Stochastic Node Caching for Memory-bounded Search

Teruhisa Miura and **Toru Ishida**

Department of Social Informatics, Kyoto University,
Kyoto 606-8501, Japan
{ miura, ishida }@kuis.kyoto-u.ac.jp

Abstract

Linear-space search algorithms such as IDA* (Iterative Deepening A*) cache only those nodes on the current search path, but may revisit the same node again and again. This causes IDA* to take an impractically long time to find a solution. In this paper, we propose a simple and effective algorithm called *Stochastic Node Caching* (SNC) for reducing the number of revisits. SNC caches a node with the best estimate, which is currently known of the minimum estimated cost from the node to the goal node. Unlike previous related research such as MREC, SNC caches nodes selectively, based on a fixed probability. We demonstrate that SNC can effectively reduce the number of revisits compared to MREC, especially when the state-space forms a lattice.

Introduction

Linear-space search algorithms such as IDA* (Korf 1985) perform a series of depth-first search iterations, gradually extending the search depth. Since they cache only nodes on the current search path, the amount of memory required by them is only linear to the depth of the current search path. By the virtue of small memory requirement, IDA* can solve some problems that A* cannot. Nevertheless, there are problems that neither A* and IDA* can solve; A* runs out of memory and IDA* takes an impractically long time as it cannot avoid revisiting the same node.

There have been several efforts to deal with such problems, by using a limited amount of memory to store information needed to avoid revisits. Among such efforts, MREC(Sen & Bagchi 1989) is a generalization of IDA* that uses a limited amount of memory to cache generated nodes. It caches as many nodes as possible, until the memory limit is reached; at that point, MREC starts iterative deepening in the same manner as IDA* from the cached nodes. Unfortunately, the efficiency of MREC's memory usage is sometimes poor, since the nodes generated and cached at the beginning of search are not necessarily the ones that are visited most.

In this paper, we propose a new mechanism called *Stochastic Node Caching* (SNC) that can effectively reduce the number of revisits. In contrast to MREC which caches nodes greedily, SNC caches nodes *selectively*. Whenever it expands a node, it decides whether to keep the node in memory by flipping a (possibly biased) coin. This selective caching allows SNC to store, with high probability, only nodes that are visited most frequently.

To evaluate the power of SNC, we apply MREC and SNC to the multiple sequence alignment problem. We show that, compared with MREC, SNC can reduce effectively the total number of revisits in the multiple sequence alignment problem wherein the state-space forms a lattice.

Previous Work

Before going into the details of SNC, we describe IDA* and MREC algorithms, upon which SNC is based. We use the following notations in this section.

n_s	Start node.
n_i	Successor of node n.
M	Maximum number of cached nodes.
N	Current number of cached nodes.
$c(n, n_i)$	Cost from node n to node n_i.
$g(n)$	Cost from the start node n_s to node n along the current search path.
$h(n)$	Estimated cost from node n to the goal node. A value of the estimate function for node n if not cached, and a value kept in memory for node n if cached.
$f(n)$	Estimated cost from the start node n_s through node n to the goal node.
θ	Global variable. θ is the node cutoff threshold for the current iteration.
θ'	Global variable. θ' is the node cutoff threshold for the next iteration.

IDA* IDA* performs a series of depth-first search iterations, increasing the threshold value θ. θ is used to prune branches during each iteration. For the next iteration, the value is increased to the minimum cost of all nodes that were generated but not expanded in the last iteration. Therefore, IDA* can eventually find the minimum cost path to a goal node. The IDA* algorithm is described as follows.

```
SEARCH (n_s)
  θ := h(n_s)
  θ' := ∞
  repeat until a goal node is found
    IDA*(n_s,0)
    θ := θ'
    θ' := ∞
  end repeat

IDA*(n, g(n))
  for each successor n_i of n do
    f(n_i) := g(n) + c(n, n_i) + h(n_i)
    if n_i is a goal node and f(n_i) ≤ θ then stop
    if f(n_i) ≤ θ then
      IDA*(n_i, g(n) + c(n, n_i))
    else if f(n_i) < θ' then θ' := f(n_i)
    end if
  end do
```

MREC MREC is a generalization of IDA* that uses a limited memory pool for caching nodes. Let M be the number of nodes that can be stored in the pool. When $M = 0$, MREC is identical to IDA*.

