped alignment is used together with restricted insertion alignment and the optimisation we describe next, the speed of the gapped alignment stage in BLAST is more than doubled.

Our improvements to BLAST include an overlooked optimisation described by Zhang et. al. [37]. This optimisation was proposed to reduce the number of accesses to previously computed values. However, we have observed that this optimisation permits a rearrangement of the recurrence relation used to compute gapped alignments, reducing the number of arithmetic and comparison operations per alignment with no effect on the result. We describe and explain this optimisation, and show that it reduces the cost of the alignment stages by around 20%. The NCBI implementation of BLAST does not use this optimisation.

This paper is organised as follows. We overview homology search and the BLAST algorithm in Section II, describing in detail the process used to generate gapped alignments. Section III describes our two new gapped alignment techniques, and illustrate how they can be employed by BLAST. In Section IV, we present and discuss the results of comparing

M. Cameron, H.E. Williams and A. Cannane are with the School of Computer Science and Information Technology, RMIT University, GPO Box 2476V, Melbourne, Australia, 3001.

E-mail: {mcam,hugh,cannane}@cs.rmit.edu.au

[1]See `http://www.ncbi.nlm.nih.gov/`

our new techniques to the existing methods used by BLAST. Finally, Section V presents our conclusions and a discussion of planned future work.

II. Background

In this section, we present a short overview of homology search and explain the steps used by the BLAST algorithm to search genomic collections. We focus on detailing gapped local alignment, and discuss the BLAST dropoff heuristic that is used to reduce the search space for local alignments. Longer overviews of homology search and local alignment can be found elsewhere [1], [19], [32].

A. Homology Search

The widespread availability of computing resources and genomic collections has changed the approach molecular biologists use to characterise sequences. Fundamental to understanding the function of genomic sequences is determining common evolutionary ancestry, that is, finding *homology* between sequences. By comparing sequences and finding homology between two sequences — one of which has known function, structure, origin, or product — inference may be used as to the biochemical role, evolutionary history, and chemical structure of the second unknown sequence. Homologous sequences usually share common elements of three-dimensional folding and secondary structure; sometimes, however, homologous sequences do not share common function.

Homology is inferred when the statistical similarity between two sequences exceeds a threshold. The similarity is measured by finding similar regions in two sequences, usually through computing a *local alignment*, typically using a variant of the Smith-Waterman algorithm [34]. This estimation requires the measurement of the number of point mutations, or elementary changes, to transform a sequence region in one sequence into a region in another, and then expressing this as a probability that the score has arisen by chance [4]. This model of using specialised string comparison algorithms for genomic sequences has been shown to be an effective model of the evolutionary process [32].

To compute an optimal local alignment, all possible evolutionary pathways between two sequences are computed with respect to a scoring scheme. In practice, this requires tabulation of scores in a matrix of size $l_1 \times l_2$ for two sequences of lengths l_1 and l_2. In addition, to construct the evolutionary pathway and display this to a user, an additional matrix is required to store *traceback* information. The algorithm is therefore $O(n^2)$ is both time and space.

An example result from a local alignment between human and silkworm lysozyme sequences is shown in Figure 1. An optimal local alignment extending over 117 amino-acids is shown in the format typically returned to the user. Scoring of local alignment usually relies on a mutation data matrix [15], [20] and experimentally derived penalties for *gaps*. We explain the generalised local alignment approach that incorporates *gap* scoring and mutation matrices later in Section II-C.

B. The BLAST *Algorithm*

Without specialised hardware, Smith-Waterman local alignment is impractical for the comparison of a query to each sequence in a large genomic collection. This has necessitated heuristics to allow searches to be practical on desktop workstations, and to allow institutes to provide search services to large numbers of users. The FASTP algorithm [30] — which was later revised as FASTA [31] — was the first successful heuristic approach to local alignment. However, since 1990, BLAST [5] has been the most popular heuristic local alignment tool and in widespread use, first as the BLAST1 suite of tools, and since 1997 as the BLAST2 and PSI-BLAST tools [6].

The BLAST algorithm is a four-stage process that is efficient and effective for searching genomic databases. The steps progressively reduce the search space, but each is more fine-grain and takes longer to process each sequence. Table I shows the average time spent performing each stage of the algorithm. We briefly discuss the four stages in this section; more detail on the first two steps of BLAST can be found elsewhere [6], and the final two steps are discussed further in Sections II-C and II-D.

Stage 1: In the first stage, each subject sequence y from the collection of sequences being searched is retrieved and compared to the query sequence x using the algorithm of Wilbur and Lipman [35]. This identifies high-scoring matches between fixed-length overlapping subsequences (or *words*) of length W extracted from the query and subject sequences. Typically, $W = 2$, $W = 3$, or $W = 4$ for amino-acid search. These matches are referred to as *hits*, and the offset i from x and j from y of each hit is passed to the second stage of the algorithm. In practice, BLAST uses an efficient lookup table from the query sequence to identify hits for each subject sequence.

To examine the performance of this first stage — and produce the results shown in Table I — we carried out a simple experiment. We randomly extracted 100 sequences from the GenBank non-redundant (NR) protein database [8] and then searched the entire database using each as query. We found that the first stage consumed on average around 37% of the total time for all four stages, and that an average of 229 hits were found per collection sequence; almost all sequences had at least one hit. In total, the non-redundant database we used — which is discussed further in Section IV — contained 1,873,745 sequences, and an average of 332 residues per sequence.

Stage 2: The second stage determines whether two or more hits h of length W could form the basis of a local alignment that does not include insertions or deletions of residues. To determine this, the *diagonal* p of each of the hits is determined by computing the difference in the query and subject sequence offsets, $p = j - i$. If two hits, $h[i_1, j_1]$ and $h[i_2, j_2]$, are found to occur on the same diagonal (since $j_1 - i_1 = j_2 - i_2$) and $i_1 - i_2$ is less than a constant A, then an *ungapped extension* is attempted. The parameter A influences the accuracy of BLAST, and is one of many parameters that can be tuned in the algorithm.

The left side of Figure 2 illustrates the first two stages of the algorithm. The figure shows a matrix used to compute the

```
1GD6:   1 KTFTRCGLVHELRK---HGFEENLMRNWVCLVEHESSRDTSKTNTNR-NGSKDYGLFQIN 56
          K F RC L   L++    G+    + NW+CL + ES  +T  TN N  + S DYG+FQIN
1LZ1:   1 KVFERCELARTLKRLGMDGYRGISLANWMCLAKWESGYNTRATNYNAGDRSTDYGIFQIN 60

1GD6:  57 DRYWCSKGASPG--KDCNVKCSDLLTDDITKAAKCAKKIYKRHR-FDAWYGWKNHCQ   110
           RYWC+ G +PG    C++ CS LL D+I  A  CAK++ + +   AW  W+N CQ
1LZ1:  61 SRYWCNDGKTPGAVNACHLSCSALLQDNIADAVACAKRVVRDPQGIRAWVAWRNRCQ   117
```

Fig. 1. Smith-Waterman alignment of silkworm lysozyme (PDB accession 1GD6, 119 amino-acids) and human lysozyme (PDB accession 1LZ1, 130 amino-acids). A BLOSUM62 mutation matrix is used, with gap open penalty of 11 and extension penalty of 1. The Smith-Waterman score is 215, with 41% identity in a 117 amino-acid overlap. The '+' in the middle line indicates a conservative substitution, whilst a capital letter indicates an identity.

TABLE I

AVERAGE RUNTIME FOR EACH STAGE OF THE BLAST ALGORITHM FOR 100 RANDOMLY-SELECTED SEQUENCES FROM THE GENBANK NON-REDUNDANT DATABASE SEARCHED AGAINST THE ENTIRE DATABASE. EXPERIMENTS WERE CONDUCTED USING THE NCBI IMPLEMENTATION OF BLAST AND DEFAULT PARAMETERS, INCLUDING $W = 3$, $T = 11$, $S1 = 22$ BITS, UNGAPPED DROPOFF OF 7 BITS, GAPPED DROPOFF OF 15 BITS AND BLOSUM62 SUBSTITUTION MATRIX WITH GAP PENALTIES OF 11 AND 1.

Stage	Task	Percentage of overall time
1	Find high-scoring short hits	37%
2	Identify pairs of hits on the same diagonal	18%
2	Perform ungapped extensions	13%
3	Perform gapped extension	30%
4	Collect traceback information and display alignments	2%

similarity between two sequences. The width of the matrix is the length l_1 of the query sequence x, that is, there is one column for each residue in the query sequence. Similarly, the height of the matrix is the length l_2 of the subject sequence y that is being considered. Therefore, each cell in the matrix represents an intersection between a residue from each sequence, and is used to tabulate a score on the optimal evolutionary pathway that considers those two residues. This process is discussed in more detail in Section II-C.

