Pergamon

Mathl. Comput. Modelling Vol. 22, No. 4-7, pp. 397-414, 1995
Copyright©1995 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0895-7177/95 $9.50 + 0.00

0895-7177(95)00146-8

Improving Search Efficiency Using Possible Subgoals

J. F. DILLENBURG AND P. C. NELSON*
Department of Electrical Engineering and Computer Science (M/C 154)
University of Illinois at Chicago
851 South Morgan Street, Chicago, IL 60607-7053, U.S.A.
nelson@eecs.uic.edu

Abstract—A primary function of most advanced traveler information systems involves the ability to plan optimal routes. Although the route planning ability of ATI systems can be facilitated using centralized computing resources, most ATI systems currently under development use in-vehicle computational resources. The primary advantage of this type of approach is fault tolerance; vehicles can continue to plan routes even in the absence of a centralized computing center. However, there are significant challenges associated with planning routes using in-vehicle electronics. These challenges result from trying to keep the cost of the in-vehicle electronics to a reasonable level. Thus, the route planning algorithm must be both space and time efficient to compensate for the limited amount of memory and computational resources available. This paper describes a new heuristic search algorithm that can be used for in-vehicle route planning.

Certain types of heuristic search problems may contain an identifiable subgoal. This subgoal can be used to break a search into two parts, thus reducing search complexity. However, it is often the case that instead of one subgoal, there will be many possible subgoals, not all of which lie on an optimal path to the eventual goal. For this case, the search cannot be simply broken into two parts. However, the possibility of reducing the search complexity still exists.

Previously, Chakrabarti *et al.* [1] developed a search algorithm called Algorithm I which exploits islands to improve search efficiency. An island, as defined by Chakrabarti *et al.* [1], is a possible subgoal. Previously, this algorithm had only been analyzed theoretically. In this paper, some experimental results comparing Algorithm I to A* are presented. Algorithm I has also been generalized to cases where more than one possible subgoal can appear on an optimal path. Two new heuristic island search algorithms have been created from this generalization and are shown to provide even further improvement over Algorithms I and A*. The use of possible subgoals can make any type of search more efficient, not just A*. Modifications are discussed which describe how to incorporate possible subgoal knowledge into IDA* search.

Keywords—Heuristic search, Route planning, Possible subgoals, Islands, Iterative deepening, Semi-admissible search.

NOMENCLATURE

NOTATION FOR ISLAND SEARCH ALGORITHMS

s	start node
g	goal node
m, n	other nodes
$i, i_1, i_2, \ldots, i_k$	island nodes
$c(m, n)$	cost of direct path from m to n
$g^*(n)$	cost of least cost path from s to n
$h^*(n)$	cost of least cost path from n to g
$h^*(n/i)$	cost of least cost path from n to g containing i

*Author to whom all correspondence should be addressed.

We would like to thank the anonymous referees for their contributions to this paper.

$h^*(n,m)$	cost of least cost path from n to m
$f^*(n)$	$g^*(n) + h^*(n)$
$g(n), h(n), h(n/i), h(n,m), f(n)$	estimates of $g^*(n)$, $h^*(n)$, $h^*(n/i), h^*(n,m)$, and $f^*(n)$
OPEN1, OPEN2, CLOSED1, CLOSED2	lists
IS	set of island nodes.
E	lower bound on number of islands on optimal path to the goal
$/s/$	number of elements in set s
$\mathrm{IS}_p(n)$	set of island nodes already passed through along a path from s to node n
P_{opt}	set of nodes on an optimal path from s to g

NOTATION FOR ALGORITHM $\mathrm{I_{np}}$

$h^*(n/t)$	least cost path from n to g constrained to contain all the islands in set t
$h(n/t)$	estimate of $h^*(n/t)$

1. INTRODUCTION

For some heuristic search problems, a set of possible subgoals can be identified. These possible subgoals really amount to secondary heuristic information which can potentially help guide the search by breaking it into multiple components. For example, given a starting node s where the optimal path from s to g is expected to pass through some subgoal i, the search can be broken into two components: s to i and i to g. By doing so, the maximum search depth is reduced, providing a potentially significant improvement in search performance. The subtleties arise as there are usually many subgoals, and furthermore, only some unknown subset of the subgoals will actually lie on an optimal path.

Many search problems contain identifiable subgoals which can be used to guide the search and thereby reduce the search cost. Take, for instance, the real-life problem of navigating across a city. Say it is necessary to travel across the city and it is known that the city is divided by a river, see Figure 1. Since the river must be crossed on a bridge, this information can be used to help us plan our route. In this example, one of the two bridges b_1 or b_2 will necessarily be a subgoal in our search.

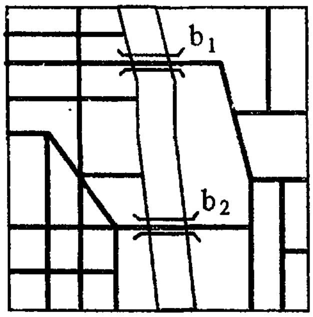


Figure 1. Example search problem with identifiable islands. Roads are represented as dark lines. The two bridges b_1 and b_2 form an island set.

Actually, the word subgoal is a slight misnomer and the term *island* is preferable. It is important here to note the difference between islands and subgoals. The best way to understand islands is to group them together into what is called an island set. An island set is defined by the fact that at least one element of the set will be a node on an optimal path. An island is now defined as simply being one element of the island set. The example in Figure 1 contains an island set with two elements, one for each bridge across the river. One of the two elements in this list must be on an optimal path (i.e., we must cross one of the bridges to get to the other side). Another example of an island set is the set of all nodes in the search space. It can be shown that, in this extreme case, the island set would not provide any efficiency improvements.

Island search generalizes search by allowing the addition of islands. Island search will be shown to be superior to normal A* search in terms of search efficiency when islands are involved. Island search uses the same heuristic estimate that A* search would use. The increase in efficiency is due solely to the additional information provided by knowing the location of islands.

In this paper, it is assumed that the possible subgoal information is known *a priori*. In other words, island search will only improve search efficiency for problem domains where possible subgoal information is known. For problem domains where possible subgoal information is easy to obtain, but not reliable (i.e., it is possible for no islands to be on an optimal path), a ε-admissible (not optimal) version of island search can be used, see Section 5. As an example of this, consider Figure 1 once again. By testing to see if a straight line between the start and the destination intersects any rivers, it is possible to determine which bridges are possible subgoals. This method is not guaranteed to produce the correct results every time and so the ε-admissible version of island search should be used.

The rest of this paper is organized into five major sections. Section 2 contains an overview of the past work done on island search along with some new theoretical and experimental results. Section 3 presents a new concept in island search which we call multiple level island search. More theoretical and experimental results will show that multiple level island search is superior to the traditional A* search under certain circumstances. Section 4 presents a IVHS route planning problem and shows how island search can be used to improve route planning efficiency. Section 5 shows how the island search concept can be extended into other paradigms such as depth-first iterative deepening. Finally, Section 6 summarizes the results and mentions future research possibilities.