The two major differences between MREC and IDA* are as follows.

1. *Cached nodes:*

   MREC caches every node n_i with $h(n_i)$ when node n is expanded. IDA* caches only nodes on the current search path.

2. *Selection of a node to be expanded:*

   MREC memorizes $h(n_i)$ with n_i and uses it for selecting a node to be expanded. MREC expands node n_i if $f(n_i) = g(n)+c(n, n_i)+h(n_i)$ is less than the threshold value θ. IDA* expands nodes from left to right.

The estimated cost $h(n)$, which is memorized with node n, is used not to revisit the descendants of node n. After searching the descendants of node n, MREC updates $h(n)$ to $\min_i\{c(n, n_i) + h(n_i)\}$. Thus, $h(n)$ is equal to the maximum lower bound of estimated costs from node n to the goal node. After the number of cached nodes N reaches M, MREC performs iterative deepening in the manner identical to IDA* from the frontier nodes.

In this paper, however, we use a modified version of MREC, where the algorithm caches only expanded nodes, not generated ones.

The algorithm is as follows.

```
SEARCH (n_s)
  θ := h(n_s)
  θ' := ∞
  repeat until a goal node is found
    MREC(n_s, 0)
    θ := θ'
    θ' := ∞
  end repeat
```

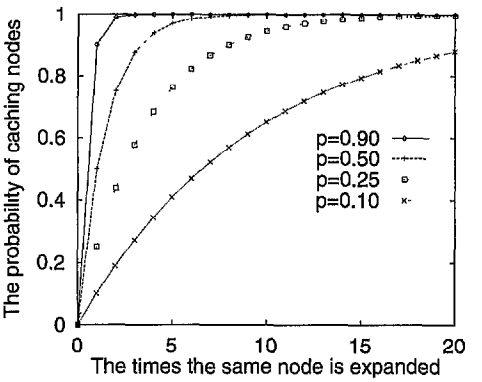

Figure 1: The probability of a node being cached

```
MREC(n, g(n))
  cutoff := ∞
  if N < M then cache n with h(n)
  for each successor n_i of n do
    f(n_i) := g(n) + c(n, n_i) + h(n_i)
    if n_i is a goal node and f(n_i) ≤ θ then stop
    if f(n_i) ≤ θ then
      MREC(n_i, g(n) + c(n, n_i))
    else if f(n_i) < θ' then θ' := f(n_i)
    end if
    if h(n_i) + c(n, n_i) < cutoff then
      cutoff := h(n_i) + c(n, n_i)
    end if
  end if
  h(n) := cutoff
```

Stochastic Node Caching

The aim of SNC is to efficiently reduce the number of revisits. Just like MREC, SNC has the memory that can store up to M nodes. It takes an additional parameter p, which is the probability of a node being cached every time it is expanded. It follows that the overall probability of a node being stored after it is expanded t times is $1 - (1 - p)^t$; the more frequently the same node is expanded, the higher the probability of it being cached becomes. Figure 1 shows the probability of a node being cached versus the number of times the node is expanded for various values of p. SNC is identical to IDA* and MREC when $p = 0$ and $p = 1$, respectively.