The short black lines in Figure 2 represent high-scoring hits of length W that match between both sequences. That is, each such black line represents the beginning of the shortest possible evolutionary pathway (of minimum length W) that is considered by the BLAST algorithm. In the figure, there are two cases where two hits are located on the same diagonal less than the maximum distance A apart. For each of these pairs, an ungapped extension is performed to determine if the hits are likely to form part of a high-scoring alignment, and the region covered by the extension is illustrated by a grey line. If an ungapped extension scores above the value of $S1$ — another constant determined by an external parameter — it is considered successful and is passed on to the third stage of the algorithm. In this example, the longer extension scores above $S1$, while the shorter does not.

We measured the performance of the second stage, using GenBank NR and the same 100 randomly-selected queries described previously. On average, 9.8 ungapped extensions are performed per collection sequence, but less than 0.01% of these produce a score above the cutoff, $S1$. The effect is that around 11% of the database sequences are passed on to the third stage. The second stage consumes on average 31% of the total search time.

Stage 3: In the third stage, a gapped alignment is performed to determine if the high scoring ungapped region forms part of a larger, higher scoring alignment. The right side of Figure 2 illustrates an example where this is the case: the single, high-scoring ungapped extension identified in Stage 2 is considered as the basis of a gapped alignment, and the black line shows the alignment identified through this process. The gapped alignment process is described in more detail in Section II-C.

The gapped alignment algorithm used by BLAST differs from Smith-Waterman local alignment. Rather than exhaustively computing all possible paths between the sequences, the gapped scheme explores only insertions and deletions that augment the high-scoring ungapped alignment. Therefore, this step begins by identifying a *seed* point that lies within a high-scoring portion of the ungapped region. After this, a gapped alignment is attempted. We use the term *gapped alignment* to refer to the approach used by BLAST and *local alignment* to refer to the exhaustive Smith-Waterman approach.

A gapped alignment stops when the score falls below a value determined by a dropoff parameter, X. This parameter controls the sensitivity and speed trade-off: the higher the value of X, the greater the alignment sensitivity but the slower the search process. If the resulting gapped alignment scores more than $S2$ — which is determined from an external E-value cutoff parameter — it is passed on to the fourth and final stage of BLAST. On average, in our experiment with GenBank NR described previously, we found that less than 0.01% of the gapped alignments performed during the third stage score above the default E-value cutoff of 10, and that this stage consumes on average 30% of the total search time.

Stage 4: In the final stage of the BLAST algorithm, the final alignments to be displayed to the user are rescored. During the rescoring, the alignment traceback pathway itself is recorded so that it can be displayed in the format shown in Figure 1; the third stage records only scores, and not the evolutionary pathway that leads to that score. The other important difference during rescoring is that the value of the dropoff parameter, X,

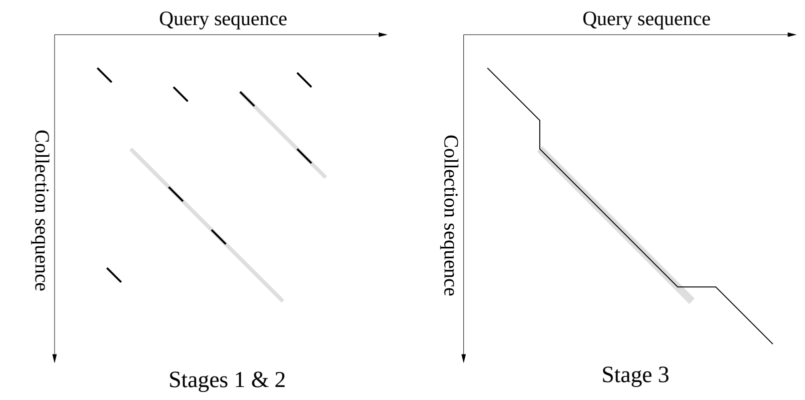

Fig. 2. Illustration of the first three stages of the BLAST algorithm. In stages 1 and 2, short high-scoring alignments or *hits* are identified, shown as short black lines. When two hits occur near each other and on the same diagonal an ungapped extension is performed, with the result shown as a longer grey line. In this example, the longer of the two ungapped extensions scores above $S1$ and is passed on to stage 3, where it is used as a starting point for constructing a higher-scoring gapped alignment.

is increased to attempt to find a higher scoring alignment.

The number of alignments processed during this final stage of the algorithm is limited by two important factors. First, it is determined by the number of sequences found to score above the E-value cutoff in the third stage of the algorithm. Second, the value of the parameter B — which defaults to 500 — limits the total number of alignments to be displayed to the user. Around half of the 100 queries we evaluated in our experiment with GenBank produced more than 500 high-scoring alignments. In this case, only the 500 highest scoring alignments from the third stage are rescored in this stage and subsequently displayed to the user. On average, the final stage consumes only 2% of the total search time.

Statistics: BLAST uses Karlin-Altschul statistics [4], [21] to report the statistical significance of each gapped alignment to the user as an E-value. The E-value represents the chance that an alignment with at least the same score would be found if a randomly constructed query with typical amino-acid composition was searched against a randomly generated database. The length of the query sequence and the size of the collection are taken into consideration.

There are three stages to determine the statistical significance of a gapped alignment:

1) A nominal score S is determined for each alignment using a mutation data scoring matrix — typically either a BLOSUM [20] or PAM [15] matrix — and a function for penalising gaps. Nominal scores are generally written without units.
2) The nominal score is converted to a normalized score S' as follows:
$$S' = \frac{\lambda S - ln\ K}{ln\ 2}$$
where the values of λ and K are pre-computed by random simulation for each scoring matrix and gap penalty combination. Scores in this normalized form are expressed in bits, and are comparable across different scoring schemes.
3) The normalized score is converted into an E-value as follows:
$$E = \frac{Q}{2^{S'}}$$
where Q is the search space size, that is, $Q \approx mn$ where m and n are the total number of residues in the query and the collection respectively. BLAST uses two additional pre-computed values α and β to calculate the exact value of Q, taking into consideration that high-scoring alignments generally cannot start a short distance from the end of either sequence. This *edge effect correction* is discussed in more detail elsewhere [3].

The resulting value of E is reported to the user as the alignment E-value. Note also that the equations can be inverted to determine the minimum nominal score required to achieve a specific E-value. This approach is used by BLAST to determine the cutoff parameter $S2$ from the user-specified E-value.

C. *Gapped alignments*

In this paper, we present two improvements to the gapped alignment technique that is used for the third stage of the BLAST algorithm. However, before we introduce these new techniques, we describe the gapped alignment algorithm that is employed by BLAST in more detail.

BLAST uses the popular Gotoh algorithm [17] to produce gapped alignments that employ an *affine gap cost* model. Under this model, three possible evolutionary events can occur with respect to each possible pair of residues drawn from two sequences x and y: first, the residues are aligned (meaning the residues are conserved because they are the same, or one is substituted for the other); second, an insertion is made with respect to y; and, last, an insertion is made with respect to x. As discussed previously, the first class of event is scored using a mutation data matrix. The latter two classes of event are scored with the affine gap model, where the penalty for beginning an insertion is typically high but the penalty for continuing it as a *gap* is low. Specifically, the cost c of a gap of length k is defined by $c(k) = k \times e + o$, where o is the cost of opening and e the cost of extending a gap, and

$e > 0,\ o > 0,\ k \geq 1$. We add to this definition the pre-computed cost d of opening a gap and the first insertion in that gap, that is, $d = o + e$.

As discussed previously, the alignment recursion itself requires a matrix of size $l_1 \times l_2$ for two sequences x and y of lengths l_1 and l_2 respectively. Each cell $[i, j]$ in the matrix represents the highest scoring alignment between x and y that ends with the i^{th} residue in x and the j^{th} residue in y. The value in each cell $[i, j]$ is dependent on three of the immediate neighbours of the cell at coordinates $[i-1, j-1]$, $[i-1, j]$, and $[i, j-1]$, and these map to the three possible evolutionary events that can affect the alignment of the i^{th} residue in x and the j^{th} residue in y. This dependence is illustrated in Figure 3, where character matches are represented by diagonal arrows and insertions with respect to y and x are represented by horizontal and vertical arrows respectively.