2. SINGLE LEVEL ISLAND SEARCH

Island search can be divided into two categories: single and multiple level island search. The former, Algorithm I, was first introduced in [1]. For single level island search, it is assumed that only one subgoal will appear on a path from the start to the goal. Multiple level island search generalizes this idea to cover cases where more than one island may be on a path from the start to the goal. Multiple level island search will be discussed further in Section 3. In this section, Algorithm I will be described and some empirical results will be shown comparing the relative merits of island based search to the traditional A* search [2,3].

Description of Algorithm I

Algorithm I is basically an A* algorithm modified in two ways. First, there are two OPEN and CLOSED lists, OPEN1, OPEN2, CLOSED1, and CLOSED2. Normal A* search uses only one OPEN and CLOSED list. The extra lists are used in Algorithm I to keep track of which nodes are from a path that contains an island. The second difference is that there are two heuristic estimates employed, $h(n)$ and $h(n/i)$. The heuristic is identical to the heuristic used in A*. The heuristic $h(n/i)$ estimates the cost of a path from n to the goal constrained to contain an island i. Any nodes which are generated along a path up to and including an island are placed in OPEN1. After reaching an island, newly generated nodes are placed in OPEN2. Nodes which have already been expanded are placed in CLOSED1 if they were chosen from OPEN1, or in CLOSED2 if chosen from OPEN2, or the node is an island. A detailed description of Algorithm I can be found in [1].

A search algorithm is said to be *admissible* if it finds an optimal path whenever an optimal path exists. A proof of the admissibility of this algorithm when at least one island lies on an optimal path can also be found in [1]. Also proved is the fact that Algorithm I expands no more nodes than A*.

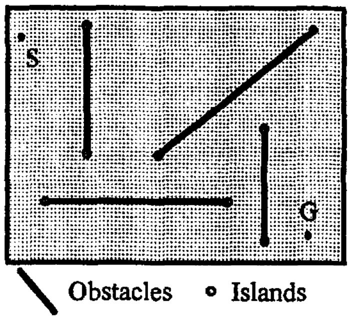
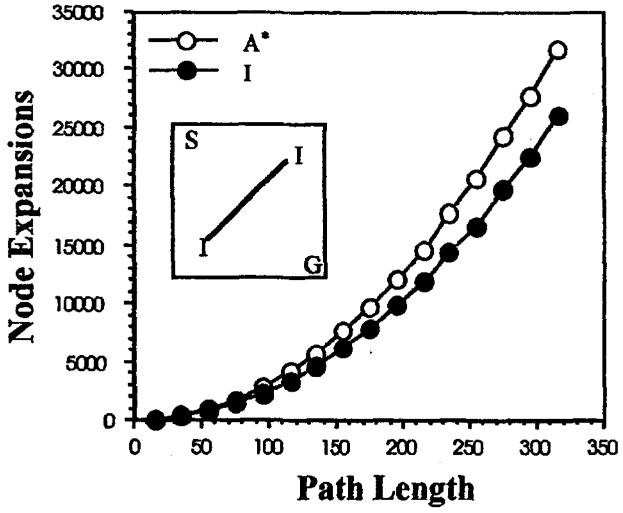
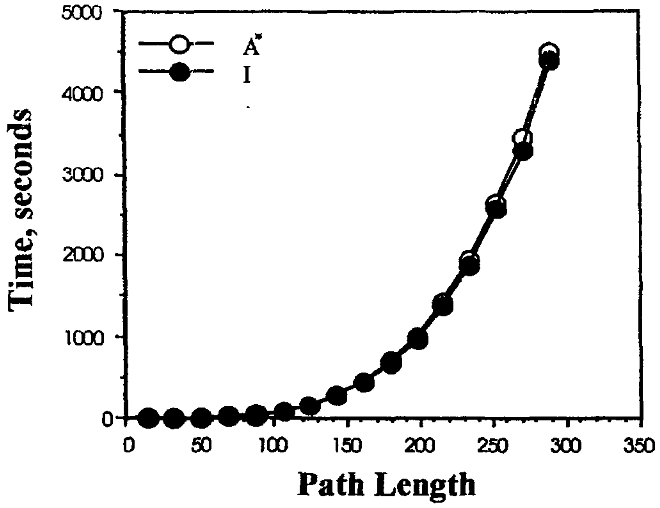


Figure 2. Sample grid search problem used to test island search algorithms.



(a) Node expansion comparison between Algorithm I and A* search. One obstacle and two islands.



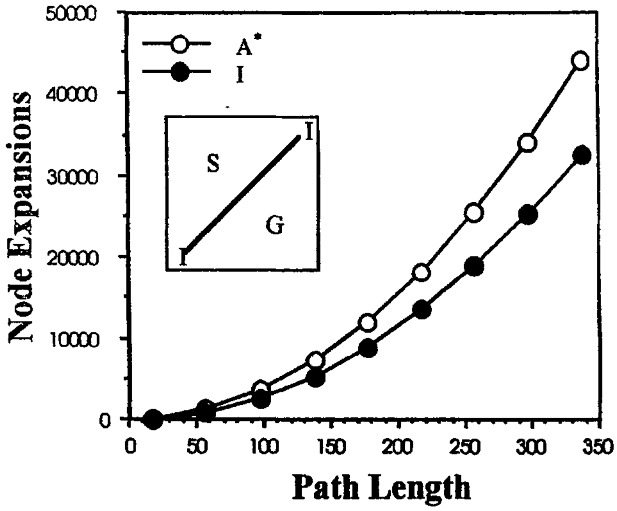
(b) Elapsed CPU time comparison between Algorithm I and A* search. One obstacle and two islands.

Figure 3.

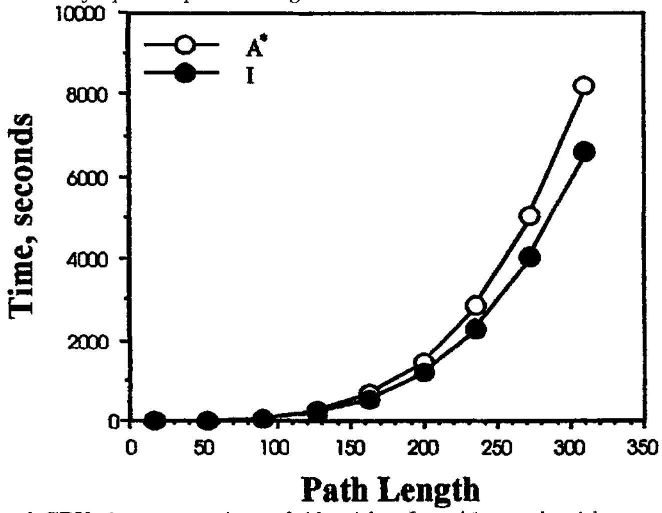
Empirical Results

Although it has been shown that using islands may decrease the number of node expansions, there is still no empirical evidence to show exactly how much of an improvement can be expected. In order to gather such empirical evidence, Algorithm I and A* were tested using a rectilinear grid search. The object of the search was to find the shortest path from point s to point g on a grid. To make the search more interesting, obstacles were placed between s and g, see Figure 2. The search used a unity cost function and the air-distance (straight-line) heuristic. The air-distance heuristic is monotonic for both Algorithm I and A*.

