Multiple Sequence Alignment using Anytime A*

Rong Zhou and Eric A. Hansen
Computer Science Department
Mississippi State University
Mississippi State, MS 39762
{rzhou,hansen}@cs.msstate.edu

Alignment of multiple DNA or protein sequences is a central problem in computational biology. To create an alignment, gaps are inserted into sequences to shift characters to matching positions and a scoring function is used to rank the biological plausibility of alignments. Multiple sequence alignments are used to identify homologies among different species that reveal evolutionary history from a common ancestor. They are also used to discover genetic causes of certain diseases and to predict protein structure, which has significant importance in the design of drugs.

The multiple sequence alignment problem can be formalized as a shortest-path problem through a d-dimensional lattice, where d is the number of sequences to be aligned (Gusfield 1997). Dynamic programming is the traditional approach to constructing optimal alignments. Improved performance has recently been achieved using A*. However, the multiple alignment problem presents a difficulty for the classic A* algorithm. Its branching factor of $2^d - 1$ is so large that the size of the open list dramatically exceeds the number of nodes A* must expand to find an optimal solution.

Two solutions to this problem have been proposed in the literature. Yoshizumi et al. (2000) describe an extension of A*, called *A* with Partial Expansion* (PEA*). Instead of generating all successors of a node when it is expanded, PEA* inserts only the most promising successors into the open list. The "partially expanded" node is re-inserted into the open list with a revised f-cost equal to the least f-cost of its unexpanded successors, so that it can be re-expanded later. Use of this technique dramatically reduces the size of the open list, and PEA* can solve larger multiple sequence alignment problems than A*. Unfortunately, the reduced space complexity of PEA* is achieved at the cost of node re-expansion overhead. The tradeoff between space and time complexity is adjusted by setting a "cutoff value" C, which determines which successor nodes to add to the open list.

Another way to reduce the size of the open list is to prune nodes from the open list if their f-cost is equal to or greater than a previously established upper bound, since such nodes will never be expanded by A*. This approach was first proposed by Ikeda and Imai (1999), who called it *enhanced A** (EA*). One way to obtain an upper bound is to use the solution found by weighted A* search using a weight $w > 1$ in the node evaluation function $f(n) = g(n) + wh(n)$. Ikeda and Imai suggested this method of obtaining an upper bound, but did not report experimental results for it.

In this abstract, we describe a third approach to reducing the size of the open list. In this approach, we also use weighted A* search to quickly find a solution that provides an upper bound that can be used to prune the open list. But because the first solution found may not be optimal, we continue the weighted search in order to find a sequence of improved solutions that eventually converges to an optimal solution. This also provides a sequence of improved upper bounds that can further prune the open list. We call this strategy *Anytime A** (Hansen & Zilberstein 1996). Anytime A* refines both an upper bound, corresponding to the cost of the best solution found so far, and a lower bound, given by the unexpanded node with the least unweighted f-cost. Both bounds approach each other until convergence to a provably optimal solution. Before convergence, the difference between the two bounds gives an error bound on the quality of the currently available solution. Pseudocode for the algorithm is given at the end of the paper. The open list is pruned in lines 10 through 12.

Figures 1 and 2 compare the performance of Anytime A* (ATA*) to A* with Partial Expansion and Enhanced A* (where Enhanced A* uses the first solution found by weighted A* as an upper bound to prune the open list). The PAM250 cost matrix is used with a gap cost of 8. All three algorithms require dramatically less memory than conventional A* in solving the multiple sequence alignment problem, allowing a larger number of sequences to be aligned.

Figure 1 compares their performance in aligning eight sequences from a highly similar set of sequences used in earlier experiments (Ikeda & Imai 1999; Yoshizumi, Miura, & Ishida 2000). On average, Anytime A* runs more than 7 times faster and stores only 26% more nodes than PEA* using a cutoff of $C = 0$. When PEA* uses a cutoff of $C = 50$, it stores 40% more nodes than Anytime A* and still runs 18% slower on average. Enhanced A* performs best on this test set. It runs 20% faster than Anytime A* and stores 4% fewer nodes.

Figure 3 compares the performance of the algorithms in aligning five sequences from a set of dissimilar sequences used in earlier experiments (Kobayashi & Imai 1998). For

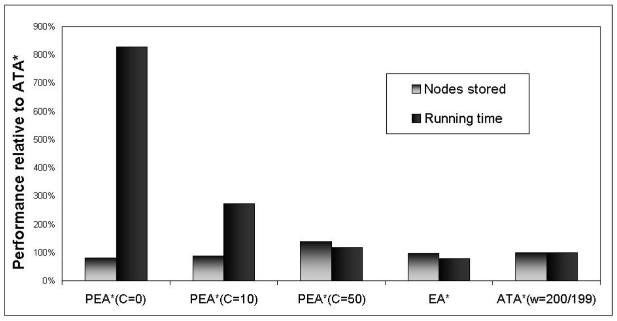

Figure 1: Average performance in aligning 8 similar sequences from (Yoshizumi, Miura, & Ishida 2000).

this test set, Anytime A* is only 2.4% slower than enhanced A* and stores 40% fewer nodes. Its better performance on the second test set is explained as follows. Because the sequences in the first test set are very similar and the heuristic is extremely accurate, the first solution found by weighted A* is optimal or close to it – a best-case scenario for Enhanced A*. Because the sequences in the second set are dissimilar and the heuristic is less accurate, the first solution found by weighted A* is usually not optimal. Anytime A* continues to find better solutions that improve the upper bound, which in turn improves memory-efficiency. Figure 2 illustrates its anytime behavior by showing how the upper and lower bounds gradually converge.