To perform the gapped alignment, three values must be recorded for each cell $[i, j]$. The score $B(i, j)$ is the best score for any alignment ending at $[i, j]$, the score stored at $I_x(i, j)$ represents the best score for an alignment ending at $[i+1, j]$ with an insertion with respect to y, and $I_y(i, j)$ represents the best score for an alignment ending at $[i, j+1]$ with an insertion with respect to x. Using these values, the following recurrence relations can be employed to compute the score of the optimal gapped alignment between two sequences using affine gap costs:

$$(1) \qquad M(i,j) = B(i-1, j-1) + s(x_i, y_j)$$

$$(2) \qquad B(i,j) = \max \begin{cases} I_x(i-1, j) \\ I_y(i, j-1) \\ M(i,j) \end{cases}$$

$$(3) \qquad I_x(i,j) = \max \begin{cases} M(i,j) - d \\ I_x(i-1, j) - e \end{cases}$$

$$(4) \qquad I_y(i,j) = \max \begin{cases} M(i,j) - d \\ I_y(i, j-1) - e \end{cases}$$

where $s(x_i, y_j)$ is the score resulting in a match between the i^{th} character of x and the j^{th} character of y, and the scalar value M represents the best score for an alignment ending at $[i, j]$ with a match. In addition to these recurrence relations, initialisation rules are required to handle boundary conditions; typically, all cells where $i = 0$ or $j = 0$ are initialised to $-\infty$, except for the alignment starting point $[0, 0]$ which is initialised to zero.

These rules are not exactly as used by BLAST. Rather, they include the important optimisation described by Zhang et. al. [37] where $I_x(i, j)$ and $I_y(i, j)$ store scores for $[i+1, j]$ and $[i, j+1]$ respectively. The motivation of this is to reduce accesses to previously computed values. However, there is an important additional advantage that — to our knowledge — has not been previously observed: it permits rule reorganisation that can reduce the computation required to produce gapped alignments, leading to the savings in computation that we discuss next. The optimisation is not included in NCBI BLAST. We use it in our baseline implementation and new schemes.

The rules of Zhang et. al. [37] afford a performance advantage over the original BLAST approach in two ways. First, the processing of each cell requires four instead of five arithmetic operations, since the value of $M(i, j) - d$ in equation (3) can be reused in equation (4). Second, the number of comparisons that need to be performed can be reduced from four to three for some cells. To illustrate this, consider the case where the computation of $B(i, j)$ reveals that $I_x(i-1, j) \geq M(i, j)$, that is, when $I_x(i-1, j)$ is found to be the largest of the three values in equation (2). In this event, it can be deduced that $I_x(i-1, j) - e > M(i, j) - d$, avoiding the need to compare these two values when calculating $I_x(i, j)$ in equation (3). The same approach can be used when calculating $I_y(i, j)$ in equation (4).

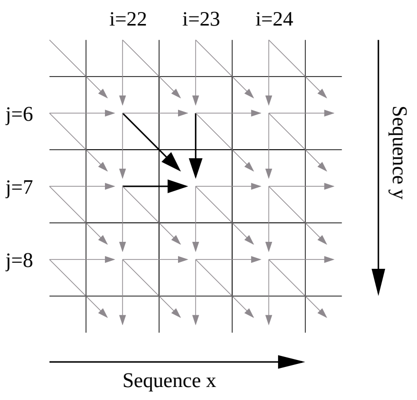

Fig. 3. Portion of the dynamic programming matrix used to perform a gapped alignment. Diagonal arrows represent character matches, while vertical and horizontal arrows represent insertions with respect to x and y respectively. The values for $i = 23$ and $j = 7$ are dependent on the three immediate neighbours at coordinates $[22, 7]$, $[22, 6]$ and $[23, 6]$.

The highest gapped alignment score is found by recording the highest $B(i, j)$ value that is computed. However, if the alignment itself is required for display to the user — as in the fourth stage of BLAST — the algorithm must be modified. The modifications required include recording information about how scores are derived for each cell in the matrix, so that a traceback can be performed to find the path resulting in the optimal score. Optimising the storage of traceback information, and reconstructing tracebacks using linear space, is a well-understood problem that is described elsewhere [11], [27]; as our novel schemes address improving the performance of the third stage of BLAST, we do not consider this problem further. Indeed, it is unclear how important this problem is in practice: as sequences grow in number but not in length, it is execution speed rather than main-memory requirements that is the limiting factor for searching current collections on modern hardware.

The variation of Gotoh's algorithm that we have described is suitable for finding the optimal gapped alignment passing through a given starting point. For BLAST, this start point is a seed point determined from its previous steps. In contrast, the rigorous Smith-Waterman algorithm constructs local alignments that allow the alignment to begin at any point in the dynamic programming matrix. This is achieved by a small adjustment to the recursive relations that disallows negative values.

Disallowing negative values has been used to optimise the

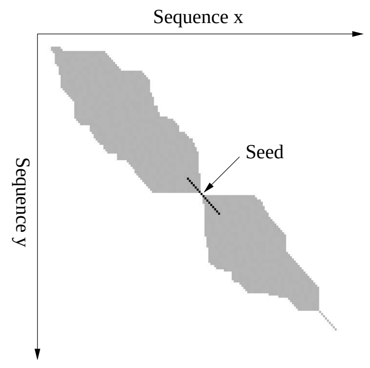

Fig. 4. Illustration of a seeded gapped alignment with dropoff. The black line represents a high-scoring ungapped alignment between sequences x and y. A point on the alignment is selected as the seed and a gapped extension is performed in each direction, beginning at that point. The grey regions represent cells in the alignment matrix that exceed the best score seen so far, less a constant dropoff value.

Smith-Waterman algorithm and almost double its execution speed [28]. However, unfortunately, this optimisation is not applicable to the gapped alignment technique used by BLAST. Myers and Durbin [28] have also recently investigated using a table lookup approach to increasing the speed of the Smith-Waterman algorithm and found they could achieve a speed increase of up to 1.5 times when using the popular BLOSUM62 data mutation matrix (and up to twice as fast with other matrices). However, this scheme also relies on the sparsity of high scoring cells in the matrix, again typically not the case for the gapped alignment stage in BLAST.

D. Seeded Gapped Alignment with Dropoff

As we described previously, when a high scoring ungapped alignment is found during the second stage of a BLAST search, a gapped alignment is performed. However, BLAST does not process the entire dynamic programming matrix associated with aligning the two sequences. Instead, it considers only gapped alignments that pass through a seed point $[a, b]$, which is chosen as the centre residue of a high scoring region within the ungapped alignment; it is unclear if this is the best approach to choosing a seed point and we plan to investigate other choices in future work. The algorithm then performs a *gapped extension* beginning at the seed, and proceeding in each direction, towards both the start and end of the sequences. The final alignment score is the sum of the scores resulting from extension in each direction.

BLAST uses a *dropoff* technique [36] to further restrict the number of cells processed in local alignment. To do this, BLAST considers only regions of the alignment matrix with a score $B(i, j)$ that is greater than the best score found so far minus the value of a dropoff parameter, X. Figure 4 illustrates how this technique limits the area of the matrix to be processed: the grey shaded region represents cells that are locally aligned in the third and fourth stages of BLAST as a gapped extension is performed in each direction from the seed point.

The dropoff scheme is effective in reducing the computational cost of local alignment, while being sensitive in finding homologous sequences. When two sequences share a small region with a low alignment score, the alignment scores typically fall towards zero within a few cells as a gapped extension is attempted. This limits the area of the matrix that needs to be processed. Indeed, since the majority of gapped extensions are triggered by ungapped alignments that are not in fact part of a higher scoring gapped alignment, the dropoff technique is highly effective in reducing the number of cells that need to be be computed in local alignment. In our experiment with 100 queries and the GenBank collection, we found that on average less than 2% of all cells in the matrix are processed when the dropoff technique is applied using default parameters.

The dropoff technique used in BLAST is similar to the alignment within a fixed diagonal band [12] heuristic that is employed by FASTA. Both techniques are effective in avoiding processing cells in the alignment matrix. However, we believe that the dropoff technique is the more effective technique because it is adaptive, varying both the direction and size of the region to be computed based on the alignment scores determined so far. We have not experimented with fixed diagonal regions in our work.

III. A New Approach to Gapped Alignment

In this section, we propose two novel gapped alignment algorithms. The first is used as a new stage in the BLAST algorithm that filters alignments between the second and third stages. The second aims to reduce the time taken to generate gapped alignments in the third stage. Both of the novel algorithms differ from traditional gapped alignment by reducing the computation required for each cell in an alignment matrix. This approach is orthogonal to the dropoff technique already employed by BLAST, which instead limits the number of cells that need to be processed. Therefore, a significant advantage of our novel techniques is that they can be used in combination with the existing dropoff heuristic. We report results of integrating our alignment scheme into the final stages of BLAST in Section IV.