Let RANDOM be a function that returns a real number from zero to one at random. The SNC algorithms is as follows.

```
SEARCH (n_s)
  θ := h(n_s)
  θ' := ∞
  repeat until a goal node is found
    SNC(n_s, 0)
    θ := θ'
    θ' := ∞
  end repeat
```

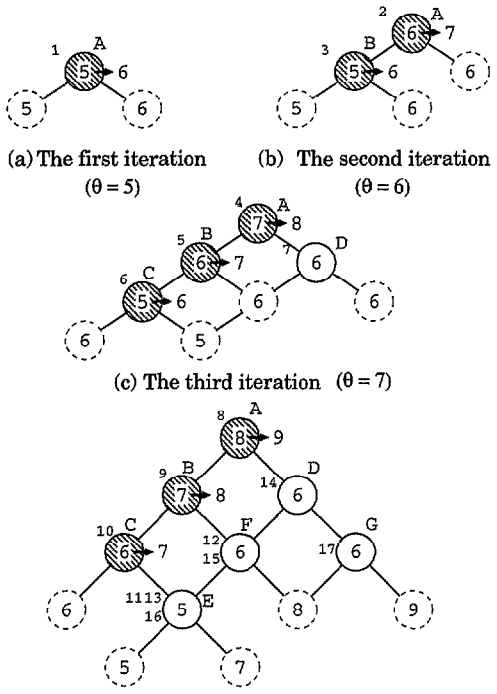

Figure 2: Search behavior of MREC

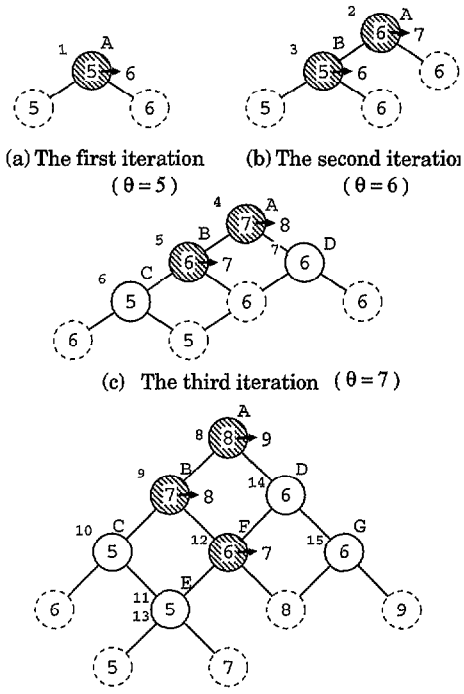

Figure 3: Search behavior of SNC

```
SNC(n, g(n))
  cutoff := ∞
  if N < M and RAND ≤ p then
    cache n with h(n)
  end if
  for each successor n_i of n do
    f(n_i) := g(n) + c(n, n_i) + h(n_i)
    if n_i is a goal node and f(n_i) ≤ θ then stop
    if f(n_i) ≤ θ then
      SNC(n_i, g(n) + c(n, n_i))
    else if f(n_i) < θ' then θ' := f(n_i)
    end if
    if h(n_i) + c(n, n_i) < cutoff then
      cutoff := h(n_i) + c(n, n_i)
    end if
  end if
  h(n) := cutoff
```

The difference between SNC and MREC is as follows.

- *Selection of a node to be cached:*
  MREC always caches as many expanded nodes as possible, while SNC stochastically caches expanded nodes.

This difference in the two algorithms results in a difference in the total number of visits. Figures 2 and 3 illustrate the search behaviors of the two algorithms when $M = 3$. In these examples, we assume unit cost for each edge. The shaded circles represent the cached node. The hollow circles represent the expanded nodes and the dashed circles represent the generated but not expanded nodes. The digits in the circles represent $h(n)$ at the current iteration. The digits on the right side of the arrow represent $h(n)$ after searching the descendants of node n. The digits on the left side of the circle represent the order of visiting a node.

In Figure 2(a),(b) and (c), MREC caches the first three nodes (A, B and C) in order of their expansion. On the other hand, SNC is less likely to cache node C as shown in Figure 3(c), because of its stochastic behavior. In this case, SNC is more likely to cache node F, which is visited more often than node C. By caching node F, SNC can avoid visiting descendants of node F afterwards. In Figure 2(d), for example, MREC visits node E and F, after visiting node D. On the other hand, SNC does not visit node E and F, as shown Figure 3(d).

