Memory-Bounded A* Graph Search

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

Abstract

The strategy for memory-bound A* search adopted by MA* (Chakrabarti et al. 1989), SMA* (Russell 1992), and SMAG* (Kaindl and Khorsand 1994) is to prune the least-promising nodes from the open list when memory is full, in order to make room for insertion of new nodes. To preserve search information from pruned nodes, heuristic estimates are backed-up through the search graph. We show that even when the heuristic function is consistent, backed-up heuristic estimates become inconsistent. Thus, it is always possible to find a better path to a node that has been previously expanded. We describe how to modify a memory-bounded A* graph-search algorithm so that it handles the discovery of a better path to a previously expanded node in a more efficient way. We demonstrate its improved performance on a challenging graph-search problem in computational biology.

Introduction

The well-known A* algorithm finds a least-cost path from a start node to a goal node in a graph, guided by a heuristic evaluation function, $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the best path found from the start node to node n and $h(n)$ estimates the cost of the best path from node n to the goal node. A significant drawback of A* is that the amount of memory required to store the open and closed lists used to organize the search is exponential in the depth of the search. This has led to development of several memory-bounded extensions of A*. One approach abandons open and closed lists, and integrates the heuristic estimates into a depth-first search strategy that requires only linear space in the depth of the search. Examples include IDA* (Korf 1985), RBFS (Korf 1993), and variants that exploit additional memory to improve performance, either by caching part of the search tree to avoid node regenerations (Miura and Ishida 1998), or, in graph search, by using a transposition table to detect and avoid duplicate paths (Reinefeld and Marsland 1994). A second approach to memory-bounded search retains the open and closed lists of A*, but does not let them grow beyond a bound on memory. When the bound is reached, the least promising nodes in the open list are pruned to make room for more promising nodes to be inserted. This approach was first adopted by the MA* algorithm of Chakrabarti et al. (1989). Russell (1992) described a simpler and more efficient version of MA*, called SMA*. Kaindl and Khorsand (1994) described a graph-search extension of SMA*, called SMAG*. Because it uses open and closed lists, we refer to this second approach as memory-bounded A*, and we focus on it in the rest of this paper.

Heuristic search in graphs is more complex than in trees because there can be more than one path to a node. To avoid duplicate search effort, A* must recognize when it re-encounters the same node along different paths. This is accomplished by comparing each newly generated node to nodes already in the open and closed lists. The value of g can change when a shorter path to a node is found, and, if the node has already been expanded, the g-values of all of its descendents may need to be revised. Propagating these changes complicates the implemenation of A*. However, Hart, Nilsson and Raphael (1968) showed that this complication can be avoided if A* uses a consistent (or monotone) heuristic. In a well-known result, they proved that it is impossible to find a better path to a node that has previously been expanded, if the heuristic is consistent.

We begin this paper with the observation that the result of Hart, Nilsson and Raphael does not extend to memory-bounded A* search. Because MA*, SMA*, and SMAG* modify heuristic estimates during search by performing backups, heuristic estimates become inconsistent, even if the original heuristic function is consistent. Consequently, memory-bounded A* graph search cannot avoid the complication of finding better paths to already expanded nodes. We carefully analyze this problem, and present an improved version of SMAG* that handles it more efficiently than the original SMAG* algorithm.

Background

In graph search, whenever A* expands a node, it checks each child to determine if it is a duplicate of a node already in the open or closed list. If a child n' of n is already present in the open list and $g(n') > g(n) + c(n,n')$, then $g(n')$ is reset to $g(n) + c(n,n')$ and the pointer to its best parent is redirected. (A* maintains a pointer to a node's parent along the best path from the start node, to allow the best path from the start to the goal to be extracted at the end of the search.) If a child n' of n is already present in the closed list and a better path to it is found, then, similarly, $g(n')$ is reset to the smaller path cost and the pointer to its best parent is redirected. In addition, A* must update the g-values and backward pointers of all the node's descendents in the search graph. We focus on this latter complication of graph search in the rest of this paper.