The rectilinear grid search problem described here contains islands at the edges of obstacles. The reason for this is that the edge of at least one obstacle needs to be skirted in order to reach the goal. This assumes that the optimal path is blocked by at least one obstacle. The island set in this case is the set of all the obstacle edges. If the optimal path were not blocked by any obstacles, then there would not be any acceptable island set to use.



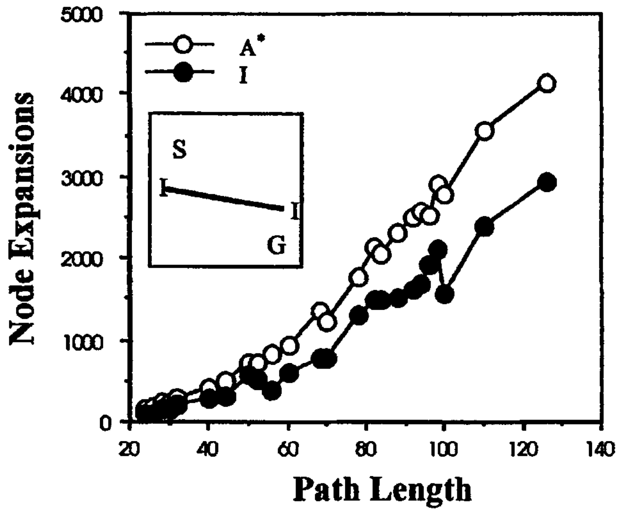
(a) Node expansion comparison of Algorithm I to A* search with one obstacle, two islands, and every optimal path through an island.



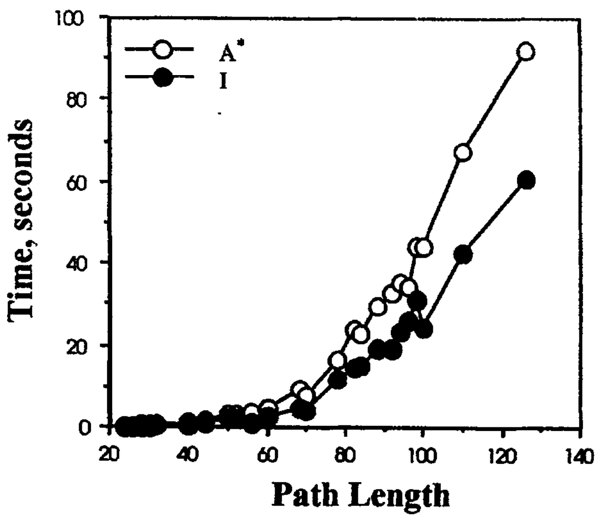
(b) Elapsed CPU time comparison of Algorithm I to A* search with one obstacle, two islands, and every optimal path through an island.

Figure 4.

Figures 3a and 3b show the results from some test cases where one obstacle was placed directly between the start and the goal. The path length from the start to the goal was varied and the obstacle size was changed in proportion with the start to goal distance. Each point on Figure 3a and 3b represents one test case. For this test, it is known that there exists at least one island on an optimal path, so no preprocessing is required to determine the number of islands. The results in Figure 3a show Algorithm I yielding a modest node expansion improvement over A*. Figure 3b shows that Algorithm I is only slightly faster than A*. For all the cases in these figures, at least one of the islands lies on an optimal solution, thus satisfying the admissibility requirement of Algorithm I. However, the optimal path is not necessarily unique because of the grid constraints. The result is that there may be optimal paths which do not contain an island. Intuitively, the island is unnecessary and would seem to provide little, if any, benefit for these cases.



(a) Node expansion comparison of Algorithm I to A* search. One random obstacle, two islands, every optimal path through an island.



(b) Elapsed CPU time comparison of Algorithm I to A* search with one obstacle, two islands, and every optimal path through an island.

Figure 5.

A similar set of tests were run with the exception that for the cases used in these tests, every optimal path is known to pass through an island. The results of these tests are shown in Figures 4a and 4b. The islands are necessary in these cases and, as would be expected, provide a greater benefit. These cases are probably more realistic as direct paths are usually going to be obstructed in such a way as to necessitate passing through an island.

For the test cases used in Figures 3 and 4, the obstacles were placed perpendicular to the line segment formed by S and G. Since the heuristic being used is the air distance heuristic, the search will attempt to follow this straight-line path. Therefore, the placement of these obstacles favors Algorithm I. Figures 5a and 5b are similar to Figures 4a and 4b, except that for these cases the obstacles were placed at random angles with respect to S and G while insuring every optimal path goes through an island. Each point on the graph in Figures 5a and 5b is an average of 10 random obstacle placements.

3. MULTIPLE LEVEL ISLAND SEARCH

The test results presented in Section 2 show that Algorithm I does offer some improvement over A*. However, Algorithm I reverts back to A* after it passes through an island. Often an optimal path will pass through several islands on the way to the goal. Clearly, this is the case for the grid search instance shown in Figure 2. Multiple level island search generalizes the idea of island search to cover cases where it is possible to encounter one or more islands on an optimal path from the start to the goal.

In the following sections, two types of multiple level island search algorithms will be presented. The first one, Algorithm I_n, is most similar to Algorithm I. Instead of searching directly for the goal after an island is found, Algorithm I_n searches for the next island. Only after a fixed number of islands have been passed does Algorithm I_n search for the goal. Algorithm I_{np} is a slightly more sophisticated version of Algorithm I_n. I_{np} looks not for the next island, but for the next set of islands. This further improves search performance in certain cases.

Algorithm I_n

Algorithm I_n is exactly the same as Algorithm I, but with the addition of the variable $IS_p(n)$ and the user input parameter E. The addition of these two variables allows Algorithm I_n to keep track of how many islands each newly generated node has passed through on a path from the start to that node. The variable $IS_p(n)$ is used to keep track of which islands were passed by each node. The quantity E is a lower bound on the number of islands between the start and the goal that must be passed through on an optimal path. Algorithm I_n searches for a path to the goal through an island if $|IS_p(n)| < E$; otherwise it searches for a path directly to the goal. A newly generated node n will be placed on OPEN1 when $|IS_p(n)| < E$. Only when $|IS_p(n) \geq E$ will n be placed on OPEN2. As testimony to the generality of Algorithm I_n, note that for $E = 0$ Algorithm I_n is equivalent to A* and for $E = 1$, it is equivalent to Algorithm I.

DESCRIPTION OF ALGORITHM I_n.