Because of the stochastic caching, SNC does not cache always a node with its ancestors unlike MREC. SNC's search frontier is a jumbled collection of nodes in the search space.

Evaluation

Example Problems

We applied the MREC and SNC algorithms to the *Multiple sequence alignment* problem. This problem is of extracting the common features among several sequences. Multiple sequence alignment is used in various ways for biological sequence analysis. Protein sequences that have a similar biological function tend to share a common pattern. The alignment for sequences of such proteins is useful for

predicting the function or the three-dimensional structure of a protein. If the sequence of the protein which has an unknown function is similar to an alignment, the protein is likely to have the same function and three-dimensional structure as proteins with the same sequence alignment. It is important to align as many sequences as possible, since it will greatly improve the reliability of the alignment. (Boguski *et al.* 1992).

A biological sequence is composed of alphabetic characters representing its constituents. For example, the protein sequence consists of 20 amino acids. The following figure shows a part of the aligned sequences.

```
Hal VNKMDLVD--YGESEYKQVVEEV-KDLLTQVRFDSENAK
Met VNKMDTVN--FSEADYNELKKMIGDQLLKMIGFNPEQIN
Tha INKMDATSPPYSEKRYNEVKADA-EKLLRSIGFK-D-IS
Thc VNKMDMVN--YDEKKFKAVAEQV-KKLLMMLGYK-N-FP
Sul INKMDLADTPYDEKRFKEIVDTV-SKFMKSFGFDMNKVK
Ent VNKMDAIQ--YKQERYEEIKKEI-SAFLKKTGYNPDKIP
Pla VNKMDTVK--YSEDRYEEIKKEV-KDYLKKVGYQADKVD
```

Hyphens, or gaps, are inserted into the sequences so that the same character occupy the same column.

We formulate the multiple sequence alignment problem as a search problem (Carrillo & Lipman 1988; Ikeda & Imai 1994). The following notations are used.

d	Number of sequences to be aligned.
S_k	k-th sequence.
$L(S_1, \ldots, S_d)$	State-space which is a d-dimensional lattice. k-th axis corresponds to S_k. This lattice is the Cartesian product of the d sequences.
$L(S_i, S_j)$	Two dimensional lattice which is the projection of $L(S_1, \ldots, S_d)$ on the plane determined by S_i and S_j.
γ	Path in $L(S_1, \ldots, S_d)$
γ_{ij}	Path in $L(S_i, S_j)$. It is the projection of γ on $L(S_i, S_j)$.
s	The start node.
t	The goal node.
t_{ij}	Node in $L(S_i, S_j)$. It is the projection of the goal node t on $L(S_i, S_j)$.
u, v	Node in $L(S_1, \ldots, S_d)$.
u_{ij}	Node in $L(S_i, S_j)$. It is the projection of u on $L(S_i, S_j)$.
$c(u, v)$	Cost of the edge (u, v) in $L(S_1, \ldots, S_d)$.
$c(u_{ij}, v_{ij})$	Cost of the edge (u_{ij}, v_{ij}) which is the projection of (u, v) on $L(S_i, S_j)$.
$m(\gamma)$	Cost of γ.
$m(\gamma_{ij})$	Cost of γ_{ij}.
$h^*_{ij}(v_{ij})$	Cost of the shortest path from v_{ij} to the goal node t_{ij} in $L(S_i, S_j)$.

We can define the multiple sequence alignment problem as the problem of finding the shortest path from the start node to the goal node in the d-dimensional lattice $L(S_1, S_2, \ldots, S_d)$. The cost of the path in the d-dimensional lattice is defined as follows.