There are two ways in which it can be handled by A*. In the original version of A* (Hart *et al.* 1968), node n' is moved from the closed list back to the open list. This sets in motion a chain of node selections and (re-)expansions that eventually propagates the updated information through the search graph. Nilsson (1980) later described a version of A* that handles this differently. Instead of moving n' from the closed list back to the open list, it leaves n' on the closed list and immediately updates the g-values and backward pointers of all of its descendents in the search graph. This allows A* to use the revised g-values as soon as possible to improve selection of nodes from the open list. But propagation incurs a potentially exponential overhead. As a result, the first version of A* is usually more efficient, and it is more widely used in practice.

To prevent the combined size of the open and closed lists from exceeding memory, MA*, SMA*, and SMAG* prune the least-promising nodes from the open list when memory is full. They also allow partial expansion of nodes. Nodes are generated (and pruned) one at a time and the open list contains both unexpanded nodes and partially expanded nodes. To preserve search information that would be lost by pruning, heuristic estimates are "backed-up" through the search tree, where a backup takes the form,

$$h(n) := \min_{n' \in Children(n)} \{c(n,n') + h(n')\}.$$

These backed-up heuristic estimates improve the direction of the search after pruning, when the parents of pruned nodes are moved from the closed to open list.

The combined effect of pruning, partial expansion, and backups makes the extension of memory-bounded search to graphs non-trivial. A significant complication is the need to prune a node from the closed list if it does not occur on the best path to any node on the fringe. If this is not done, nodes can accumulate on the closed list that perform no function and cannot be pruned – in effect, a "memory leak" is created. To detect such nodes, Kaindl and Khorsand (1994) introduced a sophisticated book-keeping technique called "blocking." This is the principal innovation of their graph-search extension of memory-bounded A*.

In the rest of this paper, we focus on how to handle the discovery of a better path to a previously-expanded node. If a better path to a node is found, its g-value and backward pointer are revised and this change is propagated to any descendents. We have seen that A* does this in one of two ways – by propagating the changes immediately, or by ensuring that this is done eventually by moving the closed node back to the open list. Instead of adopting either strategy, Kaindl and Khorsand's SMAG* algorithm performs no propagation at all. It simply prunes all the descendents of a node once a better path to it is found. These nodes can be re-generated, and SMAG* still finds an optimal solution. But throwing away the search information they contain can decrease the efficiency of the search.

The contribution of this paper is to show how to preserve this information by propagating revised g-values to descendents, instead of pruning the desdendents. But first, we underscore the importance of this issue by showing that in memory-bounded A*, use of a consistent heuristic cannot prevent discovery of a better path to an already-expanded node.

Heuristic inconsistency

A heuristic evaluation function h is said to be *consistent* (or equivalently, *monotone*) if, for every node n and successor node n',

$$h(n) \leq h(n') + c(n,n'),$$

where $c(n,n')$ is the cost of the arc from n to n'. For graph-search problems, Hart, Nilsson and Raphael (1968) proved that if a heuristic is consistent, it is never possible for A* to find a better path to an already-expanded node. If this results extends to memory-bounded search, it limits the significance of any technique for improving the efficiency with which SMAG* handles the case of finding a shorter path to an already-expanded node, since most heuristics are consistent. But we show this result does not extend to SMAG*.

In SMAG*, heuristic estimates are backed-up through the search graph. This increases the value of h (and thus f) for nodes on the closed list. When nodes are pruned from the open list to free memory, their parent nodes are restored from the closed list to the open list. Although the improved heuristic estimates of the parent nodes are admissible, they may no longer be consistent.

