BIOINFORMATICS ORIGINAL PAPER Vol. 24 no. 16 2008, pages 1772–1778
doi:10.1093/bioinformatics/btn308

Sequence analysis

Memory-efficient dynamic programming backtrace and pairwise local sequence alignment

Lee A. Newberg[1,2]

[1]Center for Bioinformatics, Wadsworth Center, New York State Department of Health, Albany, NY 12208-3425 and [2]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA

Received on April 23, 2008; revised on June 5, 2008; accepted on June 10, 2008

Advance Access publication June 16, 2008

Associate Editor: Thomas Lengauer

ABSTRACT

Motivation: A backtrace through a dynamic programming algorithm's intermediate results in search of an optimal path, or to sample paths according to an implied probability distribution, or as the second stage of a forward–backward algorithm, is a task of fundamental importance in computational biology. When there is insufficient space to store all intermediate results in high-speed memory (e.g. cache) existing approaches store selected stages of the computation, and recompute missing values from these checkpoints on an as-needed basis.

Results: Here we present an optimal checkpointing strategy, and demonstrate its utility with pairwise local sequence alignment of sequences of length 10 000.

Availability: Sample C++-code for optimal backtrace is available in the Supplementary Materials.

Contact: leen@cs.rpi.edu

Supplementary information: Supplementary data is available at *Bioinformatics* online.

1 INTRODUCTION

Dynamic programming algorithms are often used to find an optimal solution by backtracking through intermediate values of the computation. Typical examples are that of chain matrix multiplication, string algorithms such as longest common subsequence, the Viterbi (1967) algorithm for hidden Markov models, and sequence alignment algorithms such as those of Needleman and Wunsch (1970) and Smith and Waterman (1981). Dynamic programming algorithm backtraces are also used for random sampling, where the score for each possible backtrace path is deemed to be (proportional to) the probability of the path, and it is desired to choose a path according to that probability distribution. A typical example is the algorithm of Ding and Lawrence (1999) for the sampling of RNA secondary structure. A third use for dynamic programming backtraces is as the second step of a forward–backward algorithm, such as that of Baum–Welch (Baum *et al.*, 1970) for finding the parameters of a hidden Markov model. Sometimes a trade-off with run time allows a problem to be solved without a backtrace through stored results, e.g. sequence alignment (Durbin *et al.*, 2006, Section 2.6; Myers and Miller, 1998; Waterman, 1995, page 211) and Baum–Welch (Miklós and Meyer, 2005), but this is not always the case.

When there is not enough space to store all intermediate results in high-speed memory, checkpointing strategies are employed, whereby selected stages of the computation are stored, and missing information is recomputed from these checkpoints on an as-needed basis. A stage of the computation, also known as a frontier, is a set of intermediate values that are sufficient for making subsequent computations. For instance, in a 2D dynamic programming algorithm that computes a small number of values for each (i,j) in a grid from the neighboring 'earlier' values associated with $(i-1,j)$, $(i,j-1)$ and $(i-1,j-1)$, we could define a stage as a row of the computation grid. In this case, stage k would be the values associated with the cells $\{(k,j): j=j_{\min}\cdots j_{\max}\}$, and the stage k values would be sufficient for computing values for cell (i,j) for any $i>k$. Similarly one could use columns to define stages. In many cases it makes sense to have overlapping stages; in the above example stage k might be the k-th diagonal frontier, i.e. the computation values associated with the cells $\{(i,j): i+j\in\{k-1,k\}\}$.

Herein we will describe an optimal checkpointing strategy that provably minimizes the number of stage re-computations necessary in performing a backtrace with limited high-speed memory. The algorithm is simple and efficient. Note that, because this limited-memory approach can be used to allow significant increases in locality of reference, it can provide more efficient computations even when the amount of high-speed memory might otherwise be considered sufficient.

We build upon a previous approach that is fairly memory-efficient, which is described in *Bioinformatics* (Grice *et al.*, 1997; Wheeler and Hughey, 2000). With memory enough to store M stages, their '2-level' algorithm uses the memory to compute the first M stages, but then retains only the M-th stage as a checkpoint, discarding the previous ones. Using the remaining $M-1$ memory locations, the algorithm computes stages $M+1,\ldots,2M-1$, and then uses the $(2M-1)$th stage as the second checkpoint. It continues this process, using the $(M+(M-1)+(M-2))$th stage as its third checkpoint, and so forth, up to and including $M+(M-1)+\cdots+1=M(M+1)/2$ as its M-th checkpoint. Thus, if $N=M(M+1)/2$ stages are needed in the backtrace, they can be achieved with $M=\mathcal{O}(\sqrt{N})$ memory locations; in the backtrace, each missing stage is computed using the space freed by discarding the checkpoints that are no longer needed.

Because the algorithm needs to compute each stage at most twice, once in the forward pass to create the checkpoints and once during the backtrace, the overall number of stage computations of the memory-reduced algorithm is at most double what it would have been.