$$m(\gamma) = \sum_{1 \le i \le j \le d} m(\gamma_{ij}).$$

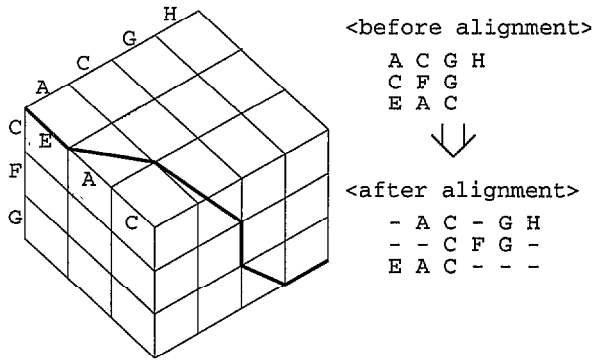

Figure 4: State-space representation

The cost of the edge in this lattice is defined as follows.

$$c(u, v) = \sum_{1 \le i \le j \le d} c(u_{ij}, v_{ij}),$$

where the cost of the edge $c(u_{ij}, v_{ij})$ is given by the PAM-250 matrix(Dayhoff, Schwartz, & Orcutt 1978) which represents the mutation distance between two amino acids or characters. The number of operators is usually $2^d - 1$, because a general node on the lattice $L(S_1, \ldots, S_d)$ has $2^d - 1$ descendant nodes.

Figure 4 depicts the state-space representation of the multiple sequence alignment problem of three sequences $S_1 =$ `ACGH`, $S_2 =$ `CFG` and $S_3 =$ `EAC`. In this 3-dimensional lattice, the top left-hand corner is the start node and the bottom right-hand corner is the goal node. A path from the start node to the goal node determines an alignment of the three sequences; for example, the path drawn with bold line in Figure 4 corresponds to the following alignment.

```
S1  -AC-GH
S2  --CFG-
S3  EAC---
```

In this example, the cost of the path is calculated by

$$\begin{aligned} m(\gamma) &= m(\gamma_{12}) + m(\gamma_{13}) + m(\gamma_{23}) \\ &= c(-,-) + c(\mathtt{A},-) + c(\mathtt{C},\mathtt{C}) + c(-,\mathtt{F}) \\ &\quad + c(\mathtt{G},\mathtt{G}) + c(\mathtt{H},-) + \cdots. \end{aligned}$$

Using this formulation, Ikeda *et al.* (Ikeda & Imai 1994) successfully applied the A* algorithm to the multiple sequence alignment problem. Though it could align up to seven sequences, there is little or no hope of it solving more than seven sequences by A* because of its large memory requirement. Therefore, we first applied the linear-space search algorithm IDA* to this problem. We used the same cost function(the PAM-250 matrix and the gap cost of 8) and the same seven sequences[1] as in Ikeda *et al*'s experiments.

[1]These are elongation factor TU(EF-TU) of Haloarcula marismortui and Methanococcus vannielii, and elongation factor 1α(EF-1α) of Thermoplasma acidophilum, Thermococcus celer, Sulfolobus acidocaldarius, Entamoeba histolytica and Plasmodium falciparum.

Table 1: The number of visited nodes: protein sequence alignment problem

M	MREC	SNC		
		$p = 0.1$	$p = 0.01$	$p = 0.001$
5000	1,932,448,612	1,588,688,060(0.82)	1,418,349,564(0.73)	1,856,357,576(0.96)
2000	12,099,281,720	10,755,536,884(0.89)	5,894,480,820(0.49)	4,972,223,330(0.41)
1000	34,684,660,730	32,337,946,952(0.93)	20,402,428,944(0.59)	12,683,118,881(0.37)

The average sequence length is 430. These sequences are elongation factors from various species, obtained from the Genbank database [2]. The heuristic estimate function in Ikeda *et al*'s experiment is constructed as

$$h(v) = \sum_{1 \le i \le j \le d} h^*_{ij}(v_{ij}),$$

where h^*_{ij} is calculated by the dynamic programming for each pair of S_i and S_j before the search algorithm is applied.

After several trials, we found that IDA* could align only four sequences in our computing environment, owing to the large number of revisits. Thus, this is not a toy problem for either A* or IDA*.

Results

We applied the two algorithms MREC and SNC to the multiple sequence alignment problem. We measured their performance by the cumulative number of visited nodes, to which the running time of the algorithms is proportional.