Figure **??** shows part of a search graph that illustrates

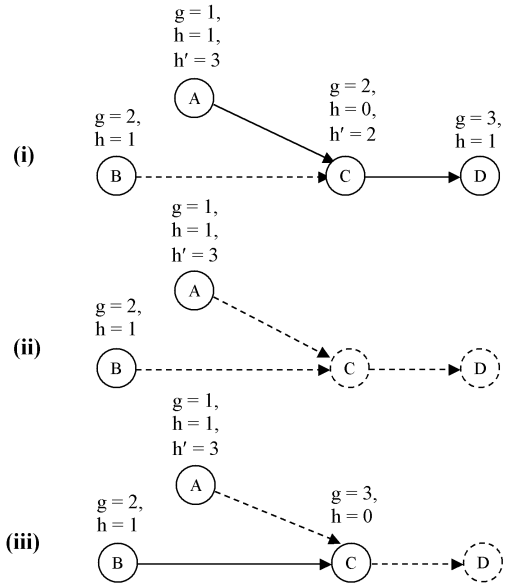

Figure 1: Backing up a consistent heuristic, followed by pruning, can create an inconsistent heuristic.

this. Dark lines indicate nodes and arcs in the explicit graph. Dashed lines indicate nodes and arcs in the implicit graph only (that is, they have not been generated). All arcs are assumed to have a cost of 1. The static heuristic evaluation function h is consistent. Because the g-value of node A is 1 and the g-value of node B is 2, the best path to node C is through node A. The first time node C is expanded, A* has found the best path to it, through node A. Figure **??**(i) shows the situation after node C is expanded for the first time. The heuristic estimates of nodes C and D are backed-up to node A. Backed-up heuristics are represented by h'. Panel (ii) shows the same portion of the search graph after nodes C and D are pruned. Now, node B will be expanded before node A, and, after node B is expanded, panel (iii) shows that node C will be expanded, although the best path to node C through node A has not yet been found. This happens because the backed-up heuristic for node A is not consistent with the heuristic for node C. When node A is eventually re-expanded, a better path to node C is found.

When a heuristic is admissible but not consistent, Mero (1984) introduced a technique called *pathmax* that restores consistency. When a node n is expanded, the h-value of each child n' is checked. If $h(n') < h(n) - c(n, n')$, then the value $h(n')$ is increased to $h(n) - c(n, n')$. By improving the heuristic value of fringe nodes, pathmax can reduce the number of node expansions needed to find an optimal solution. Because pathmax restores the consistency of the heuristic *within the explicit graph*, it is tempting to conclude that its use means it is impossible to find a better path to an already-expanded node. In fact, Chakrabarti et al. (1989) mistakenly assumed this when they discussed the possibility of an extension of MA* to graph-search problems. In their words: "Once pathmax is properly maintained, the heuristic estimate effectively becomes monotone and there is no need to bring a node from CLOSED to OPEN." (p. 201) But this is not true. As Nilsson (1998, p. 153) explains: "When a heuristic function does not satisfy the consistency condition, but is otherwise admissible, then (using an idea proposed by Mero (1998)) we can adjust it (during search) to one that does satisfy the consistency condition... But it is possible that a node on CLOSED that has its h-value adjusted in this way may have to be moved back to OPEN." Figure **??**(iii) illustrates that pathmax does not prevent an inconsistent heuristic from causing a node to be expanded before the best path to it has been found. The reason for this is that *pathmax* only restores the consistency of the heuristic along arcs that have been explicitly generated. But in graph search, two nodes may occur in the open list before all arcs between them have been generated, as Figure **??**(iii) shows. As a result, a node may be expanded before the best path to it is found.

Propagating g-values

When A* finds a better path to an already expanded node, it propagates the node's revised g-value to its descendents. By contrast, SMAG* simply prunes all of the node's descendents. The descendents can be regenerated, and SMAG* still finds an optimal solution. But the decision to prune seems unnecessary and crude. Propagating revised g-values to a node's descendents is more in keeping with the original A* algorithm, and also, more in keeping with the strategy of memory-bounded A*. This strategy is to retain as many nodes as possible in memory (to avoid re-expansions), and when memory is full, to prune the least promising node(s). But when SMAG* finds a better path to a node, its behavior contradicts this strategy. Pruning all of a node's descendents contradicts the strategy of retaining as many nodes as possible in memory, and also, it contradicts the strategy of pruning the worst nodes. If the node's descendents have survived previous pruning *and* their f-values can be immediately improved, why prune them? Instead, we consider how to adapt the propagation strategies used by A* to memory-bounded A*.