A. Semi-gapped alignment

In this section, we propose a new *semi-gapped* algorithm that compromises between the speed of ungapped alignment and the accuracy of gapped alignment. Our motivation is to add a new stage in the BLAST algorithm that efficiently and accurately reduces the subset of sequences identified by ungapped alignment to a very small set that are subsequently aligned using the computationally expensive gapped alignment stage. Specifically, we aim to reduce the computation at each cell in the alignment matrix while still producing alignment scores similar to those found using the Gotoh algorithm.

Our idea is to restrict where insertions and deletions can occur in an alignment, leading to an approach that combines the features of fast, heuristic ungapped alignment and the slower, more rigorous gapped alignment. The basic approach is as follows: we allow insertions in sequence y only at every N^{th} character (where $j \equiv 0, \mathrm{modulo}\, N$) and insertions in sequence x at every N^{th} character (where $i \equiv 0, \mathrm{modulo}\, N$).

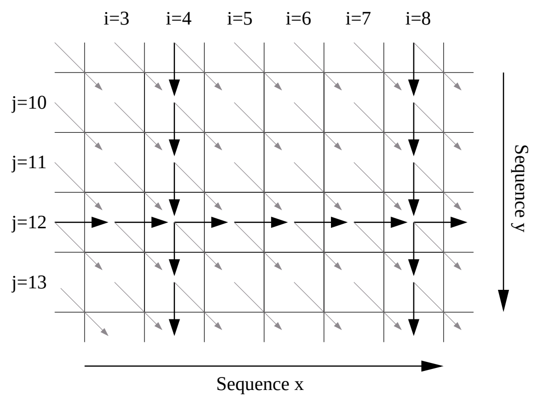

Fig. 5. Portion of the dynamic programming matrix used to perform a semi-gapped alignment where $N = 4$. Arrows indicate how values are derived for each cell. For most cells only the match operation is considered. For columns where $i = 4$ or $i = 8$, insertions with respect to x are also considered. Similarly, insertions with respect to y are also considered for the row where $j = 12$.

Figure 5 illustrates the effect these two constraints have on how values are derived for each cell in the matrix. We explain the rationale behind this approach next, and discuss the reduction in computational cost and the effect of varying N later.

We propose that semi-gapped alignments be attempted only after an ungapped alignment exceeds a cutoff, and that a successful semi-gapped alignment be used to trigger a further gapped alignment as the final stage of BLAST. We believe that this semi-gapped alignment stage should be an effective and efficient additional step in BLAST for three reasons:

1) We have observed that most gapped alignments calculated using affine gap costs contain long ungapped regions separated by gaps of more than one indel that move the alignment from one matrix diagonal to another; semi-gapped alignment still permits these gaps, but forces the gap start and end points to be moved. In the worst case, the optimal position for opening the gap is midway between rows or columns where the insertion is allowed, and the gap position must be moved by $\lfloor \frac{N}{2} \rfloor$ residues from this optimal position. On average, a move of $\frac{\lceil \frac{N}{2} \rceil \times \lfloor \frac{N}{2} \rfloor}{N}$ residues is required. Unless the optimal gap is surrounded by high-scoring matches or the neighbouring ungapped regions are flanked by low-scoring matches, this shift has little influence on the overall score.
2) The scoring penalty associated with opening and extending a gap often outweighs any reduction in score that may be caused by opening the gap at a sub-optimal location.
3) It has been observed by Altschul [2] that when aligning distantly related protein sequences, "conserved residues frequently fall into ungapped blocks separated by relatively non-conserved regions". This suggests that gaps often occur in less conserved regions and that the exact location of the gap is of less importance, given that the surrounding matches are likely to be low scoring anyway.

We now explain how the constraints imposed by our semi-gapped scoring technique affect the recurrence relations and the processing required for each cell in the alignment matrix. Observe that, depending on its location in the matrix, each cell in a semi-gapped alignment permits one of four cases:

1) Insertion in either the query or subject sequence is permissible, that is, the standard gapped alignment recursion applies; or,
2) Only insertion with respect to the query is allowed; or,
3) Only insertion with respect to the subject is allowed; or,
4) No insertion is permissible, that is, the standard ungapped alignment recursion applies.

Therefore, different recurrence relations are applicable depending on what operations are permitted. We consider these operations with respect to the four cases next.

The semi-gapped recurrence relations are shown in Table II. The original recurrence described in Section II is used for the first case, that is, for cells where $j \equiv 0, \text{modulo}\, N$ and $i \equiv 0, \text{modulo}\, N$, and where insertion in both sequences is allowed; these relations are shown at the intersection of the row labelled "I_y allowed" and the column "I_x allowed". When insertion in sequence y is not permitted, that is, $j \not\equiv 0, \text{modulo}\, N$ and the second case applies, then there is no need to compute the value of I_x and the recursion is simplified to that shown at the intersection of the row labelled "I_y not allowed" and the column labelled "I_x allowed". Similarly, when $i \not\equiv 0, \text{modulo}\, N$, then the third case applies and there is no need to compute the value of I_y and the recursion is simplified to that at the intersection of the row "I_y allowed" and the column "I_x not allowed". Last, when only ungapped alignment is permitted, the fourth case applies and the simple recursion at the intersection of ""I_y not allowed" and "I_x not allowed" is used.

Computational Costs: Table III shows the number of arithmetic operations and comparisons required for each of the four different recurrence relations. In addition, it shows how varying N affects the number of cells in an alignment matrix that are computed using each of the four relations. For example, the table shows that in the regular gapped alignment case — shown as the intersection of "I_x allowed" and "I_y allowed" — four arithmetic operations are required per cell, and either three or four comparisons (as described for the optimisation in Section II-C) are required. The table also shows that only 1 in every N^2 cells require this fully-gapped alignment. For the cases where I_x is allowed but I_y is not, and where I_y is allowed but I_x is not, only three arithmetic operations and two comparisons are required. We can also apply optimisations similar to those used in the regular gapped alignment case that reduces the number of comparisons for some cells further, from two down to one. Importantly, ungapped alignment is inexpensive — requiring only one arithmetic operation — and for $N \geq 4$ the majority of cells are in this class.

When $N \geq 2$, an additional overhead is required to determine the class of each cell. Since the value of j does not change while processing each row in the matrix, there is no need to determine for each cell whether insertion in sequence y is allowed; this can be done once for each row at negligible cost. In contrast, the value of i must be checked for each cell

TABLE II

RECURRENCE RELATIONS FOR EACH CELL TYPE IN SEMI-GAPPED ALIGNMENT.

	I_x allowed	I_x not allowed
I_y allowed	$M(i,j) = B(i-1,j-1) + s(x_i,y_j)$ $B(i,j) = \max \begin{cases} I_x(i-1,j) \\ I_y(i,j-1) \\ M(i,j) \end{cases}$ $I_x(i,j) = \max \begin{cases} M(i,j) - d \\ I_x(i-1,j) - e \end{cases}$ $I_y(i,j) = \max \begin{cases} M(i,j) - d \\ I_y(i,j-1) - e \end{cases}$	$M(i,j) = B(i-1,j-1) + s(x_i,y_j)$ $B(i,j) = \max \begin{cases} I_y(i,j-1) \\ M(i,j) \end{cases}$ $I_y(i,j) = \max \begin{cases} M(i,j) - d \\ I_y(i,j-1) - e \end{cases}$
I_y not allowed	$M(i,j) = B(i-1,j-1) + s(x_i,y_j)$ $B(i,j) = \max \begin{cases} I_x(i-1,j) \\ M(i,j) \end{cases}$ $I_x(i,j) = \max \begin{cases} M(i,j) - d \\ I_x(i-1,j) - e \end{cases}$	$B(i,j) = B(i-1,j-1) + s(x_i,y_j)$

TABLE III

FREQUENCY OF EACH TYPE OF CELL IN A SEMI-GAPPED ALIGNMENT AND NUMBER OF ASSOCIATED OPERATIONS.