Inputs:	IS	Island set
	E	Minimum number of islands on path
	s	Start state
	g	Goal state
Output:	Optimal path from s to g, if a path exists	

ALGORITHM I_n.

```
Step 1. Put $s$ into OPEN1. OPEN2, CLOSED1 and CLOSED2 are empty. Set
        $g(s) = 0$, $f(s) = 0$, $IS_p(s) = \emptyset$ and $IS = (i_1, i_2, \ldots, i_k)$.
Step 2. If both OPEN1 and OPEN2 are empty, then exit with failure.
Step 3. Select one node $n$ from OPEN1 or OPEN2 such that $f(n)$ is minimum, resolve
        ties arbitrarily but always in favor of the goal node. Put $n$ in CLOSED1 if it
        was selected from OPEN1, otherwise put $n$ into CLOSED2.
Step 4. If $n$ is the goal, then exit successfully with the solution obtained.
Step 5. Expand node $n$. If there are no successors then go to Step 2. For each
        successor $n_i$ compute
            $g_i = g(n) + c(n, n_i)$.
Step 6. If $n \in IS$ then set $IS_{pi} = IS_p(n) \cup \{n\}$ else set $IS_{pi} = IS_p(n)$.
```

Step 7. If $|\mathrm{IS}_{\mathrm{pi}}| \geq E$ or if n was selected from OPEN2, then do:

(1) If an immediate successor n_i is not in any of OPEN1, OPEN2, CLOSED1, or CLOSED2, then do:

$$\begin{aligned}&g(n_i) = g_i\\&f(n_i) = g_i + h(n_i)\\&\mathrm{IS}_{\mathrm{p}}(n_i) = \mathrm{IS}_{\mathrm{pi}}\\&\text{Put } n_i \text{ in OPEN2.}\end{aligned}$$

(2) If an immediate successor n_i is already in one of OPEN1, OPEN2, CLOSED1, or CLOSED2, and $g(n_i) > g_i$, then do:

$$\begin{aligned}&g(n_i) = g_i\\&f(n_i) = g(n_i) + h(n_i)\\&\mathrm{IS}_{\mathrm{p}}(n_i) = \mathrm{IS}_{\mathrm{pi}}\\&\text{Put } n_i \text{ in OPEN2.}\end{aligned}$$

Step 8. If $|\mathrm{IS}_{\mathrm{pi}}| < E$ and n was selected from OPEN1, then do:

(1) If an immediate successor n_i is not in any of OPEN1, OPEN2, CLOSED1 or CLOSED2, then do:

$$\begin{aligned}&g(n_i) = g_i\\&f(n_i) = g_i + \min_j h(n_i/i_j) \qquad i_j \in \mathrm{IS} - \mathrm{IS}_{\mathrm{pi}}\\&\mathrm{IS}_{\mathrm{p}}(n_i) = \mathrm{IS}_{\mathrm{pi}}\\&\text{Put } n_i \text{ in OPEN1.}\end{aligned}$$

(2) If an immediate successor n_i is in either OPEN1 or CLOSED1 *and* $g(n_i) > g_i$, then do:

$$\begin{aligned}&g(n_i) = g_i\\&f(n_i) = g_i + \min_j h(n_i/i_j) \qquad i_j \in \mathrm{IS}\text{-}\mathrm{IS}_{\mathrm{pi}}\\&\mathrm{IS}_{\mathrm{p}}(n_i) = \mathrm{IS}_{\mathrm{pi}}\\&\text{Put } n_i \text{ in OPEN1.}\end{aligned}$$

(3) If an immediate successor n_i is in either OPEN2 or CLOSED2 and $g(n_i) > g_i$, then do:

$$\begin{aligned}&g(n_i) = g_i\\&f(n_i) = g_i + h(n_i)\\&\mathrm{IS}_{\mathrm{p}}(n_i) = \mathrm{IS}_{\mathrm{pi}}\\&\text{Put } n_i \text{ in OPEN2.}\end{aligned}$$

Step 9. Go to Step 2.

Admissibility of Algorithm $\mathrm{I_n}$

Recall that a search algorithm is *admissible* if it finds an optimal path whenever an optimal path exists. This section will present an informal proof of the admissibility of Algorithm $\mathrm{I_n}$ given the following assumptions:

- At least E islands lie on an optimal path from the start s to the goal g.
- The heuristic estimate is admissible: $0 \leq h \leq h^*$.
- The heuristic obeys the triangle inequality $h(n, n') \leq h(n, m) + h(m, n')$.

A necessary condition for Algorithm $\mathrm{I_n}$ being admissible is that it terminates when a goal is reachable. This has been proven for both finite and infinite graphs in [4].

THEOREM 1. *Algorithm* $\mathrm{I_n}$ *is admissible if there exists a path from* s *to* g.

PROOF. The algorithm can either terminate in Step 2 after both OPEN lists are empty or in Step 4 after finding the goal. The algorithm cannot terminate in Step 2 because at least one node exists on the OPEN lists that is on an optimal path [4]. Therefore, the algorithm must terminate in Step 4 after finding the goal. The path it found to the goal must be an optimal one. To see this, assume the contrary, that the algorithm terminated without finding an optimal path so that $f(g) > f^*(s)$. There must exist a node n' on one of the OPEN lists just before termination with $f(n') \leq f^*(s)$. But this means that the algorithm would have chosen n' over g because of its lower f value. By contradiction, we see the algorithm must terminate by finding an optimal path. ∎

Efficiency of Algorithm $\mathrm{I_n}$

After showing Algorithm $\mathrm{I_n}$ is admissible, it is easily proved that it expands no more nodes than Algorithm I. This means that for any search in which islands can be identified, Algorithm $\mathrm{I_n}$ would yield a search at least as efficient as Algorithm I, which was previously shown to be at least as efficient as A*. We keep the same assumptions as in the previous section.

THEOREM 2. *Algorithm* $\mathrm{I_n}$ *expands no more nodes than Algorithm* I.

PROOF. Nilsson [3] showed that for two admissible algorithms, the one with the greater heuristic estimate will not expand any more nodes than its less informed counterpart. This is the case between Algorithm $\mathrm{I_n}$ and Algorithm I because

$$h_{I_n} = \begin{cases} h(n/i), & \text{if } |\mathrm{IS}_p(n)| < E \text{ or} \\ h(n), & \text{otherwise, and} \end{cases}$$

$$h_I = \begin{cases} h(n/i), & \text{if } |\mathrm{IS}_p(n)| < 1 \text{ or} \\ h(n), & \text{otherwise,} \end{cases}$$

so one can see that $h_{\mathrm{I_n}} \geq H_I$ when $E \geq 1$. ∎

Empirical Results

The rectilinear grid search problem was used to test Algorithm $\mathrm{I_n}$. The test involved partitioning the grid search space into $n+1$ spaces using n obstacles, each with a small hole, as shown in the diagram in Figure 6. This can be viewed as $n+1$ rooms placed end to end with a doorway (island) between each room in a random location. Algorithm $\mathrm{I_n}$ is admissible for this test so long as $E \leq n$. The graph in Figure 6 compares the performance of A*, Algorithm I and Algorithm $\mathrm{I_n}$ with respect to node expansions. For this case, E was set to 4 for Algorithm $\mathrm{I_n}$, since there are four islands on the optimal path. Each point in Figure 6 is an average of 10 random obstacle placements. As you can see, Algorithm I offers little improvement over A* as the size of the problem increases. In fact, Algorithm I actually needed more time to solve these problems than A* because the extra time required to evaluate the island heuristic did not make up for the very small reduction in node expansions. Algorithm $\mathrm{I_n}$, however, does yield a reduction in node expansions over Algorithm I and A*. Timing results for Algorithm $\mathrm{I_n}$ are presented later. As we shall see in the next section, island search can be improved even further.