Two propagation strategies are used with A* graph search and both can be extended to memory-bounded A*, with appropriate modification. The first strategy is to move a node from the closed list back to the open list. In memory-bounded A*, we must consider how this strategy generalizes to partially-expanded nodes. If a partially-expanded node is on the open list, it has descendents that have already

been generated and need to have their g-values updated. To allow partial node expansion, each node maintains a pointer to the next successor node it will generate. (We call this pointer the *successor index*.) When a node is first opened, its successor index is initialized. If a shorter path is found to a closed node, it is moved from the closed list back to the open list and its successor index is re-initialized. If a shorter path is found to a partially-expanded node, it remains on the open list but its successor index is also re-initialized. Initializing the successor index means the next time the node is expanded, it acts as a newly-opened node and attempts to generate its first successor. This ensures that revised g-values are propagated to all descendents of the node.

The second strategy for propagating g-values used by A* is to invoke a recursive procedure that immediately propagates the revised g-value to all of the node's descendents. This strategy can also be adapted for memory-bounded A* graph search, but it is more complex to implement. In the original A* algorithm, this strategy requires each node to store pointers to all of its successor nodes. (Otherwise, such pointers are not necessary in the original A* algorithm.) In memory-bounded search, it is not reasonable to store pointers to all successor nodes in each node. Besides requiring extra memory, forward pointers can become corrupted when successor nodes are pruned from memory. (The book-keeping that would be necessary to properly maintain all pointers in the presence of pruning is prohibitive.) However, it is still possible to extend this second strategy for propagating g-values to memory-bounded A*. We do so by modifying the strategy so that it only propagates revised g-values to descendent nodes that occur along a best path, since SMAG* must maintain forward pointers to nodes that occur along a best path to allow pruning. Because descendent nodes that are not found on current best paths are not considered, there is a danger that the best path to some other node may change in a way that is not detected by this method. To ensure that this does not cause a problem, we add an extra step. Whenever the g-value of a node is changed, we re-initialize its successor index, and, if it is a closed node, we re-open it. This solution is admittedly inelegant. Although revised g-values are propagated immediately, the strategy of re-opening nodes is still resorted to. Nevertheless, there are some problems for which this second strategy is useful, even in this modified form. We describe one example in the experimental results section.

Because the complexity of propagating the improved g-value of a node to its descendents is not more than the complexity of pruning the descendents, it cannot increase the complexity of the algorithm. It can improve its performance, however. Using either strategy for propagating revised g-values, it is possible to preserve search information – especially backed-up heuristic estimates – that can improve the direction of the search and reduce the time it takes to find a solution.

Pseudocode

Pseudocode for our revised SMAG* algorithm is given in an appendix. It can be compared to the pseudocode for SMAG* given by Kaindl and Khorsand (1994). The principal difference is in the procedure GRAPH-CONTROL. Instead of pruning the descendents of a node, revised g-values are propagated by either the first strategy of re-opening the node or by the second strategy of invoking a recursive procedure called PROPAGATE that updates the g-values of descendents immediately.

We note two other differences between our SMAG* pseudocode and that of Kaindl and Khorsand. First, we keep track of the depth, $d(n)$, of each node, as do Chakrabarti et al. (1989) and Russell (1992). This is necessary for tie-breaking when $d(n) \neq g(n)$, since the deepest among tied nodes must be expanded first. Although Kaindl and Khorsand do not keep track of node depth in their pseudocode, it is harmless on their test problem, the Fifteen Puzzle, because $d(n) = g(n)$ for it and $g(n)$ can be used as a tie-breaker. In general, however, the depth of a node must be maintained in addition to the g-value. Moreover, when revised g-values are propagated after a better path to a node is found, revised depth values should also be propagated.