As mentioned before, we used a modified version of MREC that caches expanded nodes, not generated ones. The reason for this modification is to ensure fair comparison with SNC; in the problem like multiple sequence alignment in which branching factor is large and the edge costs have many significant digits, the original MREC caches a lot of nodes that will never be visited. In addition, since we evaluate the performance with respect to the number of visits, not the number of expansions, this modification does not work disadvantageously for MREC; it only improves the efficiency of MREC's memory usage.

The following are observed.

Table 1 shows the result of aligning seven protein sequences. The figure in a parenthesis shows the ratio of SNC value for MREC's corresponding value. In this case, each node has $2^7 - 1$ predecessors and successors. Because there is an enormous number of paths through the same node, linear-space search visits the same node again and again. These results show that SNC reduces the number of revisits effectively compared to MREC. The advantage of SNC over MREC increases as the cache size M decreases. When $M = 1000$ and $p = 0.001$, SNC reduces the number of visits by 63% compared to MREC. In other words, SNC can solve this problem three times faster than MREC.

The major results obtained from the above experiments are as follows.

[2]http://www.genome.ad.jp

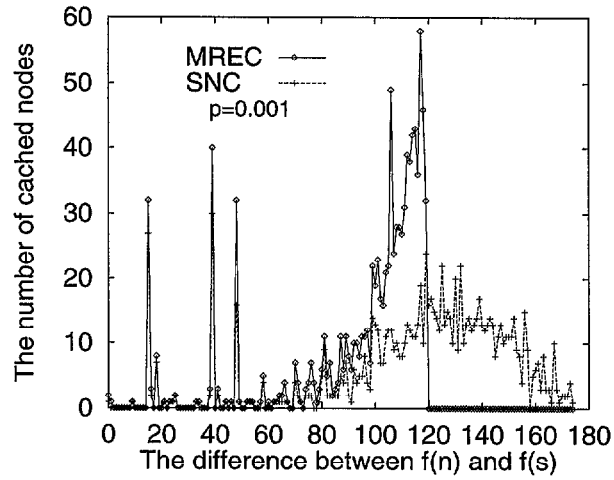

Figure 5: The locations of cached nodes: Protein sequence alignment ($M = 1000$)

1. *SNC can cache nodes that are far from the root node.* Figure 5 shows the location of the cached nodes in the search space. The x-axis shows the difference between the initial estimated cost for node n and that for the start node, i.e., $f(n) - f(s)$. The y-axis shows the number of cached nodes of which $f(n) - f(s)$ value correspond to x-value. The difference between the initial estimate cost $f(s)$ for the start node and the optimal cost $f^*(t)$ to the goal node is 244 in this problem. The value of $f(n) - f(s)$ is 0 and 244 when n is the start and the goal node, respectively. Both algorithms expand nodes in ascending order of initial estimated cost $f(n)$. As shown in Figure 5, SNC can cache nodes that are far from the root node. MREC caches nodes in the order of node expansion. On the other hand, SNC caches only nodes that are visited for many times. As a result, SNC can avoid to revisit such nodes that are visited for many times.

2. *The probability of node caching determines the performance of SNC.*

   When the probability of node caching is low, SNC can cache nodes which are located far from the start node. Thus, SNC may reduce revisits by the effect of caching such nodes. Until MREC caches the maximum number of nodes, however, SNC visits more nodes and cannot outperform MREC, because SNC only caches a part of