	I_x allowed		I_x not allowed	
I_y allowed	Arithmetic operations =	4	Arithmetic operations =	3
	Comparisons =	3 or 4	Comparisons =	1 or 2
	Cell type frequency =	$\frac{1}{N^2}$	Cell type frequency =	$\frac{N-1}{N^2}$
I_y not allowed	Arithmetic operations =	3	Arithmetic operations =	1
	Comparisons =	1 or 2	Comparisons =	0
	Cell type frequency =	$\frac{N-1}{N^2}$	Cell type frequency =	$\frac{(N-1)^2}{N^2}$

TABLE IV

AVERAGE NUMBER OF OPERATIONS PER CELL FOR VARYING VALUES OF N.

N	1	2	3	4	5	6	7	8	9	10	11	12
Arithmetic operations	4.00	3.75	3.22	2.94	2.76	2.64	2.55	2.48	2.43	2.39	2.36	2.33
Comparisons	4.00	3.00	2.33	2.00	1.80	1.67	1.57	1.50	1.44	1.40	1.36	1.33

individually to determine if $i \equiv 0, \text{modulo}\, N$. This involves an additional arithmetic operation and comparison per cell to determine its class.

Table IV illustrates how the average number of operations per cell varies as N is increased from 1 to 12, including the overhead of determining the class of each cell. When $N = 1$, only gapped alignment is used and, as N increases, the average computational cost per cell decreases. However, the reduction in computational cost as N is increased has an effect on accuracy. Therefore, similarly to other parameters in BLAST, the value of N must be carefully chosen. We report experiments with varying values of N in Section IV.

Triggering Gapped Alignment: Semi-gapped alignment is an additional stage in BLAST that follows ungapped alignment and precedes gapped alignment. Therefore, similarly to all other non-final stages, a criterion must be established to trigger processing of an alignment in a subsequent stage.

After experimentation, we found the following approach is effective for deciding whether semi-gapped alignments should be passed to the final, gapped alignment stage in BLAST. We score each candidate sequence using semi-gapped alignment and, if the score exceeds $R \times S2$, then we proceed to local alignment. The value of $S2$ is the existing BLAST nominal score required to achieve cutoff, and R is a new constant such that $0 < R \leq 1$. Using the existing $S2$ constant has an important advantage: if the score from semi-gapped alignment exceeds $S2$, then there is no requirement for gapped alignment and the sequence can be passed to stage four of the BLAST algorithm where the refined score and traceback information is determined. We report experiments that vary R in Section IV.

Figure 6 illustrates how the semi-gapped alignment stage is incorporated into the BLAST algorithm. First, an ungapped extension is performed and the resulting alignment (shown at the left of the figure) scores above an $S1$ cutoff score of 40. Second, a semi-gapped alignment is performed as shown in the middle of the figure. The light grey lines highlight the columns and rows in the semi-gapped alignment matrix where insertions are allowed; we show that the two gaps in the alignment occur in these permitted regions. The resulting score of 87 is recorded, and because this lies between $R \times S2 = 63$ and $S2 = 90$, a gapped alignment is required to determine if the alignment scores more than $S2$. The resulting gapped alignment (show at the right of the figure) scores 95, and is therefore significant enough for display to the user.

B. Restricted insertion alignment

In this section, we describe our second novel technique to improve gapped alignment in BLAST. This technique — which

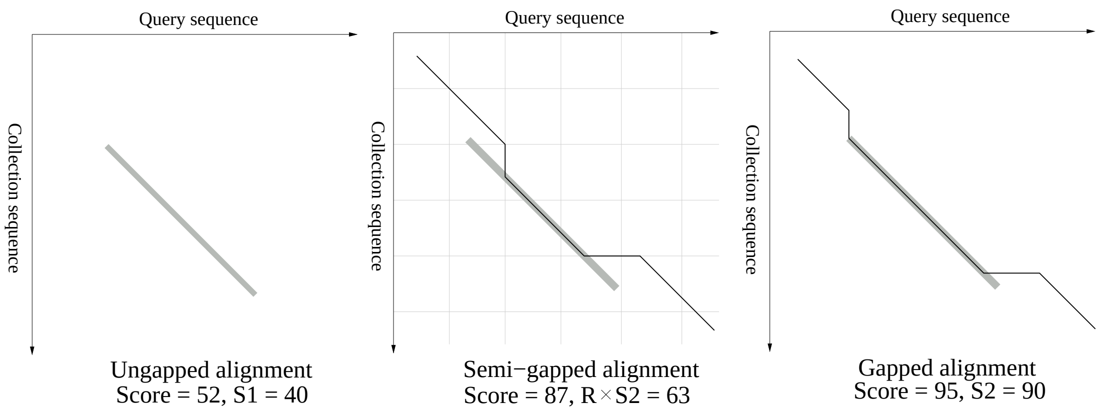

Fig. 6. The new process for scoring sequences. First, an ungapped extension is performed (left). If the resulting alignment scores above the $S1$ cutoff, a semi-gapped alignment is then performed (middle). The grid overlay shows the rows and columns where insertions are permitted. Finally, if the semi-gapped alignment scores between $R \times S2$ and $S2$ then a gapped alignment is performed (right). In this example, the alignment scores above the cutoff at each stage, and its final score is high enough for it to be displayed to the user.

```
Query:   75 LIARSAFGDLYLWGEEIGASLKITSI 100
              +   SAFG + LW E+     K TSI
Sbjct:   70 ALGFSAFGKILLWNED----YKTTSI 91

Query:   12 NQDDYYI-------DPEGRFFIVADG 30
            N+D YY                +VADG
Sbjct:  169 NEDTYYAGRFSLGD-------LVADG 187
```

Fig. 7. Two example pairwise alignments. The top shows an example of an alignment that contains a single gap. The bottom shows an example of the infrequent case where an alignment contains two adjacent gaps.

we refer to as *restricted insertion alignment* — is orthogonal to semi-gapped alignment, and can be applied either to it or to the gapped alignment stage in BLAST.

Restricted insertion alignment aims to reduce computation for unlikely evolutionary events. We have experimentally observed that, in optimal gapped or local alignments, insertions in one sequence are very infrequently adjacent to insertions in the other. Figure 7 shows examples of gaps in sequences. To the left of the figure, the typical case of gap insertion is shown: a gap exists in one sequence and contributes to an optimal alignment. The right of the figure illustrates the rare case: two gaps are adjacent in the two sequences but still lead to an optimal alignment. In experiments — again with 100 random sequences from GenBank — we found that the latter case was extremely rare: less than 0.02% of gapped alignments generated by BLAST contain adjacent insertions. Importantly, in 99% of these rare cases, an alignment that does not contain adjacent gaps and scores no more than 5% less than the optimal also exists. As a result, only 17 of the 1,013 million alignments scoring above an e-value cutoff of 10 are not detected when restricted insertion is imposed.

We propose taking advantage of the infrequency of adjacent gaps, and propose that these are not permitted in the alignment process. This works as follows. If the best score for the current cell $[i, j]$ is derived from $I_y(i, j-1)$, that is, from an insertion with respect to sequence x, then the value of $I_x(i, j)$ is not calculated. Similarly, the value for $I_y(i, j)$ is not determined if the best score for the cell is derived from $I_x(i-1, j)$.

With these new constraints, the recurrence relations from Section II-C are rewritten as follows:

$$\begin{aligned}
M(i,j) &= B(i-1, j-1) + s(x_i, y_j) \\
B(i,j) &= \max \begin{cases} I_x(i-1,j) \\ I_y(i,j-1) \\ M(i,j) \end{cases}
\end{aligned}$$

$$\begin{aligned}
if\ I_y(i,j-1) \Rightarrow &\quad B(i,j) \\
I_x(i,j) = &\quad -\infty \\
else & \\
I_x(i,j) = &\quad \max \begin{cases} M(i,j) - d \\ I_x(i-1,j) - e \end{cases}
\end{aligned}$$

$$\begin{aligned}
if\ I_x(i-1,j) \Rightarrow &\quad B(i,j) \\
I_y(i,j) = &\quad -\infty \\
else & \\
I_y(i,j) = &\quad \max \begin{cases} M(i,j) - d \\ I_y(i,j-1) - e \end{cases}
\end{aligned}$$

The outcome of this new restriction is a saving in computation per cell. For cells where either $M(i,j) \leq I_y(i,j-1)$ or $M(i,j) \leq I_x(i-1,j)$, there is a reduction from four arithmetic operations and three comparisons to two arithmetic operations and two comparisons.

Our restricted insertion alignment algorithm is similar to the two state variation of the Gotoh algorithm [16], which stores only the larger of I_x and I_y for each cell in the matrix. However, the aim and effect of this other approach is different: it aims to reduce main-memory requirements for alignments with traceback, and the effect is that adjacent insertions are treated as single insertions with only one open gap penalty. There are no computational savings in the approach and it does not offer any savings for score-only alignment without traceback. However, importantly, Durbin observes — in support of both our and their approach — that heuristics for adjacent gaps rarely affect the resulting alignment.