A second difference is that we replace the blocking procedure of their algorithm with a simpler method for deleting nodes from the closed list that cannot be part of an optimal solution. This alternative, called *cutting*, is performed by the procedure CUT. Both blocking and cutting detect closed nodes that are not on an optimal path to any node on the search fringe, so that they can be deleted from the closed list. This is necessary because such nodes cannot be pruned from memory in any other way. Blocking does so by using an extra data structure that is unnecessary using our method. We believe that cutting is more intuitive than blocking and simpler to implement. The behavior of SMAG* is the same using either method.

Experimental results

To distinguish between the original version of SMAG* and the two versions that correspond to the two strategies for propagating g-values, we refer to SMAG*-prune (the original), SMAG*-reopen (the first strategy), and SMAG*-propagate (the second strategy). We compare their performance on a challenging graph-search problem: the multiple sequence alignment problem. In computational biology, this problem involves comparing several protein or DNA

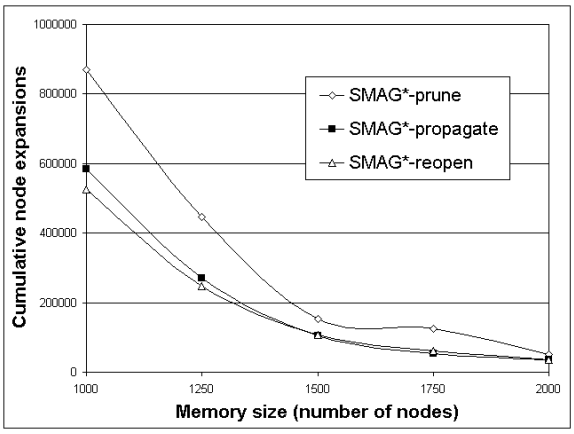

Figure 2: Both SMAG*-reopen and SMAG*-propagate outperform SMAG*-prune when backups and pruning create an inconsistent heuristic.

sequences to determine their similarity – a comparison that helps in predicting the function of the sequences or their evolutionary relationships (Carrillo and Lipman 1988). We note that this problem can be formulated as the problem of finding the least-cost path from a start node to a goal node in a k-dimensional lattice, where k is the number of sequences compared. This formulation makes it possible to use A* to solve the multiple sequence alignment problem, and the development of improved versions of A* that can solve large instances of this problem is an active area of research (Ikeda and Imai 1999; Miura and Ishida 1998; Yoshizumi et al. 2000; Korf and Zhang 2000). Because the search space is a lattice, there are combinatorially many different paths from the start node to any other node, creating a challenging graph-search problem.

We tested SMAG*-prune, SMAG*-reopen and SMAG*-propagate on a three-sequence alignment problem, with sequences of length 100. We used the same heuristic as Yoshizumi et al. (2000), which is a consistent heuristic. The results of our comparison (which are averaged over one hundred random instances of the problem) are presented in Figure **??**. When memory is large enough, there is no difference in the behavior of the algorithms. The reason is that if the heuristic is consistent, and if memory is large enough to make pruning unnecessary (or infrequent), the heuristic remains consistent and a better path to an already expanded node is never (or rarely) found. Thus, there is no reason to prune or propagate.

When the size of memory falls below some threshold (2000 nodes for this problem), SMAG*-reopen and SMAG*-propagate begin to have an advantage over SMAG*-prune. Our results show that they run up to 66% faster. The explanation is that when pruning is more frequent, nodes with backed-up heuristic estimates are moved from the closed list back to the open list, and the resulting heuristic inconsistency causes SMAG* to begin to find better paths to previously expanded nodes. When this happens, propagating revised g-values (either immediately or by reopening nodes) results in better performance than pruning because it preserves backed-up heuristic estimates that can improve the direction of search.

