ON FINDING THE FIRST LINK OF A FASTEST PATH*

David E. Kaufman
Robert L. Smith
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 95-25

May 1993
Revised November 1995

*This work was supported in part by the Intelligent Transportation Systems Research Center of Excellence at the University of Michigan.

On Finding the First Link of a Fastest Path

David E. Kaufman
Robert L. Smith
Department of Industrial and Operations Engineering
University of Michigan
Ann Arbor, Michigan 48109

May 14, 1993

Abstract

We consider algorithms designed to find the first link of a fastest path more rapidly than the entire path can be computed. In application to IVHS, such algorithms would allow drivers requesting in-vehicle route guidance to begin their trips as soon as the optimal first link could be computed, rather than waiting for the computation of the entire path before starting.

We demonstrate that the $1A^G$ algorithm (Lark and White [1]) for first-link determination can be applied by maintaining easily computed bounds on the cost of paths to the destination, even though the algorithm as originally proposed requires values of potentially difficult integer programming problems. We suggest how an IVHS system might provide the initial bounds on path durations to destination which the algorithm requires, and we suggest directions for continuing investigation.

1 Introduction

One vision of IVHS route guidance consists of an in-vehicle unit computing the fastest path for its driver's desired trip, given link travel times transmitted from a central traffic authority reflecting current conditions in the road network. However, once the driver has input the desired trip, the fastest path computation may take longer than the driver wishes to wait. Not only does this introduce delay at the beginning of the trip, but it would also work against acceptance of the new technology by end-users and reduce societal return on investment in IVHS infrastructure.

In this paper, we discuss modifications to the A^G fastest path algorithm (Lark and White [1]) which would identify the driver's optimal first decision, i.e., the first link on the fastest path, in less time than would be required to find the entire fastest path. The driver would then be free to begin his trip by executing that first decision, so that his driving time on the first link of his trip is also being used to compute succeeding optimal decisions.

Lark and White's $1A^G$ algorithm for this purpose requires at every node expansion the solution of a multiple choice integer program which expresses dynamic programming relationships. However, we show that under conditions relevant to IVHS, alternate versions of the algorithm using easily obtained surrogates for the IP solutions must terminate after exactly the same number of node expansions as the main algorithm, while still finding the optimal first decision.

Our discussion will be in terms of what has been called elsewhere quasi-dynamic routing, using a network model with static link travel times presumed to be updated by external sources to reflect changing road conditions. However, the time-dependent (fully dynamic) fastest path case should present little additional difficulty.

2 Review of A^*-type algorithms

Let the network consist of nodes $\mathcal{N} = \{1, \ldots, N\}$ and links $\mathcal{A} \subset \mathcal{N} \times \mathcal{N}$, with nonnegative link travel times $c(n, n')$ for all $(n, n') \in \mathcal{A}$. We assume that the network has no cycles of negative cost. Let the trip be from node 1 to node N.

The A^* algorithm (a generalization of Dijkstra's algorithm; see e.g. Pearl [2]) requires a *heuristic* function h. We denote the duration of the fastest path from any node n to node N by $h^*(n)$, and we assume that h is *admissible*, i.e., that $h(n) \leq h^*(n)$ for all $n \in \mathcal{N}$. Similarly, let $g^*(n)$ be the duration of the fastest path from 1 to n. We begin the algorithm by initializing $g(1) = 0$ and $g(n) = \infty$ for $n > 1$ and the set of open nodes to $S = \{1\}$.

The main step of the algorithm is then to choose a node n whose *evaluation function* $f(n) = g(n) + h(n)$ is minimal over S for *expansion.* Node expansion consists of removing n from S and setting $g(n') = \min\{g(n'), g(n) + c(n, n')\}$ for all $n' \in SCS(n)$, where $SCS(n) = \{n' : (n, n') \in \mathcal{A}\}$ is the successor set of n. For each n' whose g value was updated, set a backpointer to n indicating the optimal predecessor node, and add n' to S. This step repeats until node N is chosen for expansion, at which time the algorithm terminates with $g(n) = g^*(n)$ equal to the duration of the fastest path.