IV. Results

In this section, we present the results of experiments with our semi-gapped and restricted insertion alignment schemes.

We begin by describing the collections and measurement techniques we used to quantify performance, and then present an overall comparison of results. The result summary is followed by detailed presentations of the effect of parameter choices for our schemes.

A. *Collections, Measurements, and Environment*

The Structural Classification of Proteins or SCOP database [7], [26] has been used widely to measure the accuracy of homology search techniques [9], [14], [29]. Measures based on the SCOP database are both unbiased and highly rigorous. For our evaluation, we used version 1.65 of the ASTRAL Compendium for Sequence and Structure Analysis [10] that was released on 19 December 2003. The compendium contains sequences from the SCOP database that have been annotated with fold, superfamily, and family information. The database has been filtered so that sequences with greater than 90% identity are removed, resulting in a collection that contains 8,759 structurally classified protein sequences that have each been assigned to one of 1,293 superfamilies.

Each sequence in the ASTRAL database was used to search the entire collection. Search accuracy was measured with the commonly-used *Receiver Operating Characteristic* (ROC) [18], which provides a measure of accuracy based on the ranked list of alignments returned by each search. The ROC score provides a measure between 0 and 1, where higher values reflect better sensitivity and selectivity. A list of true positives of length L is used to determine the score. For the SCOP test, this list is comprised of SCOP sequences from the same superfamily as the query. When measuring the ROC score, the list of alignments is truncated after the first n false positives, where n is typically 50 or 100. The ROC$_n$ score is calculated as follows:

$$ROC_n = \frac{1}{nL} \sum_{1 \leq F \leq n} u^F$$

where F is the position of the F^{th} false positive in the list of reported alignments, and u^F is the number of true positives that ranked ahead of that false positive. The final ROC score is determined by finding the average score across all queries where $L \geq 1$.

Unfortunately, the SCOP database is too small to allow meaningful comparison of the speed of alignment algorithms. Therefore, we used the GenBank non-redundant protein database for timing experiments and, as described previously, randomly selected 100 sequences from the collection that were then used as queries to search that collection. We used the 30 June 2004 release of GenBank NR, which contains 1,873,745 sequences in around 622 megabytes of sequence data.

When searching the ASTRAL database, we use the size of the GenBank NR database for the calculation of E-values. We did this to ensure that the cutoff scores used when measuring the speed of searching NR and accuracy of searching ASTRAL are comparable. Without this adjustment, the values of $S2$ (and, therefore, $R \times S2$) are lower when measuring the accuracy of the semi-gapped alignment technique which, in turn, increases the number of alignments that are rescored and improves ROC scores. Using the same effective database size when measuring speed and accuracy avoids this bias.

For our timing experiments with GenBank NR, we ran each of the 100 queries and recorded all high-scoring ungapped extensions reported using the first two stages of NCBI BLAST with default parameters. These ungapped alignments were then used as input to all algorithms we tested, and timings reported are for all stages following ungapped alignment only; these timings therefore include all non-ungapped alignment stages, collecting traceback information, and preparation of alignments for display in the BLAST standard format. The best elapsed time of three runs was reported for each query.

The results presented are based on experiments carried out on an Intel Pentium 4 2.8GHz workstation with one gigabyte of main-memory while the machine was under light-load, that is, no other significant processes were running. All schemes — with the exception of NCBI BLAST — include our gapped alignment recursion optimisation proposed in Section II-C. For baseline comparisons, we used NCBI BLAST version 2.2.8. All code was compiled with the same compiler flags and optimisations as NCBI BLAST.

Our implementations of the gapped alignment stages of BLAST used the same Karlin-Altschul statistics to score the alignments as NCBI BLAST. Specifically, the pre-computed values of λ, K, α and β were taken from NCBI BLAST version 2.2.8. The composition of subject sequences was not used for scoring, that is, we did not use the composition-based statistics described by Schaffer et al. [33]. No filtering was applied to the query sequences.

B. *Overall Results*

Table V shows a comparison of our techniques to NCBI BLAST. Our results show that the combination of semi-gapped and restricted insertion alignment — labelled as *combined* — better than halves the average time taken to carry out alignments compared to NCBI BLAST. On average, over five seconds is saved per query when searching GenBank NR. Importantly, all schemes we tested have indistinguishable ROC scores, and this is supported by the total number of alignments reported from the queries on the GenBank NR collection. (We report the total number of alignments returned from the GenBank search as an indicator of overall accuracy performance, and have found that these alignments are almost identical for all schemes. However, we believe that ASTRAL ROC scores are a more definitive indicator of performance.)

We compared our schemes to our own baseline and NCBI BLAST. Our baseline — which is optimised by the recursion reorganisation described in Section II-C — is around 20% faster than NCBI BLAST. However, importantly, our heuristic approaches are still much faster: in particular, the combination of schemes is around 40% faster than our optimised baseline. We have also found that the semi-gapped alignment stage is efficient and effective: it discards an average of 87% of the high-scoring database sequences from Stage 1 and each semi-gapped alignment is performed in less than half the time of a gapped alignment.

In the experiments reported in Table V, default BLAST parameters were used. These include a gapped trigger score of

TABLE V

AVERAGE RUNTIME AND NUMBER OF HIGH-SCORING ALIGNMENTS FOR 100 QUERIES ON THE THE GENBANK NON-REDUNDANT DATABASE, AND SCOP ROC_{50} SCORES FOR THE ASTRAL COLLECTION. ALL ALIGNMENT TECHNIQUES USE DEFAULT PARAMETERS.

Scheme	GenBank NR		ASTRAL
	Time (secs)	Alignments Reported	ROC_{50}
Combined	4.90	31,160	0.339
Semi-gapped alignment only	5.02	31,160	0.339
Restricted insertion only	7.67	31,163	0.339
Baseline	8.34	31,163	0.339
NCBI BLAST	10.34	31,161	0.339

22.0 bits (which affects $S1$), scoring dropoff of $X = 15.0$ bits, an E-value cutoff of $E = 10.0$, and a maximum number of alignments to be reported of 500. For semi-gapped alignment, we use $N = 10$ and $R = 0.68$, and open and extend gap penalties of $o_s = 7$ and $e_s = 1$ respectively. We discuss parameter choices further in the next sections.

Our schemes can be alternatively parameterised to improve accuracy while having runtimes similar to NCBI BLAST. For example, by lowering the scoring required to trigger gapped alignment from the default value of 22.0 to 20.2 bits, the ROC_{50} score of the combination scheme increases from 0.338 to 0.342, and the average runtime increases to 10.24 seconds. However, since our primary aim is to reduce the computational cost of BLAST without affecting its accuracy, we do not discuss this in detail further here.

C. *Varying the E-value*

Table VI shows the effect of varying the E-value cutoff on different schemes. As described in Section II-B, an E-value represents the chance that an alignment with at least the same score would be found if a randomly constructed query with typical amino-acid composition was searched against a randomly generated database. BLAST uses a default E-value cutoff of 10, however a lower cutoff is often used in practice to reduce false positives that have chance similarities to the query. ROC scores vary depending on the E-value cutoff used, because a larger cutoff leads to an increase in the number of reported alignments which in turn improves ROC scores.

The results in Table VI show that as E decreases, the reduction in processing costs varies between different schemes; indeed, in unreported experiments with smaller values of E, this same trend continues. For semi-gapped alignment schemes, query evaluation times fall: when $E = 0.1$, the query evaluation is around 10%–25% faster than for $E = 100$, and almost two and a half times faster than NCBI BLAST when $E = 0.1$. This is because a smaller cutoff increases $S2$, resulting in rescoring of fewer semi-gapped alignments using the slow gapped scheme. For the other schemes — NCBI BLAST, our optimised baseline, and restricted insertion only — reducing E has almost no effect on speed, because none have the additional filtering step of semi-gapped alignment.

In terms of accuracy, the cutoff has the same effect on all schemes: the ROC_{50} scores achieved by all techniques are the same for all cutoffs we tested, including for cutoffs up to three magnitudes smaller than reported in Table VI. We also measured the effect of varying the maximum number of reported alignments for each scheme and found no distinguishable difference in accuracy between the schemes when a maximum of 5, 50, 500, or 5000 alignments were displayed.