The first strategy of re-opening nodes is much simpler to implement, and, as Figure **??** shows, somewhat faster. Thus, it is preferred. However, there is one case in which the second strategy of propagating g-values immediately is unavoidable. If a graph-search problem contains zero-cost edges, it is possible for a node and its child to have the same f-value and the same g-value. The SMAG* algorithm will not perform correctly unless the child is expanded before its parent in a tie-breaking situation, and the g-value cannot serve as a tie-breaker in this case. To ensure correct tie-breaking, the depth information for each node must be kept up to date, and therefore, revised depth information must be propagated immediately when a better path to a node is found. Because revised g-values can be propagated at the same time as revised depth information, the second strategy is adopted in this situation. In multiple sequence alignment, the cost function sometimes allows zero-cost edges, and that is why we mention this complication. For most other search problems, zero-cost edges do not occur and the simpler strategy of re-opening nodes can be adopted.

We qualify our experimental results by making three additional observations. First, the improvement of SMAG*-reopen and SMAG*-propagate over SMAG*-prune is proportional to the quality of the heuristic. Using a strong heuristic, the improvement can be significant. Improvement is slight or negligible using a very weak heuristic. The reason is that when a weak heuristic is used, backed-up heuristic estimates are also weak, and preserving them (instead of pruning them) provides little benefit.

Our second observation is that if the size of memory is reduced far enough, and if nodes are pruned one at a time, a phenomenon called "thrashing" sometimes occurs in which the progress of search is slowed by repeated pruning and regeneration of the same few nodes. Because pruning several nodes at once can alleviate thrashing, SMAG*-prune can sometimes have an advantage over SMAG*-propagate when the size of memory is very small. But this advantage only occurs in the presence of thrashing, and thrashing can be avoided by other means. The topic of thrashing in memory-bounded search is beyond the scope of this paper. We simply note that when thrashing is not present, SMAG*-reopen and SMAG*-propagate consistently outperform SMAG*-prune.

Our third observation is that SMAG*-reopen and SMAG*-propagate perform significantly better than SMAG*-prune only on very challenging graph-search problems with combinatorially many paths between the start node and any other node, such as the sequence alignment problem. Although the Fifteen Puzzle is also a graph-search problem, the number of duplicate paths from the start node to any other node is relatively small. Backing-up heuristic estimates and pruning nodes can create an inconsistent heuristic that makes it possible to find better paths to already-expanded nodes. But this occurs infrequently enough in the Fifteen Puzzle that SMAG*-reopen and SMAG*-propagate enjoys no significant advantage over SMAG*-prune.

Conclusion

Kaindl and Khorsand (1994) describe a graph-search extension of memory-bounded A*, called SMAG*. In this paper, we show how to improve its performance on challenging graph-search problems with combinatorially many paths from the start node to any other node. We focus on the problem of how to handle the discovery of a better path to a previously-expanded node. First, we show that even when a consistent heuristic is used, this possibility cannot be avoided. Then we argue that instead of pruning all the descendents of a node after a shorter path is found, as SMAG* does, it is better to propagate the revised g-value to its descendents, as the original A* algorithm does. We consider both methods for propagating g-values used by the original A* algorithm and show how to extend them to memory-bounded A*. We demonstrate the improved performance that results experimentally.

Our work on memory-bounded A* is motivated by our interest in using it to solve large instances of the multiple sequence alignment problem. Linear-space algorithms such as IDA* and RBFS perform poorly on this problem due to excessive node regenerations. The algorithm that currently performs best is A* with Partial Expansion (Yoshizumi et al. 2000). This algorithm is the same as SMAG*, except it does not prune the open list when memory is full. (It is not a memory-bounded algorithm.) Therefore, it seems likely that an efficient implementation of SMAG* could solve larger instances of the multiple sequence alignment problem. In this paper, we make a modest contribution toward improving the efficiency of SMAG*. We are continuing to study other ways to improve its performance.