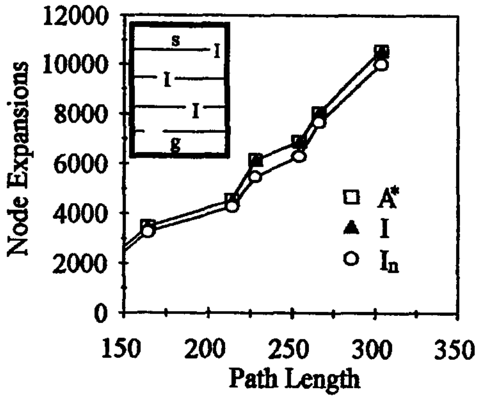


Figure 6. Comparison of A*, Algorithm I, and Algorithm I_n.

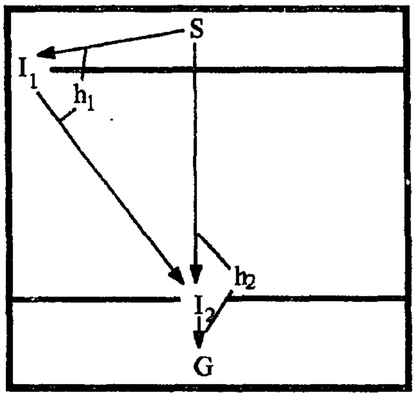


Figure 7. Search space with pronounced interference problem.

Island Interference

One problem with island search becomes evident when there is a large number of islands, but relatively few optimal paths through the islands. What can happen in this case is that the heuristic estimate $h(n/i)$ guides the search toward the wrong island. A number of test results revealed that in these cases island search is only marginally better than A*. Figure 7 is an extreme example of how an island heuristic can mislead a search. The heuristic estimate through island $I_2(h_2)$ is smaller than the estimate through $I_1(h_1)$ so $\min_j h(n_i/i_j) = h_2$ for a large portion of the search. This will lead the search towards I_2 when it really should initially be heading towards I_1. Since I_2 is so close to the goal, Algorithm I's performance will not be much better than A*. Similar situations also occur for Algorithm I_n

The quantity $\min_j h(n_i/i_j)$ picks the island which yields the lowest heuristic estimate. This guarantees admissibility, but does not cut down the size of the search space for cases similar to that depicted in Figure 7. Although Algorithm I_n does generalize to multiple islands, it still suffers from the same interference problem as Algorithm I. One solution is to use $\min_j h(n/t_j)$ in place of $\min_j h(n_i/i_j)$. The function that computes $\min_j h(n/t_j)$ is called the permuted heuristic and instead of finding the shortest path through one island in $\text{IS-IS}_p(n)$, it finds the shortest path through a permutation of $E - |\text{IS}_p(n)|$ different islands in $\text{IS-IS}_p(n)$. An algorithm for computing the permuted heuristic can be defined recursively as follows.

Permuted Heuristic.

```
function $hx(n, \text{IS}_p)$
    if ($|\text{IS}_p| == E$)
        $retval = h(n, g)$
        else
                $retval = \infty$
                for every island $i$ in IS but not in IS$p$ do
                        $retval = \min(retval, h(n,i) + hx(i, \mathrm{IS_p} \cup \{i\}))$
                end for
        end if
        $hx = retval$
end
```

The hx heuristic is admissible if and only if at least E islands are on an optimal path from the start to the goal. It is also assumed that a heuristic $h(n,m)$ is available to estimate the distance between any node and any island (or the goal). The hx heuristic can be added to Algorithm $\mathrm{I_n}$ to produce Algorithm $\mathrm{I_{np}}$ simply by replacing all occurrences of $\min_j h(n_i/i_j)$ with calls to $hx(n_i, \mathrm{IS_{pi}})$. Algorithm $\mathrm{I_{np}}$, like Algorithm $\mathrm{I_n}$, is equivalent to A* for $E = 0$ and Algorithm I for $E = 1$.

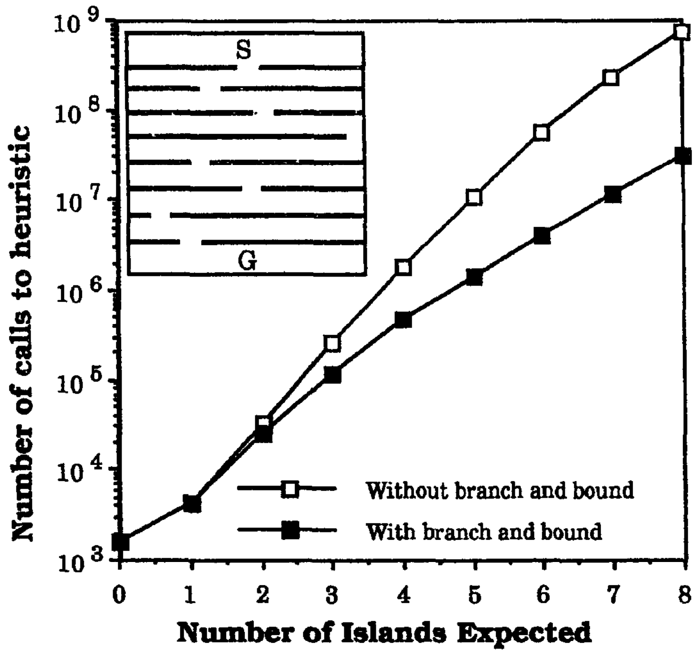


Figure 8. Comparison of algorithm complexity with and without branch and bound technique.

The permuted heuristic is, in general, intractable. This can be proved by a simple reduction to the traveling salesman problem. Note, however, that the heuristic is only intractable when the number of islands is a function of the size of the search space. Through the use of branch and bound techniques, the computational complexity can be reduced as seen in Figure 8. The branch and bound technique was applied to the permuted heuristic by keeping a global minimum and stopping a recursive call whenever hx exceeded this minimum. The use of branch and bound together with a relatively small island set can make the permuted heuristic more usable.

Admissibility of Algorithm $\mathrm{I_{np}}$

Algorithm $\mathrm{I_{np}}$ must be shown to be admissible before further claims can be made about it. The theorems showing that Algorithm $\mathrm{I_{np}}$ is admissible are very similar to those presented for Algorithm $\mathrm{I_n}$. The theorems in this section require the same assumptions made for Algorithm $\mathrm{I_n}$, exceptions noted. As with Algorithm $\mathrm{I_n}$, Algorithm $\mathrm{I_{np}}$ also terminates when a goal is reachable, see [4].