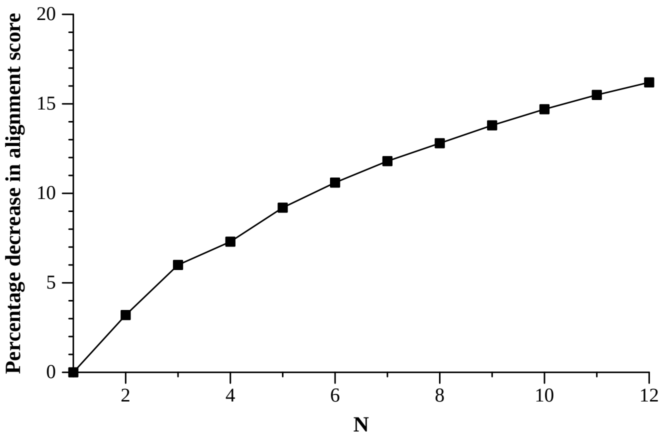

Fig. 8. Average decrease in score between semi-gapped alignments and gapped alignments for varying values of N. The same gap penalties were used for both methods. Only gapped alignments scoring above $E = 10$ were considered.

D. *Varying N and R*

Semi-gapped alignment is parameterised by two constants, N and R. The value of N controls the ratio of gapped to ungapped alignment: small values of N favour gapped alignment, and large values of N favour ungapped alignment. The value of R influences the number of semi-gapped alignments that are subsequently rescored using the gapped stage in BLAST. In our previous overall results, we report experiments with $N = 10$ and $R = 0.68$. This section shows how these values were derived experimentally, how the choices of N and R are dependent, and that they are robust when other parameters are varied.

Figure 8 shows how increasing the ratio of ungapped to gapped alignment affects alignment scores. As N increases, the score produced by semi-gapped alignment becomes increasingly lower, falling to around 15% less for values of $N \geq 11$. This is as expected: semi-gapped alignment forces insertions to occur in suboptimal locations, and the average distance between the optimal gap location and the closest

TABLE VI

AVERAGE QUERY EVALUATION TIME IN SECONDS FOR SEARCHING THE GENBANK NON-REDUNDANT DATABASE, AND RESULTS OF A SCOP ACCURACY TEST FOR ROC$_{50}$. EACH ALIGNMENT TECHNIQUE IS REPORTED FOR A RANGE OF E-VALUE CUTOFFS.

	E=100		E=10		E=0.1	
	ROC$_{50}$	Time	ROC$_{50}$	Time	ROC$_{50}$	Time
Combined	0.360	5.67	0.339	4.90	0.300	4.39
Semi-gapped alignment only	0.360	5.85	0.339	5.02	0.300	4.46
Restricted insertion only	0.360	7.78	0.339	7.67	0.300	7.53
Baseline	0.360	8.46	0.339	8.34	0.300	8.19
NCBI BLAST	0.360	10.36	0.339	10.34	0.300	10.37

permitted gap location increases with N. However, as semi-gapped alignments are only performed to identify regions that must be rescored with gapped alignment, differing scores only affect which alignments are considered in the next stage and not what is returned to users. In practice, we have found that $N = 10$ affords an excellent tradeoff between speed and accuracy.

The score reduction effect is compounded by the dropoff technique, which processes only cells that score above a dynamic threshold. Decreasing scores from semi-gapped alignment leads to a reduction in the number of matrix cells processed and, in turn, this can lead to high scoring alignments not being considered during the semi-gapped alignment stage. As we show later, lowering the open gap penalty provides an effective solution to the problem.

Figure 9 shows the effect of varying R for different values of N. Each curve in the figure shows the accuracy and speed tradeoff for a fixed value of N but with varying R. For all curves, decreasing R improves the ROC score and increases average query evaluation time. Interestingly, for values of $N \leq 10$, the curve has a characteristic shape, where the ROC score improves significantly as R is decreased from one and then reaches a near-maximum for values of approximately $R \leq 0.5$. The point on each curve shows the setting $R = 0.68$, which we advocate as the default setting and use in all experiments reported throughout our results.

E. Varying the Gap Open Penalty

As discussed in Section III-A, our semi-gapped alignment algorithm additionally penalises gaps by forcing them to occur in suboptimal locations. Specifically, on average, gaps occur $\frac{\lceil \frac{N}{2} \rceil \times \lfloor \frac{N}{2} \rfloor}{N}$ residues from their optimal open position. To compensate for this, we have explored lowering the open gap penalty for semi-gapped alignment.

Let o_g and e_g denote the open gap penalty and extend gap penalty used for gapped alignment respectively. Similarly, let o_s and e_s denote the open and extend gap penalties used for semi-gapped alignment. Because gap length has no effect on the constraints imposed by semi-gapped alignment, we let $e_s = e_g$. However, for open gap penalties, we propose $o_s = o_g - C$, where C is a constant value that reduces the open cost for semi-gapped alignment. We use a normalized value for C to ensure that this compensation is comparable across different scoring schemes.

Table VII shows the effect of varying the open gap penalty o_s and its related value of C. The combined scheme, and default values of $o_g = 11$, $N = 10$, and $R = 0.68$ are used for all results shown. In addition, because changing the open gap penalty affects the region explored by the dropoff technique, we have chosen a value of X for each value of C that provides an ROC score approximately equal to the baseline; this allows comparison of o_g values to identify the minimum query evaluation time. Our results show that $C \approx 6.0$ bits works well, that is, $o_s = 7$ is a suitable value for the BLAST default parameters.

In unreported experiments, we have also found that $C = 6.0$ bits, $N = 10$, and $R = 0.68$ provides a good compromise between accuracy and speed for most popular scoring schemes and choices of open gap penalty, o_g. We recommend and use $C = 6.0$ bits to calculate open gap penalties throughout our experiments.

F. Varying the Mutation Data Matrix

The BLOSUM62 substitution scoring or data mutation matrix is the most commonly used for BLAST searches. However, other scoring matrices are included in the default BLAST distribution:

- The BLOSUM45 matrix is constructed from alignments between distantly related proteins and is suitable for detecting distant homology
- The BLOSUM80 matrix is constructed from closely related proteins and is suitable for detecting close homology
- The older PAM30 and PAM70 matrices are still occasionally used because they can provide better sensitivity for searches conducted with short queries

We consider the effect of using these matrices in this section.

Table VIII shows a comparison of the various gapped alignment techniques for the four additional scoring matrices; results for the BLOSUM62, with $o_g = 11$, $e_g = 1$, $o_s = 7$, and $e_s = 1$ are reported in Section IV-B. For each matrix, the recommended open gap and extend gap penalties taken from the NCBI online version[2] were used and are listed in the table. As previously, we report ROC values for searching SCOP, and average query evaluation times for searching GenBank NR.

The results show that our schemes are robust for all matrices. As for BLOSUM62, the accuracy of our schemes compared to our baseline and NCBI BLAST is mostly indistinguishable. For PAM70, the combined approach has an ROC score that is 0.001 less than NCBI BLAST, but it is staggeringly more than three times as fast; simple parameter tuning can alter

[2] See `http://www.ncbi.nlm.nih.gov/BLAST/`

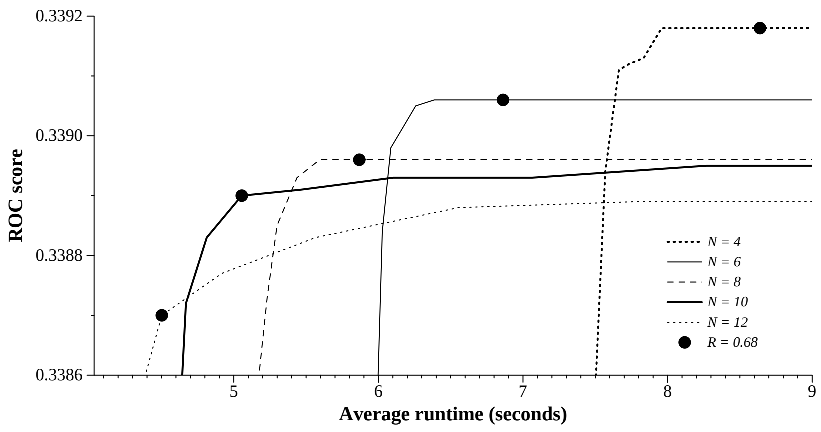

Fig. 9. Accuracy versus query evaluation times for combined semi-gapped alignment and restricted insertion alignment using different values of N and R. The default value of $R = 0.68$ is highlighted on each curve.