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grant IIS-9984952.

Appendix

Notation

s, t	Start and goal node respectively
G	The graph search space of the problem
$\hat{G}$	The explicit search graph kept in memory
$\Omega(n)$	Successors of node $n \in G$
$S(n)$	Successors of node $n \in \hat{G}$
$\Phi(n)$	Successors of node n along best path in $\hat{G}$
$p(n)$	Parent of node *n*
$c(u, v)$	Cost of edge from u to v
$d(n)$	Depth of node
$g(n)$	Cost from the start node s to node n
$h(n)$	Estimated cost from node n to the goal node
$f(n)$	$g(n) + h(n)$

Pseudocode

procedure SMAG* (s, T)
  $g(s) \leftarrow d(s) \leftarrow 0$, $F(s) \leftarrow g(s) + h(s)$
  $OPEN \leftarrow \{s\}$, $USED \leftarrow 1$, $CLOSED \leftarrow \emptyset$
  while $OPEN \neq \emptyset$ **do**
    $\mathcal{B} \leftarrow \{\beta | (\beta, x \in OPEN) \wedge (\forall x \Rightarrow F(\beta) \leq F(x))\}$
    Select $best \in \{\beta | (\beta, x \in \mathcal{B}) \wedge (\forall x \Rightarrow d(\beta) \geq d(x))\}$
    if $best \in T$ **then return** $F(best)$
    $succ \leftarrow$ next-successor(*best*) $\in \Omega(best)$
    $d(succ) \leftarrow d(best) + 1$
    $g(succ) \leftarrow g(best) + c(best, succ)$
    if $succ \neq t$ **and** $d(succ) = MAXD$ **then**
      $F(succ) \leftarrow \infty$
    else
      $F(succ) \leftarrow \max(F(best), g(succ) + h(succ))$
    if $succ \notin \hat{G}$ **then**
      MEM-CONTROL
      $S(best) \leftarrow S(best) \cup \{succ\}$
      $OPEN \leftarrow OPEN \cup \{succ\}$
      $USED \leftarrow USED + 1$
    else
      GRAPH-CONTROL($best, succ$)
    if completed(*best*) **then**
      BACKUP(*best*)
    if $S(best) = \Omega(best)$ **then**
      $OPEN \leftarrow OPEN \setminus \{best\}$
      $CLOSED \leftarrow CLOSED \cup \{best\}$
      if $\Phi(best) = \emptyset$ **then**
        CUT(*best*)
        Delete *best*
        $USED \leftarrow USED - 1$

procedure MEM-CONTROL
  if $USED = MAX$ **then**
    $\mathcal{W} \leftarrow \{\omega | (\omega, x \in OPEN) \wedge (\forall x \Rightarrow F(\omega) \geq F(x))\}$
    Select $worst \in \{\omega | (\omega, x \in \mathcal{W}) \wedge$
      $(\forall x \Rightarrow d(\omega) \leq d(x)) \wedge (S(\omega) = \emptyset)\}$
    $S(p(worst)) \leftarrow S(p(worst)) \setminus \{worst\}$
    if $p(worst) \in CLOSED$ **then**
      $CLOSED \leftarrow CLOSED \setminus \{p(worst)\}$
      $OPEN \leftarrow OPEN \cup \{p(worst)\}$
    Delete $worst$
    $USED \leftarrow USED - 1$

procedure BACKUP(n)
  $F_{min} \leftarrow F(l) | (l, x \in \Phi(n)) \wedge (\forall x \Rightarrow F(l) \leq F(x))$
  if $F_{min} > F(n)$ **then**
    $F(n) \leftarrow F_{min}$
    Reorder *OPEN* according to the new $F(n)$
    if $p(n) \neq$ **nil and** completed($p(n)$) **then**
      BACKUP($p(n)$)