The A^G algorithm (Lark and White, 1993) generalizes A^* to potentially more informative heuristic information. It uses a *heuristic set* H of nonnegative-valued functions on $\mathcal{N}$, calling H admissible if $h^* \in H$. Nodes are chosen for expansion in A^G by the evaluation function $f(n) = g(n) + \ell(n)$, where $\ell(n) = \inf\{h(n) : h \in H\}$ for $n \in \mathcal{N}$. (A^* using ℓ as its heuristic function is the special case $H = \{h : \ell \leq h\}$.) The expansion step is as in A^*, with one difference: define

$$U(n, n', H) = \sup\{h(n) - h(n') : h \in H\},$$

and ignore links (n, n') such that $U(n, n', H) < c(n, n')$.

Such links are ignored because the dynamic principle of optimality assures us that

$$h^*(n) = \min\{c(n,n') + h^*(n') : n' \in SCS(n)\}. \tag{1}$$

Thus if $U(n,n',H) < c(n,n')$, then since $h^* \in H$, we find that $h^*(n) < c(n,n') + h^*(n')$, so that n' does not achieve the minimum in (1) and hence (n,n') is not the optimal next link from n. The link is thus *pruned* from the network. In particular, if $n = 1$, then the link is eliminated from consideration as an optimal *first* decision. We now review other ways to prune links from the network.

3 Pruning strategies

We can propose a trivial modification to A^* which may find optimal first decisions quickly, without resorting to A^G. Recall for each node n' touched in any expansion, we maintain a pointer back to a predecessor node n for which $g(n) = c(n,n') + g(n')$. We can easily trace these pointers back to node 1 to identify the fastest known path to n'. Suppose that some node $m \in SCS(1)$ does not lie on the pointer path for any open node n'. This implies that for all open nodes n', we have already identified a path from 1 to n' faster than the path through m. Since any optimal path from 1 to N must pass through some open node, m cannot lie on any such path, and thus we may prune link $(1,m)$.

Another simple way to prune links applies when the lower-bounding heuristic function ℓ is *monotone*, i.e.,

$$\ell(n) \leq c(n,n') + \ell(n') \qquad \text{for all } (n,n') \in \mathcal{A}. \tag{2}$$

Then it is guaranteed that the fastest path from 1 to n' discovered before expanding n' is in fact optimal (cf. [2]). In the monotone case we may thus prune link (n,n') if the backpointer from n' does not point to n when n' is expanded. Our analysis will assume monotonicity of ℓ, and we will consider this pruning strategy to be in effect throughout our development.

The mechanics of the A^G algorithm provide a more sophisticated pruning mechanism, by allowing us to prune any link $(1,m)$ for which $U(1,m,H) < c(1,m)$. However, unless the heuristic set constrains h^* very tightly at node 1 and its successors, this pruning condition will occur infrequently. Therefore, Lark and White propose another algorithm, called $1A^G$, for the purpose of finding optimal first links. Their idea is to use node expansions to contract the heuristic set. Since U is a supremum over H, shrinking H decreases U, pruning more links from the network if the decreases in U values are sufficient to cross their thresholds. The contraction is achieved by noting that the optimal heuristic satisfies equation (1) and thus we may discard other heuristic functions. Formally, let $H_0 = H$ and at the kth node expansion (from node n, say) let

$$\Delta H_k = \{h : h(n) = \min\{c(n,n') + h(n') : n' \in SCS(n)\}\} \tag{3}$$

and use $H_k = H_{k-1} \cap \Delta H_k$ in the next iteration. Since $h^* \in \Delta H_k$ for all k, the contraction operation preserves admissibility while increasing pruning power.