Wheeler and Hughey (2000) also generalize their 2-level algorithm to an 'L-level' algorithm, where L is any positive integer. With M memory locations, the L-level algorithm can compute

$$N_{\mathrm{WH}}(M,L)=\binom{M+L-1}{L}\overset{M\to\infty}{\longrightarrow}\frac{M^L}{L!} \tag{1}$$

stages, where this formula works for any integers L and M so long as the binomial coefficient $\binom{n}{d}$ is defined to be $n!/d!(n-d)!$ when $d\geq 0$ and $n-d\geq 0$, and zero otherwise. The asymptotic limit is as $M\to\infty$ for fixed L. For the $M, L\geq 1$ algorithm, the k-th stage to be checkpointed is

$$C_k(M,L)=\sum_{j=1}^{k}N_{\mathrm{WH}}(M-(j-1),L-1) \tag{2}$$

That is, the first stage to be a checkpoint is the last stage that would be a checkpoint under the $(L-1)$-level algorithm. Generally, the k-th stage to be checkpointed is beyond the $(k-1)$th checkpoint by an amount that would be the last checkpoint for an $(L-1)$-level algorithm that uses the remaining $M-(k-1)$ memory locations. Equation (2) solves to

$$C_k(M,L)=N_{\mathrm{WH}}(M,L)-N_{\mathrm{WH}}(M-k,L)\ . \tag{3}$$

The number of stage computations for the L-level algorithm to compute $N_{\mathrm{WH}}(M,L)$ stages using M memory locations is given by the recursion

$$T_{\mathrm{WH}}(M,L) = \begin{cases} M & \text{if } L=1 \text{ and} \\ N_{\mathrm{WH}}(M,L)+ & \\ \sum_{k=1}^{M}\left(T_{\mathrm{WH}}(k,L-1)-1\right) & \text{if } L\geq 2, \end{cases} \tag{4}$$

because the L-level algorithm first computes all $N_{\mathrm{WH}}(M,L)$ stages, to get the L-level checkpoints; it then provides access to the stages in reverse order by working with the L-level checkpoints in reverse order; the L-level algorithm uses the $(L-1)$-level algorithm to generate the missing intervening stages. However, 1 is subtracted, because the last computation of each $(L-1)$-level algorithm invocation produces an L-level checkpoint that we already had available. Thus, this last computation for each $(L-1)$-level algorithm invocation is not performed. Equation (4) solves to

$$\begin{aligned} T_{\mathrm{WH}}(M,L) &= \binom{M+L-1}{1 \quad M-1 \quad L-1}-\binom{M+L-1}{L-1}+1 \\ &= N_{\mathrm{WH}}(M,L)\left(L\frac{M-1}{M}\right)+1 \\ &\overset{M\to\infty}{\longrightarrow} \frac{M^L}{(L-1)!}\ . \end{aligned} \tag{5}$$

where this formula works for any integers L and M so long as the trinomial coefficient $\binom{a+b+c}{a \quad b \quad c}$ is defined to be $(a+b+c)!/a!b!c!$ when $a,b,c\geq 0$, and zero otherwise. Thus we have a multiplier for the number of stage computations of approximately L for $M\sim\sqrt[L]{L!N}$ memory locations. Computed values of $N_{\mathrm{WH}}(M,L)$ and $T_{\mathrm{WH}}(M,L)$ for small M and L are given in Table 1 and plotted in Figure 1.

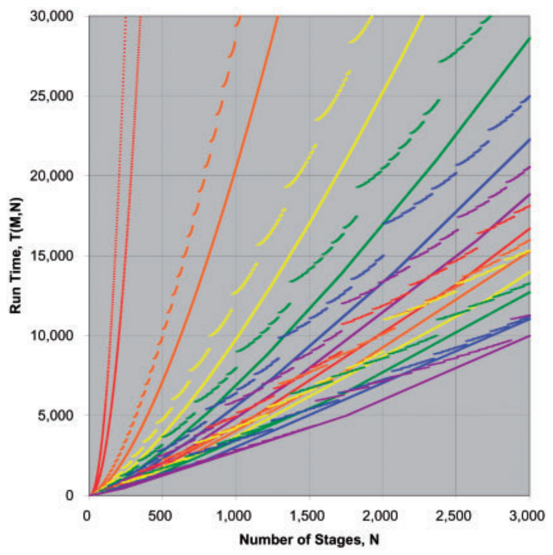

Fig. 1. Low memory comparison of algorithms. This figure exhibits the effect on the run time at low memory levels. Clockwise from the top, the curves come in 12 pairs, one each for $M=2,3,4,5,6,7,8,9,10,12,15$ and 20 memory locations. Within each pair, the curve for the L-level algorithm of Wheeler and Hughey (2000) is first; as M increases these curves become increasingly 'fractal', with jumps in the run time at several scales. The curve for the optimal checkpointing algorithm is second in each pair; these curves are piecewise linear.

The main drawback to the L-level algorithm is that it can perform badly for a value of N that falls between $N_{\mathrm{WH}}(M,L-1)$ and $N_{\mathrm{WH}}(M,L)$, for some L. Wheeler and Hughey (2000) propose an '$(L,L-1)$-level' algorithm that performs better for these intermediate values of N, but it is not optimal. Wheeler and Hughey (2000) also discuss $(L,L-1,\ldots)$-level algorithms and an optimal checkpointing algorithm, but do not provide a quick computation for choosing optimal checkpoints, nor do they give formulae to compute the number of stage computations for general values of M and N.

2 METHODS

The choice of the stages to checkpoint can be framed as an optimization problem. We write $T(M,N)$ for the number of stage computations that are needed by the optimal checkpointing algorithm for a backtrace through N stages, using room for M stages. When $N\leq M$, there is ample memory, and $T(M,N)=N$. At the other extreme, if $M=1$ there is room to compute only one stage, and $N>1$ stages cannot be computed even with an infinite amount of time available. [Following the lead of Wheeler and Hughey (2000) we bar in-place calculations.]

