Admissibility Proofs for the LCS* Algorithm

Marcelo O. Johann[1], Andrew Caldwell[2], Ricardo A. L. Reis[1], and Andrew B. Kahng[2]

[1] Federal University of Rio Grande do Sul (UFRGS), Brazil
{johann,reis}@inf.ufrgs.br
[2] University of California at Los Angeles (UCLA), USA
{caldwell,abk}@cs.ucla.edu

Abstract. Bidirectional search and heuristic search are techniques that improve the performance of a shortest path graph search. Despite many attempts to use both at the same time, this combination had been leading to worse results in average, compared to the classic unidirectional heuristic A* algorithm. In [4], a new graph search algorithm was developed, LCS*, that is the first to combine these techniques effectively. LCS* is a generic and simultaneous bidirectional heuristic algorithm which is faster than A* in most domains. This work present formal proofs of the completeness and admissibility of LCS*.

1 Introduction

Given a degree-bounded (or, locally finite) directed graph with edge costs, the shortest path problem is to find, for a given pair of nodes s and t, a path from s to t with total cost less than or equal to the cost of any other path from s to t. An algorithm that always finds this path is *admissible.*

The first admissible search algorithms were Moore's (1957) and Dijkstra's (1959), which traverse the graph in a *breadth-first* manner starting from the source until the target is reached. Search algorithms work as follows. Starting with the node s, they explore some part of the graph, known as the *search tree*, by repetitive application of the *successor operator*. Each time a successor operator is applied to a node, we say that the algorithm has *expanded* that node. We say that nodes returned by the successor operator are *generated* by the algorithm. An already expanded node is called a *closed* node, and is stored in the *closed list.* Nodes generated but not yet expanded are called *open* nodes, and are stored in the *open list.* A node in the search tree is either open or closed at a given moment. If the algorithm cannot identify that a given node was already generated (or expanded), we say that the *search tree* is part of the *search space*, which differs from the *state space* because the same state can appear more than once.

In [3] Dijkstra's algorithm is extended into a more efficient algorithm called A* by including an estimate of the remaining path cost to the target. A* selects nodes for expansion in order of increasing $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to go from s to n and $h(n)$ is an estimate of the cost to go from n to t. This process searches nodes closer to the target first, and is called *heuristic search* or *best-first search.* A* is admissible if and only if the remaining path costs are *underestimated.* Although A* may be outperformed by other algorithms in some specific graph problems, it is proven to be optimal among all other unidirectional algorithms that are not *more informed* than A* and if no *ties* are allowed [1]. A *tie* occurs when two nodes n_1 and n_2 in the search's open list have the same evaluation result, e.g.: $f(n_1) = f(n_2)$. We say that there is a *critical tie* between n_1 and n_2 when $f(n_1) = f(n_2) = C^*$, where C^* is the cost of the optimal path.

A bidirectional search can be viewed as two simultaneous searches starting from source and target nodes respectively. A new path is found when a given node is recognized to be at both search trees at the same time. This node is called a *meeting node.* The *terminating condition* of a bidirectional algorithm is not as simple as in unidirectional ones. The first complete path from s to t found by a bidirectional algorithm may not be the optimal one, and the algorithm needs to continue expanding nodes until all nodes n in at least one open list have $f(n) \geq min[f(m)]$ $\forall$ meeting node m. This behavior leads to searches *overlapping*, which is the generation and/or expansion, by both searches, of the same nodes in the state space.

In [8] it is introduced BS*, a bidirectional heuristic algorithm that avoids search overlapping with *nipping, pruning, trimming* and *screening* operations. However, this wasn't sufficient to make it better than A*. There had been great difficulty in achieving improvements with heuristic bidirectional search. The main problems reported were the complexity of keeping the needed information and making both searches meet in the middle rather than each search finding essentially disjoint equivalent (optimal) $s - t$ paths (*missing fronts* problem).

Recently, the potential gain of bidirectional search has appeared again, as some previous counter-arguments were shown to be weak [7] [5]. Perimeter search [9] presents better results under some conditions. Another improvement appears in the work of [6]. As in perimeter search, searches are not simultaneous, but one following the other. This *non-traditional* bidirectional search allows dynamic estimation to be easely implemented as single added values. Statistical data show sensible reduc-

tion in number of expanded nodes and runing time. The limitation is the fact that the algorithm must know when to change direction in order to optimize its search, and if made wrong, this decision may nullify the gains or even make the search more expensive.