The greatest difficulty in implementing A^G and $1A^G$ may be in computing values of U, which are values of optimization problems. In particular, even if H is a polytope, the heuristic set is nonconvex after the first contraction. The new constraint (3) can be expressed by introducing 0-1 variables $y(n,n')$ and constraints

$$h(n) \leq c(n,n') + h(n') \qquad \text{for all } n' \in SCS(n) \tag{4}$$

$$h(n) \geq c(n,n') + h(n') - M(1 - y(n,n')) \qquad \text{for all } n' \in SCS(n) \tag{5}$$

$$\sum_{n' \in SCS(n)} y(n,n') = 1 \tag{6}$$

for a sufficiently large value of M. Applying this formulation directly in $1A^G$ would require that at each iteration k we solve a number of 0-1 multiple-choice integer-linear programs with side constraints, whose number of multiple choice sets is k.

Instead, we will investigate methods for quickly generating solutions which may be infeasible with respect to some of the added constraints (4)–(6) but which provide upper bounds U' on the values of U. Clearly, $U'(n,n',H_k) \leq c(n,n')$ would then imply $U(n,n',H_k) \leq c(n,n')$, allowing link (n,n') to be pruned. However, we will also show that although in general U' may be greater than U, $U(n,n',H_k) < c(n,n')$ only if $U'(n,n',H_k) < c(n,n')$, and thus the alternate methods have the same pruning power as $1A^G$ but with less computational effort.

4 Tightening the upper bounds

Henceforth, we assume an admissible initial heuristic set H_0 of rectangular form $H_0 = \{h : \ell_0 \leq h \leq u_0\}$, so that we begin with upper and lower bounds $u_0(n)$ and $\ell_0(n)$ on each value $h^*(n)$. We further assume that ℓ_0 and u_0 are monotone. We now consider the effect of contracting H_0 by applying only the upper-bounding constraints (4) at each node expansion. This is the same as requiring h to be monotone with respect to (n,n'), a condition we know to be true of h^*.

For $k = 1,2,\ldots$ let n_k be the kth node expanded and let H'_k be the heuristic set obtained by applying upper-bound constraints (4) with $n = n_k$ to contract H'_{k-1} (taking $H'_0 = H_0$). It is evident that for each k, $U(n,n',H_k) \leq U(n,n',H'_k)$ for all $(n,n') \in \mathcal{A}$, since $H_k \subset H'_k$. Now define for all heuristic sets H and links (n,n')

$$U'(n,n',H) = \sup\{h(n) : h \in H\} - \inf\{h(n') : h \in H\}.$$

Then clearly $U(n,n',H) \leq U'(n,n',H)$.

Algorithm $1A^G$ already uses the values $U(n,n',H_k)$ to prune the A^* search tree. Consider two more algorithms, SU (Strong Upper bounding) and WU (Weak Upper bounding), which use $U(n,n',H'_k)$ and $U'(n,n',H'_k)$, respectively, to prune the tree. The inequalities we have just noted make it clear that WU prunes no more links than SU, which prunes no more links than $1A^G$. Note also that because h^* satisfies (4), H_k is admissible for all k, and thus

WU and SU are both guaranteed to find optimal first decisions.

Although the number of iterations (node expansions) to termination is nondecreasing from $1A^G$ to SU to WU, the effort per iteration has the opposite ordering. $U(n,n',H'_k)$ (used in SU) is the value of a linear program, while $U(n,n',H_k)$ (used in $1A^G$) is the value of an integer program. And as we now show, values of $U'(n,n',H'_k)$ (used in WU) are even easier to obtain, by maintaining decreasing upper bounds $u_k(n)$ on $h(n)$ for each n at each iteration k.