If $N>M>1$, and if we choose C as the first stage to checkpoint, then we begin by computing the first C stages, $1,\ldots,C$, by alternating the use of two memory locations. We store stage C, discarding the previous ones. Then, using the remaining $M-1$ memory locations, we recursively perform an optimal backtrace on the final $N-C$ stages, $C+1,\ldots,N$. Next, we present the retained stage C to the user. Finally, we discard stage C and recursively perform an optimal backtrace on the initial $C-1$ stages, using the full M

Table 1. The number of stages and run time for both the algorithm of Wheeler and Hughey (2000) and the optimal checkpointing algorithm

Alg. L	$T_{\text{WH}}(M,L)/N_{\text{WH}}(M,L)$							$T_{\text{opt}}(M,L)/N_{\text{opt}}(M,L)$						
	1	2	3	4	5	6	7	1	2	3	4	5	6	7
$M=1$	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1	1/1
2	2/2	4/3	7/4	11/5	16/6	22/7	29/8	3/2	6/4	12/6	20/8	30/10	42/12	56/14
3	3/3	9/6	21/10	41/15	71/21	113/28	169/36	3/3	13/8	34/15	70/24	125/35	203/48	308/63
4	4/4	16/10	46/20	106/35	211/56	379/84	631/120	4/4	22/13	70/29	170/54	350/90	644/139	1092/203
5	5/5	25/15	85/35	225/70	505/126	1009/210	1849/330	5/5	33/19	123/49	343/104	798/195	1638/335	3066/539
6	6/6	36/21	141/56	421/126	1051/252	2311/462	4621/792	6/6	46/26	196/76	616/181	1596/377	3612/713	7392/1253
7	7/7	49/28	217/84	721/210	1981/462	4753/924	10297/1716	7/7	61/34	292/111	1020/293	2910/671	7194/1385	15 972/2639

For the L-level backtracking algorithm of Wheeler and Hughey (2000) with memory suitable for storage of M stages, the left side of this table shows both $N_{\text{WH}}(M,L)$, the number of stages that can be produced in reverse order, and $T_{\text{WH}}(M,L)$, the number of stage computations required for that backtrace. [See Equations (1) and (5).] For instance, to perform a backtrace on $N=36$ stages with $M=3$ memory locations requires the $(L=7)$-level algorithm and requires $T=169$ stage computations. It is not straightforward to predict the number of stage computations for other values of N (Fig. 1).

For the optimal checkpointing algorithm presented here, the right side of this table shows both $N_{\text{opt}}(M,L)$, a number of stages that can be produced in reverse order, and $T(M,N_{\text{opt}}(M,L))$, the number of stage computations required for that backtrace. [See Equations (7) and (11).] When the number of stages is between $N_{\text{opt}}(M,L)$ and $N_{\text{opt}}(M,L+1)$, the optimal number of stage computations $T(M,N)$ is computed via linear interpolation. For instance, to do the backtrace on $N=36$ stages with $M=3$ memory locations, we observe that N falls between $N_{\text{opt}}(M=3,L=5)=35$ and $N_{\text{opt}}(M=3,L=6)=48$. Thus, the algorithm requires $T(M=3,N=36)=T(M=3,N=35)+6(36-35)=131$ stage computations. Thus, in this case, the number of stage computations for the L-level algorithm of Wheeler and Hughey (2000) is 29% higher.

memory locations. Thus, with an optimal choice for C, we have the recursion for $T(M,N)$:

$$T(M,N)= \left\{\begin{array}{ll} N & \text{if } N\le M, \\ +\infty & \text{if } N>M=1, \\ \min_C\left\{\begin{array}{l} C \\ +T(M-1,N-C) \\ +T(M,C-1)\end{array}\right\} & \text{if } N>M>1.\end{array}\right. \tag{6}$$

Note that Wheeler and Hughey (2000) give a different recursion for optimal checkpointing. Translated into our notation, their recursion is $T(M,N)=\min_C\{C+T(M-1,N-C)+T(M,C)-1\}$. This error may have impeded their further progress.

A straightforward computation of this recursion would require $\mathcal{O}(MN^2)$ time (Wheeler and Hughey, 2000). However, in the following we will show a mathematical solution to the recursion that permits the calculation of all the needed checkpoints in $\mathcal{O}(N)$ time.

As with the analysis of the L-level algorithm of Wheeler and Hughey (2000), we find it easier to initially restrict our attention to special values of N. The main contribution of this work is our subsequent generalization to arbitrary values of N.