LCS* is a new graph search algorithm ([4]) which also uses dynamic estimation to improve the prunning power. LCS* is the first to combine both bidirectional and heuristic techniques effectively using simultaneous searches in a simple way. In order to prove the completeness and admissibility of LCS*, the next sections summarizes a set of notations that are used in unidirectional and bidirectional heuristic search for theoretical values and actual data mainained by the algorithms. A set of properties associated with heuristic values is also reviewed. Then, LCS* is presented along with its proofs.

2 Definitions

2.1 Graph definition

G	$= (V, E)$, directed degree-bounded graph
s, t	source and target nodes, with $s, t \in V$
$c(n_1, n_2)$	cost of arc $(n_1, n_2) \in E$ for nodes $n_1, n_2 \in V$
$S_s(n)$	immediate successors of n, $\{x \| x \in V, (n, x) \in E\}$
$S_t(n)$	immediate predecessors of n, $\{x \| x \in V, (x, n) \in E\}$

2.2 Optimal values

P^*_{n-m}	optimal path from $n \in V$ to $m \in V$
$k^*(n, m)$	cost of the optimal P^*_{n-m} path
$g^*(n)$	$= k^*(s, n)$, cost of the optimal $s - n$ path
$h^*(n)$	$= k^*(n, t)$, cost of the optimal $n - t$ path
$f^*(n)$	$= g^*(n) + h^*(n)$
C^*	$= k^*(s, t)$, cost of the optimal $s - t$ path

2.3 Estimates used by heuristic search

$g(n)$	estimate of $g^*(n)$
$h(n)$	estimate of $h^*(n)$
$f(n)$	$= g(n) + h(n)$, estimate of $f^*(n)$
c_{min}	$0 < c_{min} \leq c(n_1, n_2) \ \forall \ n_1, n_2 \in V \mid (n_1, n_2) \in E$
$k(n_1, n_2)$	consistent estimate of $k^*(n_1, n_2)$;

2.4 Properties of the estimator function h

h is *admissible* iff $h(n) \leq h^*(n)\ \forall\ n \in V$ (underestimation)
h_1 is *no more informed than* h_2 iff $h_1(n) \leq h_2(n) \leq h^*(n)\ \forall\ n \in V$
h is *consistent* iff $h(n_1) \leq k^*(n_1, n_2) + h(n_2)\ \forall\ n_1, n_2 \in V$

3 A New Bidirectional Algorithm LCS^*

In [4] a new algorithm is introduced, called LCS*: Lower bound Cooperative Search. As any other bidirectional algorithm, LCS* requires the ability to identify the same node in the *state space*, and requires also *consistent* estimators. It is shown that LCS* outperforms A* in average in most application domains, both considering the number of nodes expanded and actual running time (Fig. 1), while keeping the same space complexity. Improvements in LCS^* are due to two concepts: cooperation and visibility.

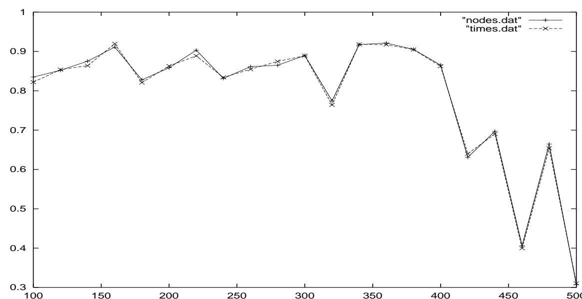

Fig. 1. LCS*/A* ratio in number of nodes and running times for 2D grids

Cooperation is found by observing that the function $g()$ of one search corresponds to the function $h()$ of the other, and vice-versa. Then, it is possible not only to reduce the amount of information stored, but also to improve the estimates in one search based on the other's paths. This is known as dynamic estimation, and was already exploited in [6]. To use dynamic estimation, one have to make new estimation functions that account for information coming from the opposite search. But in order to keep the new estimation values consistent, it is not possible to change the ordering imposed by them. Therefore, in bidirectional algorithms, the new estimation is actually the same plain estimation, but additional values are used for the terminating conditions. This becomes clear looking at

the algorithm code, and also observing the fact that the prunning power of an algorithm is the factor that makes it finish sooner or not.