Initially, the upper bounds are simply given by u_0. In the kth node expansion (of node $n = n_k$), set

$$u_k(n) = \min\{u_{k-1}(n), \min\{c(n,n') + u_{k-1}(n') : n' \in SCS(n)\}\}. \tag{7}$$

Then, if equation (7) caused an updating (i.e., $u_k(n) < u_{k-1}(n)$), set $u_k(m) = \min\{u_{k-1}(m), c(m,n) + u_k(n)\}$ for each link $(m,n) \in \mathcal{A}$ not already pruned. Then, if $u_k(m) < u_{k-1}(m)$, proceed back from m in the same fashion. It is easily shown that this occurs for only finitely many nodes as long as the network contains no negative directed cycles. Then, for all nodes not updated already, set $u_k(m) = u_{k-1}(m)$. By recursion on the iteration number k we can easily prove that $u_k(n) = \sup\{h(n) : h \in H'_k\}$ for all $n \in \mathcal{N}$. It is also evident that $\ell_0(n) = \inf\{h(n) : h \in H'_k\}$ when ℓ_0 is monotone. Therefore we can compute $U'(n,n',H'_k) = u_k(n) - \ell_0(n')$ without recourse to linear programming.

Thus the effort per iteration of WU is less than that for SU, particularly when we note that SU requires solution of a linear program for each link considered for pruning, while WU finds the values of U' for all links by only one pass backwards through the network. The following results show that WU is also just as effective as SU by the standard of number of iterations until termination.

Lemma 1 *Suppose that the set of nodes expanded through iteration k of algorithm SU does not include n', for which there exists $(n,n') \in \mathcal{A}$. Suppose that $u_k(n) - c(n,n') \geq \ell_0(n')$.*

Then there exists $h \in H'_k$ *such that* $h(n) - h(n') = c(n, n')$.

Proof: Let $h \in H'_k$ be such that $h(n) = u_k(n)$; such h exists because $u_k(n) = \sup\{h(n) : h \in H'_k\}$ and H'_k is compact. We now modify this h to satisfy the conclusions of the theorem. First, set $h(n') = u_k(n) - c(n, n')$. By hypothesis, $h(n') \geq \ell_0(n')$, and we can show by monotonicity of u_0 that $h(n') \leq u_0(n')$. Thus since n' has not been expanded, the only constraints defining H'_k which might be violated by this change are the upper-bound constraints associated with links (m, n') for nodes m already expanded. For such links, set $h(m) = \min\{h(m), c(m, n') + h(n')\}$. It is easily shown that such $h(m)$ satisfies the simple bounds given by ℓ_0 and u_0. However, this update may have created a violation of other upper-bound constraints. But if we propagate the update back just as we do for u_k, a recursive argument makes it evident that if the updates ever stop, the resulting solution must be in H'_k. As before, there can be only finitely many updates in the absence of negative cycles. Thus the updating stops with a heuristic function h meeting the desired conditions. ∎

Proposition 2 *If an link* (n, n') *is pruned by SU in iteration* k*, then it is pruned by WU in iteration* k.

Proof: We consider two cases. First, suppose n' is one of the first k nodes expanded. Then by monotonicity of ℓ_0, the optimal path from 1 to n is known. If that path does not contain (n, n'), then that link is eliminated by both algorithms. If the path does contain (n, n'), then there exists $h \in H'_k$ (in particular, h^*) such that $h(n) = c(n, n') + h(n')$. Thus $U(n, n', H'_k) \geq c(n, n')$ and hence SU does not prune (n, n').

Now suppose n' is not expanded in or before iteration k. If WU does not prune (n, n'), then

$$U'(n, n', H'_k) = u_k(n) - \ell_0(n') \geq c(n, n').$$

Lemma 1 then implies that there exists $h \in H'_k$ such that $h(n) - h(n') = c(n,n')$. Thus $U(n,n',H'_k) \geq c(n,n')$ and again SU does not prune (n,n'). ∎

Thus the monotonicity of ℓ_0 and u_0 assures us that we can apply the easily computed values $U'(n,n',H'_k)$ with exactly the same pruning effectiveness as the linear program values $U(n,n',H'_k)$. Since both algorithms terminate when they eliminate all but one of the possible first decisions, they must both terminate after the same number of iterations. In the next section, we demonstrate that a similar result holds for the integer program values $U(n,n',H_k)$.