2.1 Special values for the number of stages

With storage for $M\ge 1$ stages, and for any integer $L\ge 1$, we will define a special number of stages $N_{\text{opt}}(M,L)$, and we will calculate $T_{\text{opt}}(M,L)$, the number of stage computations required for a backtrace through $N_{\text{opt}}(M,L)$ stages using M memory locations. Let

$$\begin{aligned} N_{\text{opt}}(M,L) &= \binom{M+L-1}{L}+\binom{M+L-2}{L-1}-1 \\ &= \binom{M+L-1}{L}\left(1+\frac{L}{M+L-1}\right)-1 \overset{M\to\infty}{\longrightarrow} \frac{M^L}{L!}\ . \end{aligned} \tag{7}$$

For $M,L>1$, with $N=N_{\text{opt}}(M,L)$ stages, we set

$$C(M,N)=N_{\text{opt}}(M,L-1)+1=N-N_{\text{opt}}(M-1,L)\ , \tag{8}$$

the unique optimum checkpoint stage for the problem with M memory locations and $N=N_{\text{opt}}(M,L)$ stages (see Appendix for proofs). Plugging $C(M,N)=N_{\text{opt}}(M,L-1)+1$ and $N-C(M,N)=N_{\text{opt}}(M-1,L)$ into Equation (6), we obtain

$$T_{\text{opt}}(M,L) = \left\{\begin{array}{ll} N & \text{if } L=1 \text{ or } \\ & M=1, \text{ and} \\ N_{\text{opt}}(M,L-1)+1 & \\ +T_{\text{opt}}(M-1,L)+T_{\text{opt}}(M,L-1) & \text{if } L,M>1.\end{array}\right. \tag{9}$$

This solves to

$$\begin{aligned} T_{\text{opt}}(M,L) &= \left(\begin{array}{c} M+L-1 \\ 1\ \ M-1\ \ L-1\end{array}\right)+\left(\begin{array}{c} M+L-2 \\ 1\ \ M-1\ \ L-2\end{array}\right) \\ &\quad -2\binom{M+L-2}{L-2} \end{aligned} \tag{10}$$

$$\begin{aligned} &= \left(N_{\text{opt}}(M,L)+1\right)\left(L\,\frac{M-1}{M}\right) \\ &\quad \times\left(1+\frac{1}{(M-1)(M+2L-1)}\right)\overset{M\to\infty}{\longrightarrow}\frac{M^L}{(L-1)!}\ , \end{aligned} \tag{11}$$

where Equation (11) is defined for $M\ge 2$ only. Computed values of $N_{\text{opt}}(M,L)$ and $T_{\text{opt}}(M,L)$ for small M and L are given in Table 1 and plotted in Figure 1.

As with the L-level algorithm of Wheeler and Hughey (2000), with the optimal checkpointing algorithm, we have a multiplier for the number of stage computations of approximately L for $M\sim\sqrt[L]{L!N}$ memory locations. However, we shall now see that, with the optimal checkpointing algorithm, we easily achieve this multiplier for the number of stage computations even when the number of stages N is arbitrary.

2.2 General values for the number of stages

When N falls between two optimal values, $N_{\text{opt}}(M,L)$ and $N_{\text{opt}}(M,L+1)$, we can compute the number of stage computations $T(M,N)$ by linear interpolation between $N_{\text{opt}}(M,L)$ and $N_{\text{opt}}(M,L+1)$ (see Appendix for proofs). Noting that

$$\frac{T(M,N_{\text{opt}}(M,L+1))-T(M,N_{\text{opt}}(M,L))}{N_{\text{opt}}(M,L+1)-N_{\text{opt}}(M,L)}=L+1 \tag{12}$$

```
void backtrace(
    BIGINT $M_{ckpt}$,                    // previous checkpoint memory location
    BIGINT $M$,                           // $M$ unused memory locations
    BIGINT $N_{ckpt}$,                    // previous checkpoint stage
    BIGINT $N$,                           // backtrace through $N$ stages
    BIGINT $L$,                           // applicable level
    BIGINT $Nopt$,                        // $N_{\mathrm{opt}}(M, L)$
    void (*advance)(BIGINT $M_{\mathrm{from}}$, BIGINT $M_{\mathrm{to}}$, BIGINT $N_{\mathrm{to}}$, void *$p$),
    void (*available)(BIGINT $M$, BIGINT $N$, void *$p$),
    void *$p$)                            // See the figure legend for advance, available, and $p$
{
    while ($N > M$)
    {
        BIGINT $Noptm$;                   // $N_{\mathrm{opt}}(M-1, L)$
        BIGINT $Noptl$;                   // $N_{\mathrm{opt}}(M, L-1)$
        BIGINT $C$;                       // $N_{ckpt} + C$ is the next stage to checkpoint
        $Noptm = ((Nopt+1)(M-1)(M+2L-2))/((M+2L-1)(M+L-2)) - 1$;
        $Noptl = ((Nopt+1)L(M+2L-3))/((M+2L-1)(M+L-2)) - 1$;
        $C = \min(Nopt+1, N - Noptm)$;
        // Compute stage $N_{ckpt} + C$ from stage $N_{ckpt}$, by alternate use of two memory locations $M_{ckpt}+1$ and $M_{ckpt}+2$
        (*advance)($M_{ckpt}$, $M_{ckpt} + 1 + ((C-1)\%2)$, $N_{ckpt}+1$, $p$);
        for (BIGINT $i = 2$; $i \le C$; ++$i$)
            (*advance)($M_{ckpt} + 2 - ((C-i)\%2)$, $M_{ckpt} + 1 + ((C-i)\%2)$, $N_{ckpt}+i$, $p$);
        // Backtrace through stages $N_{ckpt}+C+1, \ldots, N_{ckpt}+N$
        backtrace($M_{ckpt}+1$, $M-1$, $N_{ckpt}+C$, $N-C$, $L$, $Noptm$, advance, available, $p$);
        // Present stage $N_{ckpt}+C$ to the user
        (*available)($M_{ckpt}+1$, $N_{ckpt}+C$, $p$);
        // Backtrace through stages $N_{ckpt}+1, \ldots N_{ckpt}+C-1$ via tail recursion
        $N = C - 1$;
        $L = L - 1$;
        $Nopt = Noptl$;
    }
    // Handle ample memory case, $N \le M$
    for (BIGINT $i = 1$; $i \le N$; ++$i$)
        (*advance)($M_{ckpt}+i-1$, $M_{ckpt}+i$, $N_{ckpt}+i$, $p$);
    for (BIGINT $i = N$; $i \ge 1$; $--i$)
        (*available)($M_{ckpt}+i$, $N_{ckpt}+i$, $p$);
}
```