procedure GRAPH-CONTROL($best, succ$)
  $\hat{succ} \leftarrow succ$'s equivalent node $\in \hat{G}$
  if $g(succ) \leq g(\hat{succ})$ **then**
    CUT($\hat{succ}$)
    $p(\hat{succ}) \leftarrow best$
    $d(\hat{succ}) \leftarrow d(best) + 1$
    $g(\hat{succ}) \leftarrow g(succ)$
    $F(\hat{succ}) \leftarrow \max(F(best), g(\hat{succ}) + h(\hat{succ}))$
    PROPAGATE($\hat{succ}$) /* used by SMAG*-prop only */
    Reopen $\hat{succ}$ /* used by SMAG*-reopen only */
    $S(best) \leftarrow S(best) \cup \{\hat{succ}\}$
    if $\hat{succ} \in OPEN$ **then**
      Reorder $OPEN$ according to new $F(\hat{succ})$
    else
      $CLOSE \leftarrow CLOSE \setminus \{\hat{succ}\}$
      $OPEN \leftarrow OPEN \cup \{\hat{succ}\}$
  Delete $succ$

procedure CUT(n)
  $S(p(n)) \leftarrow S(p(n)) \setminus \{n\}$
  if $\Phi(p(n)) = \emptyset$ **then**
    if $p(n) \in CLOSED$ **and** $p(p(n)) \neq$ **nil then**
      CUT($p(n)$)
      $CLOSED \leftarrow CLOSED \setminus \{p(n)\}$
      Delete $p(n)$
      $USED \leftarrow USED - 1$
    else if completed($p(n)$) **then**
      BACKUP($p(n)$)
  else if completed($p(n)$) **then**
    BACKUP($p(n)$)

procedure PROPAGATE(n)
  Reset the next successor index of n
  for each successor node $n_i \in S(n)$ **do**
    $d(n_i) \leftarrow d(n) + 1$
    $g(n_i) \leftarrow g(n) + c(n, n_i)$
    $F(n_i) \leftarrow \max(F(n), g(n_i) + h(n_i))$
    if $n_i \in OPEN$ **then**
      Reorder $OPEN$ according to new $F(n_i)$
    else
      $CLOSE \leftarrow CLOSE \setminus \{n_i\}$
      $OPEN \leftarrow OPEN \cup \{n_i\}$
    UPDATE(n_i)

References

Carillo, H. and Lipman, D. 1988. The multiple sequence alignment problem in biology. *SIAM Journal of Applied Mathematics* 48:1073–1082.

Chakrabarti, P.; Ghosh, S.; Acharya, A.; & DeSarkar, S. 1989. Heuristic search in restricted memory. *Artificial Intelligence* 47:197–221.

Hart, P; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, SSC-4(2):100–107.

Kaindl, H. and Khorsand, A. 1994. Memory-bounded bidirectional search. In Proc. of the Twelth National Conference on Artificial Intelligence (AAAI-94), 1359–1364.

Korf, R. 1985. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Korf, R. 1993. Linear-space best-first search. *Artificial Intelligence* 62:41–78.

Korf, R. and Zhang, W. 2000. Divide-and-conquer frontier search applied to optimal sequence alignment. Proc. of the Eighteenth National Conference on Artificial Intelligence (AAAI-00), 910–916.

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.

Mero, L. 1984. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence* 23:13–27.

Miura, T. and Ishida, T. 1998. Stochastic node caching for memory-bounded search. In Proc. of the 16th National Conference on Artificial Intelligence (AAAI-98), 450–456.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Tioga: Palo Alto, CA.

Nilsson, N. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufman: San Francisco, CA.

Reinefeld, A. and Marsland, T. 1994. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 16:701–710.

Russell, S. 1992. Efficient memory-bounded search methods. In Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92), 1–5.

Yoshizumi, T.;1 Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In Proc. of the Eighteenth National Conference on Artificial Intelligence (AAAI-00), 923–929.