THEOREM 3. *Algorithm $\mathrm{I_{np}}$ is admissible if there exists a path from s to g.*

PROOF. This proof is identical to the proof of Theorem 1.

Efficiency of Algorithm $\mathrm{I_{np}}$

It has already been shown that Algorithm $\mathrm{I_n}$ is at least as efficient as Algorithm I which is at least as efficient as A*. We can show that Algorithm $\mathrm{I_{np}}$ is at least as efficient as Algorithm $\mathrm{I_n}$. Showing that Algorithm $\mathrm{I_{np}}$ is at least as efficient as Algorithm $\mathrm{I_n}$ will prove (by transitivity) it is at least as efficient as any of the three island algorithms. Of course, all of this is under the assumptions made earlier about there being at least E islands from the island set IS on an optimal path.

THEOREM 4. *Algorithm $\mathrm{I_{np}}$ expands no more nodes than Algorithm I_n.*

PROOF. Both algorithms are admissible, so the algorithm with the lower heuristic estimate will expand at least as many nodes as its more informed counterpart.

$$h_{\mathrm{I_{np}}} = \begin{cases} h(n \mid t), & \text{if } |\mathrm{IS_p}(n) < E| \ (t \text{ is a set), or} \\ h(n), & \text{otherwise, and} \end{cases}$$
$$h_{\mathrm{I_n}} = \begin{cases} h(n \mid i), & \text{if } |\mathrm{IS_p}(n)| < E \text{ or} \\ h(n), & \text{otherwise, and} \end{cases}$$

one can see that $h_{\mathrm{I_{np}}} \geq h_{\mathrm{I_n}}$ because

$$h(n/t) = h(n,i_1) + h(i_1,i_2) + \cdots + h(i^k,g)$$

is greater than

$$h(n/i) = h(n,i) + h(i,g)$$

when

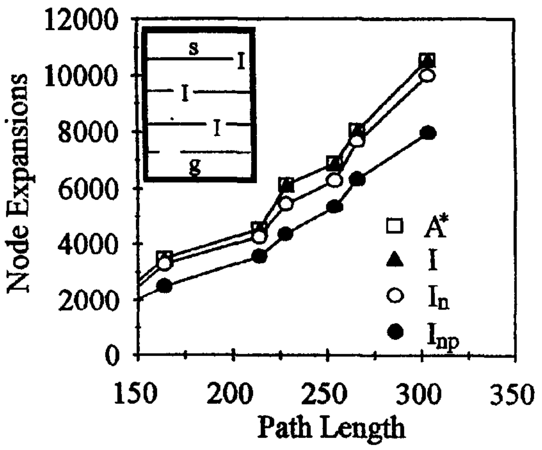
$$h(n,m) \leq h(n,n') + h(n',m).$$

∎

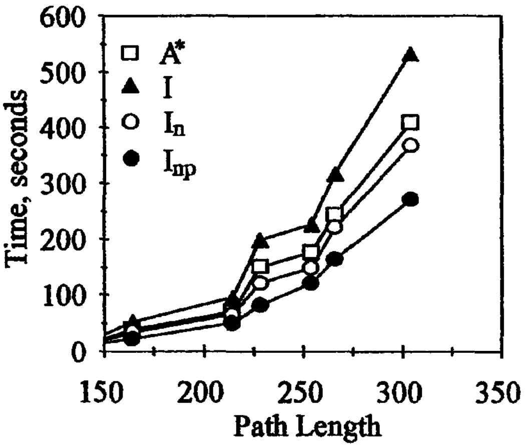
Test Results for Algorithm $\mathrm{I_{np}}$

Algorithm $\mathrm{I_{np}}$ was also tested using the rectilinear grid search problem. The particular test used in Figures 9a and 9b is similar to the test used for Algorithm $\mathrm{I_n}$ in Figure 6. Figure 9a shows the number of node expansions versus the problem size and Figure 9b shows the elapsed CPU time versus the problem size. The improved performance of Algorithm $\mathrm{I_{np}}$ is due to the elimination of the interference problem.

In addition to testing search performance as the problem size grows, it is also interesting to test the search efficiency versus the parameter E. Remembering that for $E = 0$, both algorithms are equivalent to A* and that for $E = 1$ both algorithms are equivalent to Algorithm I, we see from Figure 10a that Algorithm I provides no benefit over A* for this case involving multiple islands. Figure 10b shows that multiple level island search needs more CPU time in some cases because the extra time needed to compute the island heuristic is not made up by fewer node expansions. The improvement using Algorithm $\mathrm{I_n}$ as compared to Algorithm $\mathrm{I_{np}}$ is slight due to the interference problem. The effect of the interference problem can be seen by comparing the two curves in Figure 10a, since the permuted heuristic effectively eliminates this problem. The results show that the number of node expansions drop as the parameter E increases. Past a certain point, however, hx is no longer admissible and the number of node expansions increases dramatically. Only results from admissible heuristics are shown in Figures 10a and 10b.

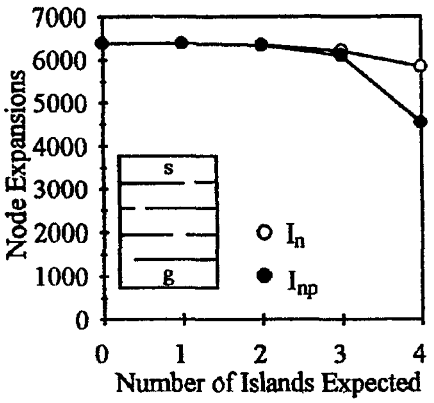


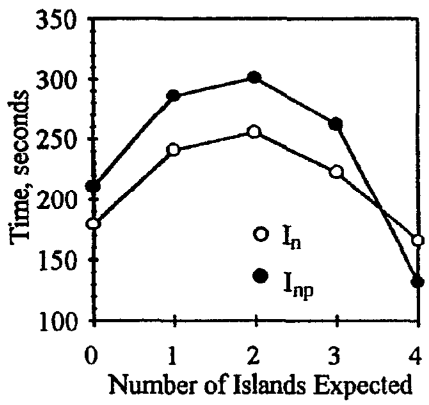
(a) Node expansion comparison between four different search methods.



(b) Elapsed CPU time comparison between four different search methods.

Figure 9.





(a) Node expansion comparison for multiple level island search.

(b) Elapsed CPU time comparison for multiple level island search

Figure 10.

Another benefit of reducing the number of node expansions is that the amount of memory needed to solve the problem is also reduced. Because fewer nodes are being expanded, fewer nodes need to be stored in the OPEN and CLOSED lists. Figure 11 shows how the amount of memory needed varies as a function of E, the lower bound on the number of islands. As E approaches the actual number of islands, the amount of memory used by Algorithm I_{np} drops substantially. Because Algorithm I_n offers only a small improvement in the number of node expansions, the amount of memory it uses does not improve for this case.