Fig. 2. The optimal checkpointing algorithm in pseudo-C++, for a backtrace through N stages using memory sufficient for M stages. Using Equation (7), find the level $L = \max\{L : N_{\mathrm{opt}}(M,L) \le N\}$. For the convention that the memory locations are labeled $0,\ldots,M-1$ and the stages are labeled $0,\ldots,N-1$, invoke backtrace(-1, M, -1, N, L, $N_{\mathrm{opt}}(M,L)$, *advance*, *available*, p); where *advance* is a pointer to a callback function that computes stage N_{to}, to be stored in memory location M_{to}, from the immediately preceding stage, which is stored in memory location M_{from} unless N_{to} is the first stage; where *available* is a pointer to a callback function invoked during backtrace so that the user can make use of stage N, stored in memory location M; and where p is a user-supplied pointer to applicable stage-independent information. BIGINT should be an integer type able to handle integers a little larger than NM^2. Note that, although the *backtrace* routine directs the callback routines on the use of the memory locations, the actual allocation and access of the memory is not handled by the *backtrace* routine. Further, note that if the generality is not required, the pointer parameters, *advance*, *available* and p, can be eliminated, and their use in the body of the function can be replaced by 'hard-wired' calls to appropriate functions. See the Supplementary Materials for C++ source code.

we derive

$$T(M,N) = T(M, N_{\mathrm{opt}}(M,L)) + (L+1)(N - N_{\mathrm{opt}}(M,L)) \;, \qquad (13)$$

for $N \ge M > 1$.

Furthermore, for N between $N_{\mathrm{opt}}(M,L)$ and $N_{\mathrm{opt}}(M,L+1)$, it is optimal to choose the initial checkpoint $C(M,N)$ so that $C(M,N)-1$ and $N-C(M,N)$ fall between the values that they would have had to equal, if N had equaled $N_{\mathrm{opt}}(M,L)$ or $N_{\mathrm{opt}}(M,L+1)$. That is, we must choose $C(M,N)$ so as to simultaneously satisfy

$$N_{\mathrm{opt}}(M,L-1) \le C(M,N) - 1 \le N_{\mathrm{opt}}(M,L) \qquad (14)$$

and

$$N_{\mathrm{opt}}(M-1,L) \le N - C(M,N) \le N_{\mathrm{opt}}(M-1,L+1) \;. \qquad (15)$$

In practice, we choose the largest legal value,

$$C(M,N) = \min\{N_{\mathrm{opt}}(M,L)+1, N - N_{\mathrm{opt}}(M-1,L)\} \;. \qquad (16)$$

The optimal checkpointing algorithm is presented in Figure 2. Note that we include not just M, and N, but also a level L and a special stage $N_{\text{opt}}(M,L)$ in the parameter list for the recursive subroutine, because the availability of values for L and $N_{\text{opt}}(M,L)$ greatly speeds the calculations of $N_{\text{opt}}(M-1,L)$ and $N_{\text{opt}}(M,L-1)$, which are needed in the calculations of optimal checkpoints:

$$\begin{aligned} &N_{\text{opt}}(M,L-1) \\ &\quad = \left(N_{\text{opt}}(M,L)+1\right)\frac{L(M+2L-3)}{(M+2L-1)(M+L-2)}-1 \qquad (17) \\ &N_{\text{opt}}(M-1,L) \\ &\quad = \left(N_{\text{opt}}(M,L)+1\right)\frac{(M-1)(M+2L-2)}{(M+2L-1)(M+L-2)}-1\,. \qquad (18) \end{aligned}$$

It is imperative that the required calculations be impervious to integer overflows. We initially prevented overflow in integer calculations, such as abc/de for Equations (17) and (18), by canceling all common factors between each variable in the numerator and each variable in the denominator. This approach requires $3\times 2=6$ invocations of Euclid's algorithm for finding a greatest common divisor. Such a procedure leaves the value of each denominator variable at 1 and, when $N_{\text{opt}}(M,L)+1$ can be represented as an integer, the numerator variable values are sufficiently small enough to permit all the needed computations—so long as extra care is taken when verifying that the initial value of N is less than $N_{\text{opt}}(M,L+1)$.

To handle computations where the number of stages exceeds the largest unsigned integer, often $2^{32}-1\approx 4\times 10^9$, we modified our C++ software implementation to use a C++ class that manipulates integers of arbitrary size.

3 SOFTWARE

Sample C++-code for optimal backtrace is available in the Supplementary Materials.