The novelty of our algorithm is to show that such dynamic estimation can be used in simultaneous, or traditional algorithms. As the boundaries of both searches overlap each other, we need to prevent their temporary data to be used by the opposite search. The concept of *visibility* is to hide temporary (estimated) data inside each search, while sharing the values that are already proven to be optimal between both search fronts, in a public space. This is accomplished by storing g and h functions in *references*, but not in the actual nodes of the state space, as in previous works. Each search has its own private open list to store references generated from it, and they are not visible to the opposite search. Once a node n is expanded, the algorithm knows its $g^*(n)$ value (this is guaranteed by admissible and consistent estimators), and this value, along with a pointer, is stored in the public state space structure. LCS* is presented on figure 2. The algorithm has the same structure and steps of BS* [8].

The information maintained by the algorithm during its execution is the following:

$Closed_s$Ç$losed_t$	set of nodes reached from s and t already expanded
$Open_s$	set of references r to nodes $n(r)$, $n(r) \in S_s(x)$
	for some $x \mid x \in Closed_s$ and $n(r) \notin Closed_s$
$Open_t$	set of references r to nodes $n(r)$, $n(r) \in S_t(x)$
	for some $x \mid x \in Closed_t$ and $n(r) \notin Closed_t$
$g(n)$	optimal path cost from s to n if $n \in Closed_s$
	or from t to n if $n \in Closed_t$
$p(n)$	parent of n from where it was expanded with
	lowest cost, $n \in Closed_s \oplus Closed_t$
$g_s(r)$	estimate of $g_s^*(n(r))$,
$g_t(r)$	estimate of $g_t^*(n(r))$,
	$n(r)$ is the node pointed to by r
$p(r)$	parent node of $n(r)$ that generated it
$d(r)$	number of arcs in the path from s to $n(r)$ if
	$r \in Open_s$ or from t to $n(r)$ if $r \in Open_t$
L_{min}	$\geq C^*$, cost of the best s-t path found so far
Ω_s	minimum add cost to reach s through $Closed_s$
Ω_t	minimum add cost to reach t through $Closed_t$
Py_s, Py_t	admissible estimation of $P^*(p)$, $p \in Open_s$ or $\in Open_s$
$MeetN$	meeting node $\in Closed_s \oplus Closed_t$
Mp_s, Mp_t	$MeetN$ pointer toward source and target

LCS* Shortest Path Algorithm

Input: $G = (V, E)$

Output: P^*_{s-t}

0. $L_{min} \leftarrow \infty; MeetN \leftarrow none$; $\Omega_s \leftarrow 0$; $\Omega_t \leftarrow 0$; $Py_s \leftarrow 0$; $Py_t \leftarrow 0$;
1. put new r_s in $Open_s$: $n(r_s) = s, d(r_s) = g_s(r_s) = 0, g_t(r_s) = dc(s,t)$;
   put new r_t in $Open_t$: $n(r_t) = t, d(r_s) = g_t(r_t) = 0, g_s(r_t) = dc(s,t)$;
2. if necessary, update Ω_s, Ω_t, Py_s, Py_t; // dynamic estimation
   if expand_from_source // direction choosing criteria
   then $\varsigma = s, \tau = t$ else $\varsigma = t, \tau = s$; endif
3. select $r \in Open_\varsigma \mid f(r) \leq f(x)\ \forall x \in Open_\varsigma; n \leftarrow n(r)$; // selection

3.1 if $Open_\varsigma$ or $Open_\tau$ is empty // stop condition 1
3.2 or $f(r) \geq L_{min} - \Omega_\tau$ // stop condition 2
  then goto step 5 (stop); endif
3.25 if $g(r) - k(\varsigma, n(r)) > L_{min} - Py_\tau$ // stop condition 2
  then return to step 2; endif // screening 2
3.3 if $n = \tau$ [and $g_\varsigma(r) < L_{min}$] // target meeting node
  then $L_{min} \leftarrow g_\varsigma(r)$;
   $MeetN \leftarrow n$; $Mp_\varsigma \leftarrow p(r); Mp_\tau \leftarrow none$;
   goto step 2; endif

4. // check/expand node $n = n(r)$ (all steps 4)