Our experimental results show that the sequence of improved solutions found by Anytime A* provide a dynamic upper bound that keeps its memory requirements close to the minimum number of nodes that must be expanded to find an optimal solution. Anytime A* is more memory-efficient than PEA* unless the latter uses the most aggressive cutoff of $C = 0$, in which case the node re-expansion overhead of PEA* slows it considerably. Anytime A* is also more memory-efficient than Enhanced A* when aligning dissimilar sequences. Anytime A* has an additional advantage over both algorithms. Because it finds a sub-optimal alignment quickly and continues to improve the alignment with additional computation time, it offers a tradeoff between solution quality and computation time that can prove useful when finding an optimal alignment is infeasible.

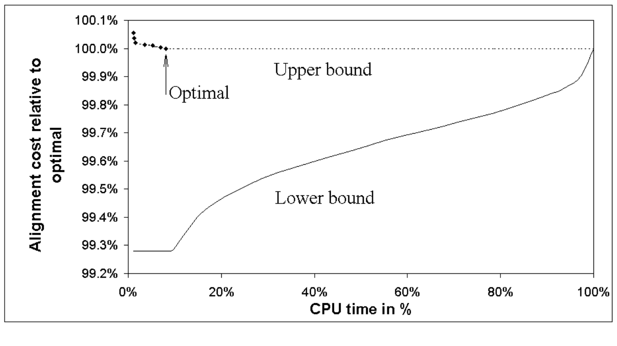

Figure 2: Convergence of bounds for Anytime A*.

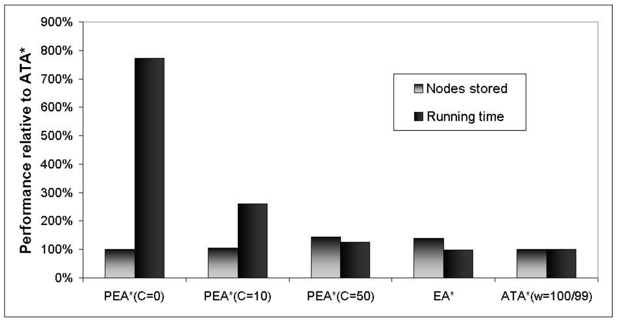

Figure 3: Average performance in aligning 5 dissimilar sequences from (Kobayashi & Imai 1998).

Pseudocode of Anytime A*

```
1  $g(s) \leftarrow 0,\ f(s) \leftarrow g(s) + w \times h(s)$
2  $OPEN \leftarrow \{s\},\ CLOSED \leftarrow \emptyset,\ bound \leftarrow \infty$
3  **while** $OPEN \neq \emptyset$ **do**
4      $n \leftarrow \arg\min_x \{f(x) \mid x \in OPEN\}$
5      $OPEN \leftarrow OPEN \setminus \{n\}$
6      $CLOSED \leftarrow CLOSED \cup \{n\}$
7      **if** $n$ is a goal node **then**
8          $bound \leftarrow g(n) + h(n)$
9          Output solution and $bound$
10         **for each** $x \in OPEN$ **and**
                   $g(x) + h(x) \geq bound$ **do**
11             $OPEN \leftarrow OPEN \setminus \{x\}$
12     **for each** $n_i \in \{x \mid x \in Successors(n),$
                   $g(n) + c(n,x) + h(x) < bound\}$ **do**
13         **if** $n_i \notin OPEN \cup CLOSED$ **or**
               $g(n_i) > g(n) + c(n, n_i)$ **then**
14             $g(n_i) \leftarrow g(n) + c(n, n_i)$
15             $f(n_i) \leftarrow g(n_i) + w \times h(n_i)$
16             $OPEN \leftarrow OPEN \cup \{n_i\}$
17             **if** $n_i \in CLOSED$ **then**
18                 $CLOSED \leftarrow CLOSED \setminus \{n_i\}$
```

References

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Hansen, E., and Zilberstein, S. 1996. Anytime heuristic search: Preliminary report. In *AAAI-96 Fall Symposium on Flexible Computation in Intelligent Systems: Results, Issues, and Opportunities*, 55–59.

Ikeda, T., and Imai, H. 1999. Enhanced A* algorithms for multiple alignments: optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theoretical Computer Science* (210):341–374.

Kobayashi, H., and Imai, H. 1998. Improvement of the A* algorithm for multiple sequence alignment. *Genome Informatics* 9:120–130.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, 923–929.