4 RESULTS

We applied the algorithm to pairwise local alignments (Smith and Waterman, 1981) of sequences of up to 3000 nucleotides of human DNA with sequences of up to 2864 nucleotides of rodent DNA. For the largest of the alignments, to keep within a 125 MB limit for total memory use, we restricted ourselves to $M=486$ stages of storage for the $N=2864$ stages. For these choices, $L=1$, $N_{\text{opt}}(M=486,L=1)=486$, and $T(M=486,N=2864)=5242$. Thus, the multiplier for the number of stage computations is $T/N=5242/2864\approx 1.83$ for memory use $M/N=486/2864\approx 17\%$. The algorithm ran in 70 s, but would have run much more slowly if it had tried to use memory for all 2864 stages, because the resulting memory swapping would have been onerous. In contrast, the 2-level algorithm of Wheeler and Hughey (2000) computes checkpoints for stages 486, 971, 1455, 1938 and 2420, and requires two computations for all other stages with index under 2420. Thus its total number of stage computations is $2864+(2420-5)=5279$, only slightly worse than 5242.

The same calculation performed on a pair of sequences, each 10 000 nucleotides long, takes 12 min to run in 125 MB of memory, a memory size sufficient to store only 138 stages instead of the full 10 000 stages. For a problem of this size, $L=2$, $N_{\text{opt}}(\text{M}=138,L=2)=9728$, and $T(M=138,N=10000)=20134$. Thus, the multiplier for the number of stage computations is $T/N=20134/10000\approx 2.01$ for memory use $M/N=138/10000\approx 1.4\%$. In contrast, the 3-level algorithm of Wheeler and Hughey (2000) is at a particular disadvantage in that it computes its only 3-level checkpoint at stage 9591, with subsequent 2-level checkpoints at 9728, 9864, and 9999. The algorithm requires 29 448 stage computations, significantly worse than 20 134. With 1 GB of memory, sufficient for storing 1104 stages for the pairwise alignment of sequences of length 10 000 nucleotides, the optimal checkpointing algorithm requires 18 896 stage computations, whereas the 2-level algorithm of Wheeler and Hughey (2000) requires 19 891 stage computations, almost 1000 more.

On similar datasets, using a probabilistic model that defines a probability distribution on the set of possible alignments, we used the optimal backtrace algorithm to compute a centroid (Ding and Lawrence, 1999), also known as a posterior decoding (Miyazawa, 1995). This task requires a dynamic programming calculation during the backtrace that is comparable to the calculation performed during the forward pass. With sufficient memory, the total computation would require $2N$ stage computations, thus the multiplier for the number of stage computations with limited memory is

$$\frac{T_{\text{twoway}}(M,N)}{T_{\text{twoway}}(N,N)}=\frac{T(M,N)+N}{2N}\overset{M\to\infty}{\longrightarrow}\frac{L+1}{2}\;; \qquad (19)$$

this value is better than L, the multiplier of the number of stage computations for the cheaper backtrace tasks.

We also can draw independent samples from the probability distribution on the set of possible alignments. Here, the run time is as slow as the centroid calculation only when the number of samples to be drawn is of the order of the smaller of the two sequence lengths.

5 DISCUSSION

We have provided an algorithm for optimal backtrace through a dynamic programming algorithm when memory is limited. The algorithm improves upon previous work via the simplicity and speed of the calculation for the index of the optimal checkpoint, and via achievement of optimal performance for a problem of arbitrary size.

A few variations are worthy of consideration. Generally, for backtrace computations, whether or not they are achieved with the optimal checkpointing algorithm described here, the first stage is computed from initial or boundary conditions, and each subsequent stage is computed from the immediately preceding stage. Thus, at least conceptually, the first stage requires special treatment. If this distinction makes implementation of the *advance* callback routine difficult, it may be prudent to compute and permanently store the first stage in the first memory location, and to run the optimal checkpointing algorithm so as to provide an optimally computed backtrace through the remaining $N-1$ stages using $M-1$ memory locations. The number of stage computations for this approach is $1+T(M-1,N-1)$.

As already described for both optimal checkpointing and the L-level algorithm of Wheeler and Hughey (2000), in the limit as the number of memory locations M goes to infinity with a fixed multiplier L for the number of stage computations, we can backtrace through $N\sim M^L/L!$ stages with $T\sim M^L/(L-1)!$ stage computations. However, for the case when memory is severely limited, it is instructive to look at the asymptotics for a fixed value of M, with L tending to infinity. For this situation we have

$$N_{\mathrm{WH}}(M,L) \overset{L\to\infty}{\longrightarrow} \frac{L^{M-1}}{(M-1)!} \tag{20}$$

$$T_{\mathrm{WH}}(M,L) \overset{L\to\infty}{\longrightarrow} \frac{L^M(M-1)}{M!}$$
$$\sim \left(\frac{M-1}{M}\sqrt[M-1]{(M-1)!}\right)N^{\frac{M}{M-1}} \tag{21}$$

$$N_{\mathrm{opt}}(M,L) \overset{L\to\infty}{\longrightarrow} 2\frac{L^{M-1}}{(M-1)!} \tag{22}$$

$$T_{\mathrm{opt}}(M,L) \overset{L\to\infty}{\longrightarrow} 2\frac{L^M(M-1)}{M!}$$
$$\sim \left(\frac{M-1}{M}\sqrt[M-1]{\frac{(M-1)!}{2}}\right)N^{\frac{M}{M-1}}$$
$$\sim \left(\sqrt[M-1]{\frac{1}{2}}\right)T_{\mathrm{WH}}(M,L)\,. \tag{23}$$

Thus, in these low-memory situations, the optimal algorithm is asymptotically faster than the L-level algorithm of Wheeler and Hughey (2000), even when the latter is applied to its optimal problem sizes $N_{\mathrm{WH}}(M,L)$. The speed multiplier is $\sqrt[M-1]{2}$, which is approximately $1+(0.693/(M-1.347))$ for moderate values of M. See Table 1 and Figure 1.