4.1 if $n \in Closed_\varsigma$ [and $g(n) \leq g_\varsigma(r)$]
  then return to step 2
4.2 if $n \in Closed_\tau$ // nipping/pruning
  then
   if $(g_\varsigma(r) + g(n) < L_{min})$
   then $L_{min} \leftarrow g_\varsigma(r) + g(n)$;
    $MeetN \leftarrow n$; $Mp_\varsigma \leftarrow p(r); Mp_\tau \leftarrow p(n)$;
    goto step 2; endif; goto step 2; // trimming
4.3 else put n in $Closed_\varsigma$; $g(n) \leftarrow g_\varsigma(r); p(n) \leftarrow p(r)$; endif
4.4 for each node m in $S_\varsigma(n)$ do: // expansion
   if $m \notin Closed_\varsigma$
   and $g(n) + c(n,m) + k(m,\tau) < L_{min} - \Omega_\tau$ // screening
   then
    put a new reference r_m in $Open_\varsigma$
     with: $n(r_m) = m, p(r_m) = n, d(r_m) = d(r) + 1$,
    $g_\varsigma(r_m) = g(n) + c(n,m), g_\tau(r_m) = k(m,\tau)$;
4.5 // earlier termination
    if $m \in Closed_\tau$ and $g_\varsigma(r_m) + g(m) < L_{min}$
    then
     $L_{min} \leftarrow g_\varsigma(r_m) + g(m)$;
     $MeetN \leftarrow m$; $Mp_\varsigma \leftarrow n; Mp_\tau \leftarrow p(m)$;
    endif endif endfor
4.6 remove and delete r from $Open_\varsigma$; goto step 2.

5. if $L_{min} < \infty$
   then the shortest path is traced from $MeetN$ with Mp_s, Mp_t
   and $p(n)$ pointers until reaching s and t, and has cost L_{min};
   else no path exists;

Fig. 2. Basic LCS^* Algorithm.

3.1 LCS* is Admissible

Assume initially that $Py_s = Py_t = \Omega_s = \Omega_t = 0$. Under these circumstances, it is possible to prove that LCS^* is complete and admissible. The proof uses optimal values in the symmetric notation, and specific data stored by the algorithm, as just defined above. The estimated evaluation function $f()$ is defined only on references, as $f(r) = g_s(r) + g_t(r)$, while the optimal function $f^*()$ is defined only on actual nodes, as $f^*(n) = g_s^*(n) + g_t^*(n)$. This helps understanding the differences between instance properties (optimal values) and information stored by the algorithm. We shall use indexes ς to designate "this search" and τ to designate the "the opposite search". So, $\varsigma = s$ and $\tau = t$ in the search beginning at s, and $\varsigma = t$ and $\tau = s$ in the search beginning at t.

As in [8], it is not possible to prove that *every reference r to a node n expanded* by LCS* has $f(r) = f^*(n(r))$ because of *nipping/pruning* operations. The proof for admissibility is conducted as follows: There are always references to nodes *in the optimal path* in both search's open lists. Once these nodes are selected for expansion, we know their $f^*()$ values. Meeting nodes are found and the algorithm terminates with the optimal solution; In the next section we show how to use values greater than zero for Ω and Py while maintaining these properties.

lemma 0 Any time during execution $C^* \leq L_{min}$
Proof: At the beginning (step 0) $L_{min} = \infty$. L_{min} is updated at the meeting nodes by $g_\varsigma(r) + g(n)$ where $n = n(r)$, $r \in Open_\varsigma$ and $n \in Closed_\tau$. Now $g_\varsigma^*(n) \leq g_\varsigma(r)$ and $g_\tau^*(n) \leq g(n)$. Therefore, $f^*(n) \leq g_\varsigma(r) + g(n)$. Since $C^* \leq f^*(n)\ \forall n \in V$, $C^* \leq g_\varsigma(r) + g(n)$ for all meeting nodes n.

lemma 1 L_{min} is non-increasing. Proof: conditions at steps 3.3, 4.2 and 4.5 of the algorithm. Thus, if at any given moment there is some value $x \geq L_{min}$, the same value x will remain $\geq L_{min}$ from this moment on, until the algorithm terminates;

lemma 2 No reference r with $f(r) \geq L_{min}$ is ever expanded. Proof: terminating condition (3.2);

lemma 3 Before LCS* **finds a meeting node on** P_{s-t}^*, there exist references $r_s' \in Open_s$ and $r_t' \in Open_t$ such that $n(r_s')$ and $n(r_t') \in P_{s-t}^*$, and $g_s(r_s') = g_s^*(n(r_s'))$ and $g_t(r_t') = g_t^*(n(r_t'))$.
Proof for r_s': Before the first node from source is expanded, there is a reference $r_s^0 \in Open_s$ with $n(r_s^0) = s$, $g_s(r_s^0) = g_s^*(n(r_s^0)) = 0$ (step 1) and trivially $r_s' = r_s^0$ because $s \in P_{s-t}^*$. Each time a reference $r_s \in Open_s \mid g_s(r_s) = g_s^*(n(r_s))$ is selected for expansion, a new reference $r_s' \mid n(r_s') \in P_{s-t}^*$ is inserted in $Open_s$ with $g_s(r_s') = g_s^*(n(r_s'))$,

for the following reasons: (i) if $n(r_s) \notin Closed_t$ it will certainly be expanded (condition at step 4.1). Otherwise, this reference is considered as a meeting node; (ii) $g_s(r'_s) = g_s(r_s) + c(r_s, r'_s)$, and then $g_s(r'_s) = g^*_s(n(r'_s))$ because $r_s, r'_s \in P^*_{s-t}$, which is optimal; (iii) $f(r'_s) \leq C^* \leq L_{min}$ (lemmas 0 and 1), and then r'_s cannot be discarded by screening. Therefore, we prove this lemma by induction.