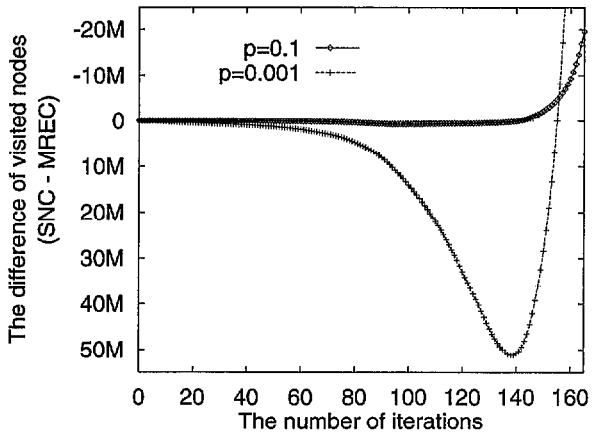

Figure 6: The difference of visited nodes (SNC – MREC) Protein sequence alignment($M = 1000$)

nodes cached by MREC. This tradeoff affects the overall performance of SNC. In the protein sequence alignment problem($M = 1000$), MREC caches M nodes at the 89th iterations, and then MREC performs IDA* for descendants of the cached frontier nodes. Most of nodes cached by SNC after that time are not cached by MREC. Figure 6 shows the difference in the numbers of nodes visited by SNC and MREC. In general, the difference increases until some point of time, then decreases. Finally, the number of nodes visited by SNC becomes less than that by MREC.

3. *SNC improves the search efficiency for such a problem where the number of revisits is enormous using linear-space search algorithms*

   In the seven sequence alignment problem, each node has $2^7 - 1$ successors. Furthermore, the value of $f(n) - f(s)$ varies between 0 to 244. Since linear-space search algorithms perform depth-first search for each value of $f(n)$, the number of iterations is large in the alignment problem. As a result, on average, IDA* visits each node at least 40,000 times for the seven sequence alignment problem.

   For such problems, SNC can reduce the number of revisits efficiently by caching nodes selectively. On the other hand, SNC may be less effective in the case where linear-space search does not visit the same node many times.

Related Work

There has been a lot of work on memory-bounded search. The work can be classified as follows.

1. Algorithms based on A* but use the limited-memory (MA*).
2. Algorithms that reduce revisits in the iterative deepening.
   (a) Globally control the node expansion (RBFS, Tayler and Korf's method).
   (b) Control the number of iterations in IDA* (IDA*_CR, DFS*).
   (c) Cache nodes and utilize them to avoid revisits (MREC, SNC).

The mechanism of SNC can be applied to algorithms based on the iterative deepening, that is, all of the classes 2(a),(b) and (c). We briefly refer to other memory-bounded search algorithms.

MA* (Chakrabarti *et al.* 1989)

The MA* algorithm dynamically caches the best nodes that were generated within memory constraints. MA* maintains the two sets OPEN and CLOSED nodes, just as A*. Until MA* caches the maximum number of nodes, it behaves like A*. When the number of cached nodes reaches the limit, MA* begins to prune the node with the highest cost in the OPEN set.

Though it surpasses MREC in the light of efficient usage of memory, it is reported that the overhead of maintaining these two sets are prohibitively expensive compared to the algorithms based on iterative deepening(Korf 1993).

RBFS (Korf 1993)

Korf proposed a linear-space best-first search algorithm, called Recursive Best-First Search. This algorithm explores nodes in a best-first order and expands fewer nodes than IDA* with a nondecreasing cost function. RBFS can be combined with MREC. The mechanism of SNC is also compatible with RBFS.

Tayler and Korf's method (Taylor & Korf 1993)

Tayler and Korf developed a method using a finite-state machine for pruning duplicate nodes. Unlike MREC and SNC, this method does not cache nodes, but keeps the structure of the state-space as the finite-state machine. At first, the method learns the finite-state machine. The learning phase consists of two steps. First, a small breadth-first search in the state-space is performed and a set of operator strings that produce duplicate nodes is detected. The operator strings represent portions of node generation paths. It then constructs the finite-state machine that recognizes such operator strings. This method prunes duplicate nodes by its finite-state machine.

This method is effective for problems in which a string of operators has the same cost at any node, such as 15-puzzle. In the space of the multiple sequence alignment problems, the edge costs have many significant digits, since it depends on which characters correspond to its edge. Because of this, the same string of operators does not always take the same value. Consequently, this method may not be effective for problems such as the multiple sequence alignment, though SNC mechanism can be incorporated in this method as well.