6 CONCLUSION

When high-speed memory is limited, dynamic programming algorithm backtraces make use of checkpoints for re-computing needed intermediate values. We have provided an easy-to-use algorithm for optimally selecting the checkpoints.

ACKNOWLEDGEMENTS

We thank the Computational Molecular Biology and Statistics Core Facility at the Wadsworth Center for the computing resources for the pairwise local sequence alignment calculations.We wish to acknowledge use of the Maple software package by Waterloo Maple, Inc., without which the calculations in this article would have been much more difficult.

Funding: This research was supported by the National Institutes of Health / National Human Genome Research Institute grant K25 HG003291.

Conflict of Interest: none declared.

REFERENCES

Baum,L.E. *et al.* (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist.*, **41**, 164–171.

Ding,Y. and Lawrence,C.E. (1999) A Bayesian statistical algorithm for RNA secondary structure prediction. *Comput. Chem.*, **23**, 387–400.

Durbin,R. *et al.* (2006) *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK.

Grice,J.A. *et al.* (1997) Reduced space sequence alignment. *Comput. Appl. Biosci.*, **13**, 45–53.

Miklós,I. and Meyer,I.M. (2005) A linear memory algorithm for Baum-Welch training. *BMC Bioinformatics*, **6**, 231.

Miyazawa,S. (1995) A reliable sequence alignment method based on probabilities of residue correspondences. *Protein Eng.*, **8**, 999–1009.

Myers,E.W. and Miller,W. (1998) Optimal alignments in linear space. *Comput. Appl. Biosci.*, **4**, 11–17.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Smith,T.F. and Waterman,M.S. (1981) Comparison of biosequences. *Adv. Appl. Math.*, **2**, 482–489.

Viterbi,A.J. (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE T. Inform. Theory*, **13**, 260–269.

Waterman,M.S. (1995) *Introduction to Computational Biology. Maps, Sequences and Genomes*. Chapman & Hall / CRC, London, UK.

Wheeler,R. and Hughey,R. (2000) Optimizing reduced-space sequence analysis. *Bioinformatics*, **16**, 1082–1090.

APPENDIX: PROOF OF RUN TIME

So that this section is self-contained, we will restate the relevant assumptions and definitions.

We define the following functions and will prove their usefulness presently.

$$N_{\mathrm{opt}}(M,L)=\binom{M+L-1}{L}+\binom{M+L-2}{L-1}-1 \tag{A1}$$

$$T_{\mathrm{opt}}(M,L) \tag{A2}$$
$$=\binom{M+L-1}{1\ \ M-1\ \ L-1}+\binom{M+L-2}{1\ \ M-1\ \ L-2}$$
$$-2\binom{M+L-2}{L-2}$$

We take as given that the number of stage computations required satisfies:

$$T(M,N)= \tag{A3}$$
$$\begin{cases} N & \text{if } N\le M,\\ +\infty & \text{if } N>M=1,\\ \min_C\left\{\begin{array}{l} C\\ +T(M-1,N-C)\\ +T(M,C-1)\end{array}\right\} & \text{if } N>M>1,\end{cases}$$

where a choice of value for C that minimizes this last expression for a given N and M is called an optimal first checkpoint, $C(M,N)$.

THEOREM. *Let N be the number of stages to be made available in a backtrace using storage for M stages. For a choice of N and M, define*

$$L\ =\ \max\{L\!:\!N\ge N_{\mathrm{opt}}(M,L)\}\,. \tag{A4}$$

We will show that

$$T(M,N)=T(M,N_{\mathrm{opt}}(M,L))+(L+1)(N-N_{\mathrm{opt}}(M,L)) \tag{A5}$$

for $N\ge M\ge 1$, where the second term is deemed zero if $N=N_{\mathrm{opt}}(M,L)$, even when $L=+\infty$. We will show that, when $N>M>1$, the value of an optimal first checkpoint $C(M,N)$ satisfies

$$N_{\mathrm{opt}}(M,L-1)\le C(M,N)-1\le N_{\mathrm{opt}}(M,L) \tag{A6}$$

and

$$N_{\mathrm{opt}}(M-1,L)\le N-C(M,N)\le N_{\mathrm{opt}}(M-1,L+1)\,. \tag{A7}$$

PROOF. *The proof will be by induction on N and M. We have two base cases: first, a base case for a low value of N and, second, a base case for a low value of M.*

Base case, $N=M\ge 1$

We wish to show that Equation (A5) correctly matches Equation (A3) when $N=M\ge 1$.

We put $L=1$ into Equation (A1) for $N_{\rm opt}(M,L)$ and Equation (A2) for $T_{\rm opt}(M,L)$:

$$N_{\rm opt}(M,L)=\binom{M}{1}+\binom{M-1}{0}-1=M \tag{A8}$$

$$T_{\rm opt}(M,L)=\binom{M}{M-1}+0-0=M\ , \tag{A9}$$

where the trinomial terms with $L-2$ vanish by our convention. Thus, for this base case, Equation (A4) indicates that $L=1$. It follows that Equation (A5) gives $T(M,N)=M$, which is in agreement with Equation (A3) because $N=M$.