lemma 4 Before LCS* **finds a meeting node on** P^*_{s-t}, every node $n \in P^*_{s-t}$ expanded by LCS* from a reference $r \mid n = n(r)$ have $g_\varsigma(r) = g^*_\varsigma(n)$.

Proof: Suppose the contrary: $r \in Open_s, f(r) \leq f(x)\ \forall x \in Open_s$ and $g_s(r) > g^*_s(n)$. Let P^0_{s-n} be the path already found from s to $n(r)$. Then, there should be an optimal path $P^*_{s-n}|P^*_{s-t} = P^*_{s-n} + P^*_{n-t}$ with cost $C(P^*_{s-n}) = g^*_s(n) < C(P^0_{s-n}) = g_s(r)$ and the algorithm did not find it (if P^*_{s-n} was already found, $n \in Closed_s$, and r would not be expanded because of the condition at 4.1). There is a reference $r'_s \in Open_s \mid n(r'_s) \in P^*_{s-n}$ with $g_s(r'_s) = g^*_s(n(r'_s))$ (from lemma 3) because n was not expanded before. Now, if $r'_s \in Open_s$ at this moment, $g_s(r'_s) < g_s(r)$ (from hypothesis), $g_t(r'_s) \leq g_t(r) + k(r'_s, r)$ (consistency) and therefore $f(r'_s) < f(r)$, contradicting the selection of r for expansion.

corollary 1 The additional condition at step 4.1 is always true for nodes along the optimal path. Therefore, we can remove it and the above mentioned properties still hold.

corollary 2 Every meeting node $n = n(r) \in P^*_{s-t}$ corresponding to a reference r have $g_\varsigma(r) + g(n) = f^*(n)$. This value is then assigned to L_{min} (steps 3.3, 4.2, 4.5).

theorem 1 LCS* is complete.

Proof: If there is a finite path P^*_{s-t} with cost C^* and the cost of any arc is at least c_{min}, then for any reference r to a node $n(r)$ further than $M = C^*/c_{min}$ steps from s, we have $f(r) \geq g(r) \geq Mc_{min} = C^*$. Clearly, no reference r to a node $n(r)$ further than M steps from s is ever expanded, for by the lemma 3, the corollary 2 to lemma 4, and the lemma 2.

theorem 2 LCS* is admissible.

Step a: From lemma 3 (there are references with $f(r) \leq f^*(n(r))$ in the open lists), lemma 0 ($f^*(n(r)) = C^* \leq L_{min}$), and lemma 2, we prove that at a certain time a reference to a node in the optimal path will be selected for expansion and recognized as a meeting node with $g_\varsigma(r) + g(n) = f^*(n)$.

Step b: After a meeting node m in the optimal path is selected (comment on corollary 2), no other meeting node can be selected, for $L_{min} \leftarrow f^*(m) = C*$.

3.2 Improved Dynamic Estimator

The power of a heuristic admissible algorithm is not how fast it goes toward the goal, but how efficiently it can compute a higher $f(n) \quad \forall n \mid f(n) \leq f^*(n)$. This is known as *pruning power* of the estimator or algorithm [11]. Values greater than 0 can be used for Ω_s, Ω_t and Py_s, Py_t to improve the pruning power of LCS^* and cause the algorithm to terminate sooner, without loosing consistency and admissibility properties.

Let $\Omega_\tau = min[g(p(m)) \; - \; k(p(m), \tau)] \; \forall m \in Open_\tau$. This resistivity value is the minimum overhead to get to τ from anywhere outside $Closed_\tau$, including nodes at the boundary of $Closed_\tau$. It corresponds to the Min idea of [6], and can be added to any static estimate $k(n, \tau)$. Since the improved dynamic estimate $k(n, \tau) + \Omega_\tau \leq k^*(n, \tau) \; \forall n \mid n \notin Closed_\tau$ or $n \in p(m) \mid m \in Open_\tau$, and no estimate is needed when $n \in Closed_\tau$ and $n \notin p(m) \mid m \in Open_\tau$, the value can be used by the algorithm (Fig. 3).