TABLE VII

AVERAGE QUERY EVALUATION TIME FOR VARYING SEMI-GAPPED ALIGNMENT OPEN GAP PENALTIES o_s AND THE RELATED VALUE C. FOR EACH PENALTY, A VALUE OF X THAT PRODUCES EQUIVALENT ROC$_{50}$ SCORES WAS CHOSEN.

Semi-gapped open gap penalty (o_s)	6	7	8	9
C (bits)	6.53	6.15	5.76	5.38
Dropoff X parameter (bits)	14.4	15.0	16.5	19.2
ROC$_{50}$ score	0.338	0.339	0.339	0.339
Time (secs)	5.35	4.93	5.23	6.18

TABLE VIII

COMPARISON OF THE GAPPED ALIGNMENT TECHNIQUES WHEN USED IN COMBINATION WITH THE SCORING MATRICES INCLUDED WITH BLAST. FOR EACH MATRIX, THE RECOMMENDED GAP PENALTIES WERE USED WITH $C = 6.0$.

Scoring matrix	BLOSUM45		BLOSUM80		PAM30		PAM70	
Gapped costs (o_g,e_g)	14,2		10,1		10,1		9,1	
Semi-gapped costs (o_s,e_s)	10,2		5,1		3,1		4,1	
	ROC$_{50}$	Time	ROC$_{50}$	Time	ROC$_{50}$	Time	ROC$_{50}$	Time
Combined	0.331	6.58	0.332	4.07	0.238	1.75	0.292	2.91
Semi-gapped only	0.331	6.73	0.332	4.21	0.238	1.84	0.293	3.01
Restricted insertion only	0.331	10.59	0.332	6.11	0.238	3.20	0.294	7.04
Baseline	0.331	11.23	0.332	6.72	0.238	3.59	0.294	7.91
NCBI BLAST	0.331	14.31	0.330	7.95	0.236	3.87	0.293	9.85

this tradeoff as required. For BLOSUM80, NCBI BLAST has an ROC score 0.002 less than the other approaches, which we have attributed to a minor difference in the implementation of the dropoff scheme. Overall, our semi-gapped schemes are relatively faster for detecting distant homologs.

V. CONCLUSION

The BLAST homology search algorithm has been widely-used and widely-adapted to different hardware, operating systems, and tasks. However, very little work — other than that of its original authors — has addressed the fundamental algorithmic steps it uses to accurately and efficiently compute gapped alignments. In this paper, we have proposed two improvements to BLAST and shown experimentally that — together with an optimisation of the alignment recursion — they halve the query evaluation time of the gapped alignment stages of BLAST with negligible effect on accuracy. We conclude that these steps are a valuable addition to BLAST and propose they are included in a new release of the tools.

We are currently investigating several extensions to this work. In this paper, we have confined our experimental work to protein sequences and we are currently investigating how these optimisations apply to nucleotide search; surprisingly, many of the features of the BLAST algorithm — such as neighbourhood words generated by the T parameter and the two-hit system for triggering ungapped alignments — are not implemented in NCBI BLASTN, and so this investigation is wide-ranging. We are also investigating optimisations to the first stage of BLAST and its related parameters. Last, we plan to release an entirely new implementation of BLAST that incorporates the outcomes of our work.

ACKNOWLEDGMENT

This work was supported by the Australian Research Council. We thank Lily Chen for providing the BLAST hardware and collection performance figures discussed in the introduction.

References

[1] S. Altschul, M. Boguski, W. Gish, and J. Wootton. Issues in searching molecular sequence databases. *Nature Genetics*, 6:119–129, 1994.

[2] S.F. Altschul. Generalized affine gap costs for protein sequence alignment. *PROTEINS: Structure, Function, and Genetics*, 32(1):88–96, 1998.

[3] S.F. Altschul, R. Bundschuh, R. Olsen, and T. Hwa. The estimation of statistical parameters for local alignment score distributions. *Nucleic Acids Research*, 29(2):351–361, 2001.

[4] S.F. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–480, 1996.

[5] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[6] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI–BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[7] Antonina Andreeva, Dave Howorth, Steven E. Brenner, Tim J. P. Hubbard, Cyrus Chothia, and Alexey G. Murzin. SCOP database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Research*, 32:D226–D229, 2004.

[8] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, B.A. Rapp, and D.L. Wheeler. Genbank. *Nucleic Acids Research*, 28(1):15–18, 2000.

[9] S.E. Brenner, C. Chothia, and T.J.P. Hubbard. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proceedings of the National Academy of Sciences USA*, 95(11):6073–6078, 1998.

[10] J.M. Chandonia, G. Hon, N.S. Walker, L. Lo Conte, P. Koehl, M. Levitt, and S.E. Brenner. The ASTRAL compendium in 2004. *Nucleic Acids Research*, 32:D189–D192, 2004.

[11] K.M. Chao, R.C. Hardison, and W. Miller. Recent developments in linear-space alignment methods: a survey. *Journal of Computational Biology*, 1(4):271–91, 1994.

[12] K.M. Chao, W.R. Pearson, and W. Miller. Aligning two sequences within a specified diagonal band. *Computer Applications in the Biosciences*, 8(5):481–487, 1992.

[13] X.L. Chen. personal communication, 2004.

[14] Z. Chen. Assessing sequence comparison methods with the average precision criterion. *Bioinformatics*, 19(18):2456–2460, 2003.

[15] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5:345–358, 1978.

[16] R. Durbin. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[17] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[18] M. Gribskov and N.L. Robinson. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers & Chemistry*, 20:25–33, 1996.

[19] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[20] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences USA*, 89(22):10915–10919, 1992.

[21] S. Karlin and S.F. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences USA*, 87(6):2264–2268, 1990.

[22] W.J. Kent. BLAT–the BLAST-like alignment tool. *Genome Research*, 12(4):656–664, 2002.

[23] M. Li, B. Ma, D. Kisman, and J. Tromp. Patternhunter II: Highly sensitive and fast homology search. *Journal of Bioinformatics and Computational Biology*, 2(3):417–439, 2004.

[24] B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002.

[25] S. McGinnis and T.L. Madden. BLAST: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research*, 32:W20–W25, 2004.

[26] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247(4):536–540, 1995.

[27] E.W. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4(1):11–17, 1988.

[28] G. Myers and R. Durbin. A table-driven, full-sensitivity similarity search algorithm. *Journal of Computational Biology*, 10(2):103–117, 2003.

[29] J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia. Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284(4):1201–1210, 1998.

[30] W.R. Pearson and D.J. Lipman. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[31] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences USA*, 85(8):2444–2448, 1988.

[32] W.R. Pearson and W. Miller. Dynamic programming algorithms for biological sequence comparison. *Methods in Enzymology*, 210:575–601, 1992.

[33] A.A. Schaffer, L. Aravind, T.L. Madden, S. Shavirin, J.L. Spouge, Y.I. Wolf, E.V. Koonin, and S.F. Altschul. Improving the accuracy of PSI–BLAST protein database searches with composition-based statistics and other refinements. *Nucleic Acids Research*, 29(14):2994–3005, 2001.

[34] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[35] W.J. Wilbur and D.J. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences USA*, 80(3):726–730, 1983.

[36] Z. Zhang, P. Berman, and W. Miller. Alignments without low-scoring regions. *Journal of Computational Biology*, 5(2):197–210, 1998.

[37] Z. Zhang, W. Pearson, and W. Miller. Aligning a DNA sequence with a protein sequence. *Journal of Computational Biology*, 4(3):339–349, 1997.

Michael Cameron is a PhD candidate in the School of Computer Science and IT at RMIT University. He is working on improving the efficiency and accuracy of homology search techniques, and the BLAST algorithm in particular, under the supervision of Hugh E. Williams and Adam Cannane. He is a member of the RMIT Bioinformatics Search Group and Search Engine Group. His research interests include homology search, sequence alignment, and data structures and algorithms. He received the BCompSc(Hons) degree from Monash University in 2001.

Hugh E. Williams is the Associate Professor in Information Retrieval in the School of Computer Science and IT at RMIT University, Australia. Within the School, he leads the Bioinformatics Search Group and co-leads the Search Engine Group. His research interests include homology search, genomic information retrieval, web search engines, multimedia retrieval, and data structures and algorithms. His PhD was awarded in 1998 by RMIT University. He is a member of the ACM.

Adam Cannane is a Research Fellow in the School of Computer Science and IT at RMIT University, Australia. He is a member of the RMIT Bioinformatics Search Group and Search Engine Group. His current research interests include genomic information retrieval, homology search, data compression, and web search engines. He received the PhD degree from RMIT University in 2002.