Base Case, $N>M=1$
We wish to show that Equation (A5) correctly matches Equation (A3) when $N>M=1$.

We put $M=1$ into Equation (A1) for $N_{\rm opt}(M,L)$ and Equation (A2) for $T_{\rm opt}(M,L)$:

$$N_{\rm opt}(M,L)=\binom{L}{L}+\binom{L-1}{L-1}-1=1 \tag{A10}$$

$$T_{\rm opt}(M,L)=\binom{L}{L-1}+\binom{L-1}{L-2}-2\binom{L-1}{L-2}=1 \tag{A11}$$

Thus, for this base case, Equation (A4) indicates that $L=+\infty$. Because N is strictly greater than $N_{\rm opt}(M,L)$ (for any L) it follows that Equation (A5) gives $T(M=1,N>1)=+\infty$, in agreement with Equation (A3), as desired.

General Case, $N>M>1$
We assume that the theorem is proved true for $N'\geq M'\geq 1$, when $N'\leq N$ and $M'\leq M$ but $(N',M')\neq(N,M)$. We will show that checkpoint choices C' and $C''=C'+1$ will give the same number of stage computations if both C' and C'' satisfy the restrictions of Equations (A6) and (A7). We will show that if C'' is too high to satisfy these restrictions then use of C' will lead to fewer stage computations; we will show that if C' is too low to satisfy these restrictions then use of C'' will lead to fewer stage computations. Together these will demonstrate that the restrictions are optimal and that they can be satisfied simultaneously. We will then show that satisfaction of the restrictions implies Equation (A5).

Let $T'(M,N)$ and $T''(M,N)$ be the number of stage computations required given that C' or C'', respectively, is chosen as the first checkpoint, and optimal checkpoints are used in all remaining subproblems:

$$T'(M,N)=C'+T(M-1,N-C')+T(M,C'-1)\ , \tag{A12}$$

and

$$T''(M,N)=C''+T(M-1,N-C'')+T(M,C''-1)\ . \tag{A13}$$

Set L_1 to be the unique integer such that

$$N_{\rm opt}(M,L_1-1)\leq C'-1<N_{\rm opt}(M,L_1)\ , \tag{A14}$$

and observe that

$$N_{\rm opt}(M,L_1-1)<C''-1\leq N_{\rm opt}(M,L_1)\ . \tag{A15}$$

Set L_2 to be the unique integer such that

$$N_{\rm opt}(M-1,L_2)<N-C'\leq N_{\rm opt}(M-1,L_2+1)\ , \tag{A16}$$

and observe that

$$N_{\rm opt}(M-1,L_2)\leq N-C''<N_{\rm opt}(M-1,L_2+1)\ . \tag{A17}$$

Using our induction hypothesis [Equation (A5)], we thus compute that

$$\begin{aligned}
&T''(M,N)-T'(M,N) && \text{(A18)}\\
&\quad=(C''-C')\\
&\qquad+(T(M-1,N-C'')-T(M-1,N-C'))\\
&\qquad+(T(M,C''-1)-T(M,C'-1)) && \text{(A19)}\\
&\quad=1-(L_2+1)+L_1=L_1-L_2\ . && \text{(A20)}
\end{aligned}$$

We observe that when both C' and C'' satisfy the restrictions given as Equations (A6) and (A7) then $L_1=L_2$ and the stages C' and C'' make equally good choices as a checkpoint. When C'' is too large for the restrictions then $L_1>L_2$, and when C' is too small for the restrictions then $L_1<L_2$. Thus, we have verified that the restrictions define an optimal first checkpoint.

Finally, using the induction hypothesis, we compute $T(M,N)$ to verify that it yields Equation (A5), using a value of C satisfying the restrictions of Equations (A6) and (A7).

$$\begin{aligned}
&T(M,N)\\
&\quad=C+T(M-1,N-C)+T(M,C-1) && \text{(A21)}\\
&\quad=C\\
&\qquad+T_{\rm opt}(M-1,L)+(L+1)(N-C-N_{\rm opt}(M-1,L))\\
&\qquad+T_{\rm opt}(M,L-1)+L(C-1-N_{\rm opt}(M,L-1)) && \text{(A22)}\\
&\quad=T_{\rm opt}(M-1,L)+T_{\rm opt}(M,L-1)+N_{\rm opt}(M,L-1)+1\\
&\qquad+(L+1)(N-N_{\rm opt}(M-1,L)-N_{\rm opt}(M,L-1)-1) && \text{(A23)}\\
&\quad=T_{\rm opt}(M,L)+(L+1)(N-N_{\rm opt}(M,L))\ , && \text{(A24)}
\end{aligned}$$

as desired. For the last equality, we have used $N_{\rm opt}(M,L)=N_{\rm opt}(M-1,L)+N_{\rm opt}(M,L-1)+1$ and $T_{\rm opt}(M,L)=T_{\rm opt}(M-1,L)+T_{\rm opt}(M,L-1)+N_{\rm opt}(M,L-1)+1$, which are easily proved from Equations (A1) and (A2).