5 Tightening the lower bounds

In the previous section, we constructed a decreasing sequence $\{u_k, k = 1,2,\ldots\}$ of upper bounds on h^*, the duration of the fastest path to completion. We used these together with the *a priori* lower bounds ℓ_0 to prune links, and in particular first decisions, from the search tree. In this section we construct an increasing sequence of lower bounds $\{\ell_k, k = 1,2,\ldots\}$ which will contribute to even more effective pruning.

As in the previous section, we assume a rectangular initial heuristic set with monotone upper and lower bounds, and we let $u_k(n)$ for each node n and iteration k be as defined at and after equation (7). Now, at the expansion of node $n = n_k$ in iteration $k \geq 1$, we perform the update

$$\ell_k(n) = \max\{\ell_{k-1}(n), \min\{c(n,n') + \ell_{k-1}(n') : n' \in SCS(n)\}\} \tag{8}$$

(note the similarity with equation (1)). Then just as with u_k, we allow the update to propagate back; if equation (8) caused an updating (i.e., $\ell_k(n) < \ell_{k-1}(n)$), set

$$\ell_k(m) = \max\{\ell_{k-1}(m), \min\{c(m,m') + \ell_{k-1}(m') : m' \in SCS(m)\}\}$$

for each link $(m,n) \in \mathcal{A}$ not already pruned, and so on. (As before, this operation must terminate finitely in the absence of negative directed cycles.) For all nodes not already updated, set $\ell_k = \ell_{k-1}$. By recursion on the iteration number k we can prove that $\ell_k(n) = \inf\{h(n) : h \in H_k\}$ for all $n \in \mathcal{N}$. Therefore we can compute

$$U'(n,n',H_k) = u_k(n) - \ell_k(n)$$

without recourse to integer programming.

Consider two algorithms, SL (Strong Lower bounding) and WL (Weak Lower bounding), which use $U(n,n',H_k)$ and $U'(n,n',H_k)$, respectively, to prune the tree. (SL is identical to $1A^G$, but we will continue to call it SL to emphasize its relation to the other algorithms under discussion.) It is evident that WL takes less work per iteration but may require more iterations than SL. However, as we now show, both algorithms terminate in exactly the same number of iterations.

Lemma 3 *Suppose that the set of nodes expanded through iteration k of algorithm SL does not include n', for which there exists $(n,n') \in \mathcal{A}$. Suppose that $u_k(n) - c(n,n') \geq \ell_0(n')$. Then there exists $h \in H_k$ such that $h(n) - h(n') = c(n,n')$.*

Proof: The construction used in proving Lemma 1 also suffices here. The update step applied there enforces the dynamic programming functional equation (1) as well as the monotonicity constraints (4). ∎

Proposition 4 *If an link (n,n') is pruned by SL in iteration k, then it is pruned by WL in iteration k.*

Proof: We consider two cases. First, suppose n' is one of the first k nodes expanded. Then by monotonicity of ℓ_0, the optimal path from 1 to n is known. If that path does not

contain (n, n'), then that link is eliminated by both algorithms. If the path does contain (n, n'), then there exists $h \in H_k$ (in particular, h^*) such that $h(n) = c(n, n') + h(n')$. Thus $U(n, n', H_k) \geq c(n, n')$ and hence SL does not prune (n, n').

Now suppose n' is not expanded in or before iteration k. If WL does not prune (n, n'), then

$$U'(n, n', H_k) = u_k(n) - \ell_k(n') = u_k(n) - \ell_0(n') \geq c(n, n').$$

Lemma 3 then implies that there exists $h \in H_k$ such that $h(n) - h(n') = c(n, n')$. Thus $U(n, n', H'_k) = c(n, n')$ and again SL does not prune (n, n'). ∎

Therefore, while WL requires relatively little work per iteration, it performs exactly as many iterations as SL to find the optimal first decision.

6 Application to IVHS

Our discussion so far has assumed that simple upper and lower bounds on the duration of the fastest path to complete the trip from any node are readily available, and these bounds satisfy monotonicity. We now consider how an IVHS environment makes it possible to obtain heuristic information in this form without excessive computational effort.