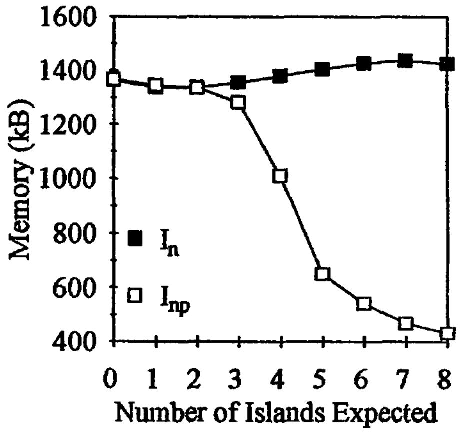


Figure 11. Memory usage in multiple level island search.

4. ISLAND SEARCH FOR ROUTE PLANNING

The test results in the previous sections show that when island information is available, it may be used to improve search performance. It may be questioned whether there actually exist any "real" problems for which island information is available. In this section, we describe an intelligent vehicle routing problem for which island search is a viable candidate.

Each year the loss of U.S. productivity due to traffic congestion is estimated to be $100 billion [5]. Unless significant improvements are made to our highway system, the Federal Highway Administration estimates that congestion delays will increase by 400 percent. In an attempt to solve congestion and other related problems associated with our highway system, the development of intelligent vehicle highway system (IVHS) technology is receiving high priority by both the public and private sector.

IVHS covers four broad interrelated areas, advanced traffic management systems, (ATMS), commercial vehicle operations (CVO), advanced vehicle control systems (AVCS), and advanced traveler information (ATI) systems [6].

Currently there are several ATI system demonstrations underway including ADVANCE [7], TRAVTEK [8], and SOCRATES [9]. The overriding goal of these and other ATI systems is to provide the motorist with useful travel information. With this in mind, a primary function of most ATI systems involves having the ability to interactively plan optimal routes.

The majority of ATI systems currently under development consist of equipping vehicles with computers that are able to exchange traffic information via radio with a centralized computer. Given this type of architecture, the route planning ability of ATI systems can be facilitated using centralized computing resources, although most ATI systems currently in development use in-vehicle computational resources [7–9]. The primary advantage of this type of approach is fault tolerance; vehicles can continue to plan routes even in the absence of a centralized computing center.

However, there are significant challenges associated with planning routes using in-vehicle electronics. These challenges are a result of trying to keep the cost of the in-vehicle electronics to a reasonable (marketable) level. Achieving this is difficult because the in-vehicle electronics must meet automotive standards (temperature, vibration, etc.) and, therefore, have poor price/performance ratios. Thus, the route planning algorithm must be both space and time efficient.

The island search Algorithms I_n and I_{np} can be used to significantly reduce the number of node expansions required to find routes, and hence, are good candidates for use in ATI systems. The reduction in node expansions is possible because most routes between two regions in a road network end up passing through a small number of intermediate nodes. These nodes correspond to an island set and can be determined through a learning process as described below.

In a practical implementation, it would be necessary to determine the set of islands associated with each pair of regions in the road network. For the purposes of testing, however, just two regions of a 60 mile radius Chicago area map were selected for study. Each region corresponded to approximately 1% of the map. The island set for the two regions was determined by planning every possible route between the two regions during simulated rush hour conditions [8]. Fifty-nine nodes were identified which were present in 95% of the routes. Six of these nodes were selected randomly for use as islands.

The island search algorithms I_n and I_{np} were tested by planning routes between 100 randomly selected start/destination pairs located in the two regions. The location of the islands was determined using simulated rush-hour conditions and the algorithms were then asked to use these islands to find routes in simulated light traffic conditions. This provided a worst case scenario for testing and was done because in practice the island set for a pair of regions would be precomputed and would be used under varying traffic conditions. Figure 12 shows the results of the tests. Each point in Figure 12 represents the average amount of time needed to plan a route over the 100 test cases. The time data is based on a simulation of a slow external storage device which requires 800 milliseconds to access each node (i.e., an automotive quality CD-ROM). Even though the island search heuristic $h(n/i)$ required more time to compute than the normal A* heuristic $h(n)$, this time is more than made up for by the reduction of node expansions. In fact, Figure 12 shows that Algorithm I_{np} was over 200% faster that A*.

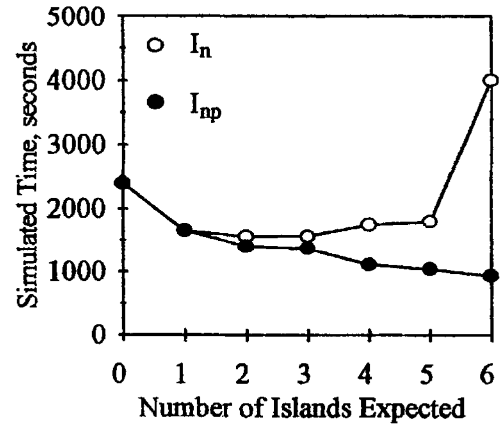


Figure 12. Simulated amount of time required to plan routes using in-vehicle electronics.

Figure 12 also shows an increase in simulated time for Algorithm I_n when $E = 6$. This is due to a combination of the interference effect and the fact that there are actually only five islands on the optimal paths for the 100 test cases. Algorithm I_{np} was not effected as much by the errant island due to its use of the permuted heuristic hx. The simulated travel times of the routes planned by the island search algorithms were optimal in all cases except when $E = 6$, in which case routes planned by Algorithm I_n were on average 1% longer and routes planned by Algorithm I_{np} were on average 6% longer than optimal.

5. EXTENSIONS TO ISLAND SEARCH

There are a number of extensions that can be made to island search. These extensions allow the algorithms to be used on a wider variety of problems. For instance, there are many different categories of admissibility [10]. These can be applied to island search just as easily as they can be applied to A*. Basically, these *semi-admissible* searches cannot guarantee an optimal solution, but instead a solution *close* to optimal. Another example of an extension that might be useful is the idea of iterative depth-first search. Iterative depth-first search maintains the guarantee of optimality, but has an advantage in terms of the amount of storage space used during the search when compared to regular best-first search since only the current path is stored in memory rather than a collection of all explored nodes. Iterative depth-first search can also be shown to be optimal with respect to time [11].