IDA*_CR (Sarkar *et al.* 1991)

IDA*_CR tries to decrease revisits by reducing the number of iterations of IDA*. Each iteration of IDA*_CR is depth-first branch-and-bound search, not depth-first search as in IDA*. At first, IDA*_CR performs an iteration with the initial estimate for the start node, $f(s)$, as the upper

bound for cutoff. Then IDA*_CR performs iterations, as increasing the upper bound for cutoff. By introducing the idea of SNC, IDA*_CR can reduce revisits in each iteration.

DFS* (Vempaty, Kumar, & Korf 1991)

Unlike IDA*_CR, DFS* performs depth-first search iterations. DFS* sets the threshold of the next iteration to the value larger than the minimum cost of all nodes which were generated but not expanded on the last iteration. When the solution is found, DFS* performs the depth-first branch and bound with the cost of this solution as its initial upper bound to find the optimal solution. By introducing the idea of SNC, DFS* can also reduce revisits in each iteration.

Conclusions

We have proposed *Stochastic Node Caching* (SNC) to reduce the number of node revisits, the most serious disadvantage of linear-space search. It caches nodes stochastically, not deterministically. To demonstrate the effectiveness of this method, we applied SNC and MREC to the multiple sequence alignment problem. We found that SNC can effectively reduce the total number of node visits compared to MREC. The greatest effects are obtained in the case where the state-space includes many paths to the same node (especially a lattice), and in the case where linear-space search performs a large number of iterations of depth-first search. For the multiple sequence alignment problem, for example, SNC visits only one-third the number of nodes visited by MREC.

We may use frequency of visits explicitly as part of the SNC scheme. SNC may be revised in order to swap out cached nodes by frequency. This idea is similar to MA*. The overhead of maintaining cached nodes by MA* seems to be expensive in the multiple sequence alignment problem too. We will have to apply approximate techniques to the SNC scheme. We will evaluate and compare SNC with various selective caching techniques in our future research.

Acknowledgments

We would like to thank Yasuhiko Kitamura, Hideyuki Nakanishi, and Masashi Shimbo for comments that greatly improved this paper.

References

Boguski, M. S.; Caballero, L.; Eisenberg, D.; Elliston, K.; Luthy, R.; Rice, P. M.; and States, D. J. 1992. *Sequence Analysis Primer*. UWBC Biotechnical Resource Series. Oxford University Press.

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

Chakrabarti, P. P.; Ghose, S.; Acharya, A.; and de Sarkar, S. C. 1989. Heuristic search in restricted memory. *Artificial Intelligence* 41:197–221.

Dayhoff, M. O.; Schwartz, R. M.; and Orcutt, B. C. 1978. *Atlas of protein sequence and structure*, volume 5. Washington DC: National Biomedical Research Foundation. 345–352.

Ikeda, T., and Imai, H. 1994. Fast A* algorithms for multiple sequence alignment. In Miyano, S.; Akutsu, T.; Imai, H.; Gotoh, O.; and Takagi, T., eds., *Proceedings of Genome Informatics Workshop 1994*, 90–99. Tokyo,Japan: Universal Academy Press.

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

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

Sarkar, U. K.; Chakrabarti, P. P.; Ghose, S.; and Sarkar, S. C. D. 1991. Reducing reexpansions in iterative-deepening search by controlling cutoff bounds. *Artificial Intelligence* 50:207–221.

Sen, A. K., and Bagchi, A. 1989. Fast recursive formulations for best-first search that allow controlled use of memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 297–302.

Taylor, L. A., and Korf, R. E. 1993. Pruning duplicate nodes in depth-first search. In *Proceedings of the eleventh National Conference on Artificial Intelligence (AAAI-93)*, 756–761.

Vempaty, N. R.; Kumar, V.; and Korf, R. E. 1991. Depth-first vs best-first search. In *Proceedings of the ninth National Conference on Artificial Intelligence (AAAI-91)*, 434–440.