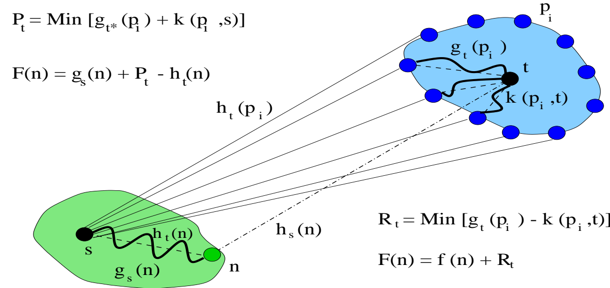

Fig. 3. Dynamic estimation using Py and Ω (R) values

To make the estimates consistent among all nodes in a given open list, we have to use the same value Ω_τ for all references $r \in Open_\varsigma$. By doing so, we do not break consistency, for $g_\tau(r_1) + \lambda \leq g_\tau(r_2) + \lambda + k(r_1, r_2)$ for any value of λ. It does not modify the ordering in which the references are stored (by $f(r)$) in the open list. The absolute values of $f(r)$ are only used to compare to L_{min}, and higher values are desired to improve the pruning power. This is why $f(r) = g_\varsigma(r) + k(n(r), \tau)$ is still used in $Open_\varsigma$ (step 4.4), but $f(r) \geq L_{min} - \Omega_\tau$ is used as pruning condition (step 3.2).

Analogously, let $Py_\tau = min[g(p(m)) + k(p(m), \varsigma)]$. This penalty is the minimum cost of an $s-t$ path through node p. It includes the additional overheads of $k(p(m), \varsigma)$ and $k(p(m), \tau)$. Then, for any node n, $F(n) = g_\varsigma(n) + Py_\tau - h_\tau(n)$ is an improved admissible estimation of $f^*(n)$, and corresponds to the Max idea of [6]. It is used together with the resistivity estimator because in some cases it may be smaller than even the static f.

4 Conclusions

This paper has presented LCS*, a new bidirectional and heuristic path search algorithm, and the proofs of its admissibility. The pruning power of LCS* is enhanced by using information from the "opposite search", implemented as single values added to the static estimates, what preserves admissibility. The new algorithm opens several possibilities for further research on cooperative searches, direction choosing criteria, among others.

References

1. R. Dechter and J. Pearl, *Generalized Best-First Search Strategies and the Optimality of A**, J. Assoc. Comput. Mach., 32 (1985), pp. 505–536.
2. S. Ghosh and A. Mahanti, *Bidirectional Heuristic Search with Limited Resources*, Information Processing Letters, 40 (1991), pp. 335–340.
3. P. E. Hart, N. J. Nilsson, and B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems, Science and Cybernetics, SSC-4 (1968), pp. 100–107.
4. M. Johann, A. Caldwell, A. Kahng, and R. Reis, *A New Bidirectional Heuristic Shortest Path Search Algorithm*. Accepted for publication in the proceedings of the International ICSC Congress on Intelligent Systems and Applications, 2000.
5. H. Kaindl, G. Kainz, A. Leeb, and H. Smetana, *How to Use Limited Memory in Heuristic Search*, in Proceedings of the International Joint Conference on Artificial Intelligence, vol. 1, 1995, pp. 236–242.
6. H. Kaindl, G. Kainz, *Bidirectional Heuristic Search Reconsidered*, in Journal of Artificial Intelligence, vol. 7, 1997, pp. 283–317.
7. A. L. Koll and H. Kaindl, *Bidirectional Best-First Search with Bounded Error: Summary of Results*, in Proceedings of the International Joint Conference on Artificial Intelligence, 1993, pp. 217–223.
8. J. B. H. Kwa, *BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm*, Artificial Intelligence, 38 (1989), pp. 95–109.
9. G. Manzini, *BIDA*: An Improved Perimeter Search Algorithm*, Artificial Intelligence, 75 (1995), pp. 347–360.
10. N. J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill Book Co., 1971, ch. State-Space Search Methods, pp. 43–79.
11. J. Pearl, *Heuristics*, Addison-Wesley, 1984, ch. Formal Properties of Heuristic Methods, pp. 73–87+.
12. I. Pohl, *Bi-Directional Search*, in Machine Intelligence, 1971, pp. 127–140.