Iterative Depth-First Island Search

An iterative depth-first search uses only linear storage space while still being able to guarantee an optimal solution path. The algorithm for an iterative depth-first version of island search (IDIS) would be as follows:

IDIS ALGORITHM.

```
Step 1. Set $next\_cutoff = h(s)$, $g(s) = f(s) = 0$. If $s$ is in IS, then set $i(s) =$ TRUE else
        set $i(s) =$ FALSE
Step 2. Place $s$ on the stack. Set $cutoff = next\_cutoff$ and $next\_cutoff = \infty$.
Step 3. If the stack is empty, go to step 2
Step 4. Pop a node $n$ off the stack. If $n$ is the goal, then exit successfully with the
        solution obtained.
Step 5. For each immediate successor $n_i$ of $n$ do the following:
        (1) Set $g(n_i) = g(n) + c(n, n_i)$
        (2) If $i(n)$ is TRUE or if $n$ is an island, then set $i(n_i) =$ TRUE and
            $f(n_i) = g(n_i) + h(n_i)$, otherwise set $i(n_i) =$FALSE and
            $f(n_i) = g(n_i) + \min_j h(n_i/i_j)$
        (3) If $f(n_i) \leq cutoff$, then push $n_i$ onto the stack, else set
            $next\_cutoff = \min(next\_cutoff, f(n_i))$
Step 6. Go to step 3
```

The iterative depth-first island search algorithm is basically the same as IDA* [11]. By adding a Boolean variable to determine when an island has been encountered, the algorithm can decide when to use the island heuristic and when to use the normal heuristic. Note that the Boolean variable can be kept on the stack along with the other state information. Since the heuristic used in IDIS follows the guidelines presented in [11] (admissible and monotone), the IDIS algorithm will also be optimal in terms of time and memory usage. Preliminary tests results comparing IDIS to IDA* also confirm that IDIS will have fewer node expansions than IDA* when possible subgoals can be identified.

An ε-admissible Multiple Level Island Search

The type of semi-admissible search that will be discussed here is called ε-admissible. A search algorithm is said to be ε-admissible if it can find a solution with a cost of not more than $f^*(s) + \varepsilon$. Compare this to the normal requirement that the solution have a cost of not more than $f^*(s)$. With an ε-admissible algorithm, we are only guaranteed that a solution is close to optimal. This is the price we pay for a potentially faster search. A simple modification to Algorithm I produces an ε-admissible algorithm. Chakrabarti *et al.* [1] made this extension to Algorithm I to create Algorithm I$'$. Algorithm I$'$ requires an additional input d which is used to guarantee that the cost of the solution will not be more than $d + f^*(s)$.

Algorithm $\mathrm{I_n}$ and $\mathrm{I_{np}}$ can become ε-admissible in a manner similar to Algorithm I. The only change to the algorithms occurs in steps 8 (1) and (2). Instead of always using $\min_j h(n_i/i_j)$ for the heuristic, we can use $h(n_i)$ whenever $|\min_j h(n_i/i_j) - h(n_i)| > d$. Basically, this change insures that the heuristic does not become too large. By comparing the possibly inadmissible $\min_j h(n_i/i_j)$ to the admissible $h(n_i)$, we can guarantee the search finds a path within the error bound. Note that for Algorithm $\mathrm{I'_{np}}$, we use the permuted heuristic in place of $\min_j(h(n_i/i_j)$.

With this change to the algorithms, it can be shown that we now have an ε-admissible algorithm when ε is equal to d. This proof is very similar to the proof of Theorem 1.

As mentioned in Section 1, the ε-admissible versions of the island search algorithms are useful in cases where information about the island set IS or the number of islands on the optimal path E is unreliable. By using an ε-admissible island search algorithm, one can guarantee a bounded solution if either IS or E are in error.

6. SUMMARY

Experimental results have been presented showing that island search techniques can be used to reduce the number of node expansions. The single level island search Algorithm I [1] was compared with the A* algorithm. Algorithm I was shown to offer some improvement. Most search spaces which contain identifiable islands will often have solution paths through multiple islands. With this in mind, two multiple level island search algorithms, $\mathrm{I_n}$ and $\mathrm{I_{np}}$, were presented which can be used in such situations. Test results show that both of these algorithms outperform Algorithm I, with Algorithm $\mathrm{I_{np}}$ showing the greatest improvement. Furthermore, Algorithm $\mathrm{I_{np}}$ has been shown to be admissible and to expand no more nodes than Algorithm $\mathrm{I_n}$.

The use of island search for route planning was also discussed. By taking advantage of the fact that many paths in a road network pass through a common set of nodes, the island Algorithms $\mathrm{I_n}$ and $\mathrm{I_{np}}$ were shown to reduce the number of node expansions without sacrificing route quality. These results could have important ramifications for advanced traveler information systems which must rely on in-vehicle route planners.

Further research is currently being done into the use of ordered multiple island sets. By partitioning the islands into ordered sets, we can reduce the cost of computing the permuted heuristic for $\mathrm{I_{np}}$ while improving the heuristic estimate. Another part of our work includes developing a parallel island search algorithm where each island serves as a search origination point in a bidirectional search scheme [12].

As mentioned previously, the island search concept can also be applied to other types of heuristic search. This paper showed how the island search concept is applied to A* and IDA*. Island search versions of MREC [13] and RTA* [14] are currently under development.

REFERENCES

1. P.P. Chakrabarti, S. Ghose and S.C. Desarkar, Heuristic search through islands, *Artificial Intelligence* **29**, 339–348, (1986).
2. P. Hart, R. Duda and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. on SSC* 4, 100–107 (1968).
3. N.J. Nilsson, *Principles of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, (1980).
4. J.F. Dillenburg, Theoretical and experimental results of heuristic island search studies, Master's Thesis, Univ. of Illinois, Chicago, IL, (1991).
5. M. Ben-Akiva, D. Bernstein, A. Hotz, H. Koutsopoulos and J. Sussman, The case for smart highways, *Technology Review* (July), 39–47 (1992).
6. R.K. Jurgen, Smart cars and highways go global, *IEEE Spectrum* (May), 26–36 (1991).
7. A. Kirson, B. Smith, D. Boyce, P. Nelson, J. Hicks, A. Sen, J. Schofer, F. Koppelman and C. Bhat, The Evolution of ADVANCE, In *3rd International Conference on Vehicle Navigation and Information Systems*, September 1992, pp. 516–522.
8. J.H. Rillings and J.W. Lewis, TravTek, *Vehicle Navigation & Information Systems Conference Proceedings*, pp. 729–738, (1991).
9. I. Catling and F.O. de Beek, SOCRATES: System of cellular radio for traffic efficiency and safety, In *Vehicle Navigation & Information Systems Conference Proceedings*, pp. 147–150, (1991).
10. J. Pearl, *Heuristics*, Addison-Wesley, Reading, MA, (1984).
11. R.E. Korf, Depth-first iterative deepening: An optimal admissible tree search, *Artificial Intelligence* **27**, 97–109 (1985).
12. P.C. Nelson, Heuristic search using islands, *Proc. Hypercubes, Concurrent Computers and Applications* **4** (1989).

13. Anup K. Sen and A. Bagchi, Fast recursive formulations for best-first search that allow controlled use of memory, In *Proc. of the Eleventh International Joint Conference on Artificial Intelligence 1*, pp. 297–302, (1989).
14. R.E. Korf, Real-time heuristic search, *Artificial Intelligence* **42**, 189–211 (1990).