In IVHS, we may assume knowledge of constant characteristics of the road network, including complete network topology (the "road layout") and physical lengths and speed limits on all links. Assuming that all vehicles obey the speed limits, then we know the minimum link travel times for all links. By solving the static shortest path problem from all nodes to the destination node N, we find lower bounds $\ell_0(n)$ on the duration of the fastest path from n to N, independent of current traffic conditions. Thus ℓ_0 could be computed off-line and stored as data for real-time computation. It is trivial to show that this choice of ℓ_0 is monotonic.

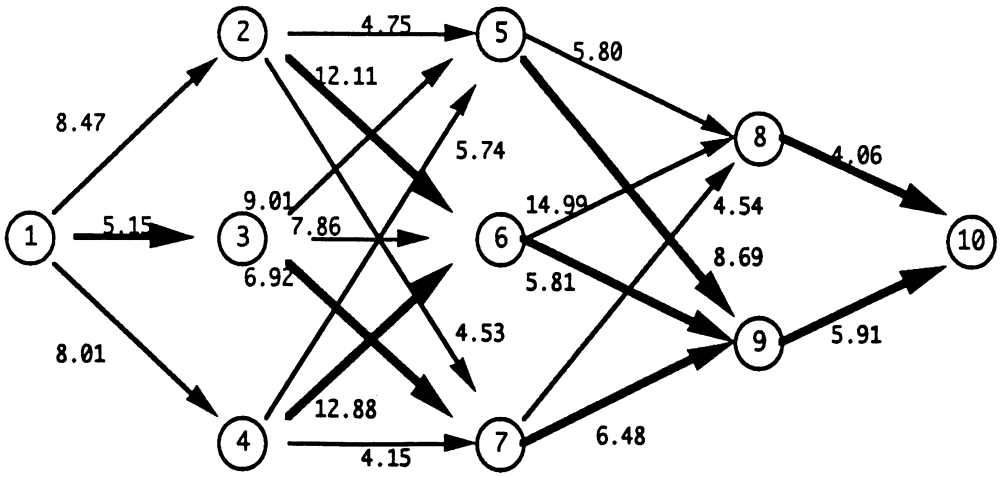

Figure 1: Sample network

We could construct the upper bounds similarly, but it might be difficult to set reliable maximum link travel times except by setting them so high that the resulting bounds would be too weak to be useful. Instead, we suggest a quick method which would be applied to data on current conditions collected in real-time. We will apply the WL algorithm to the network with link travel times as observed in real-time, to find fastest paths. So to get upper bounds, we can simply examine the durations of arbitrary (or randomly generated) paths to the destination. This computation is considerably faster than a fastest path computation, since it requires no comparisons and only one addition per node, rather than one per link. The resulting upper bound u_0 is guaranteed to be monotonic.

Consider the network shown in Figure 1, with current link travel times as shown, where the desired trip is from node 1 to node 10. The links drawn in heavy lines in the figure have been arbitrarily chosen for the computation of upper bounds, which appear in Table 1. Also appearing in the table are a lower bound heuristic, produced from data not shown here, and the values of h^*, the optimal path duration from each node. It can be verified that the A^* algorithm would expand nodes in the order $\{1,3,4,2,5,7,8,6,10\}$ with expansion of node 10 causing termination. The order of node expansion is determined solely by ℓ_0, thus the

n	1	2	3	4	5	6	7	8	9	10
$\ell_0(n)$	11.53	7.96	9.75	7.97	5.28	7.05	7.05	2.72	5.04	0
$h^*(n)$	20.67	13.13	15.52	12.75	9.86	11.72	8.60	4.06	5.91	0
$u_0(n)$	24.45	23.83	19.30	24.60	14.60	11.72	12.39	4.06	5.91	0

Table 1: Duration of optimal path to destination, and initial bounds thereon.

same order is followed in WU and WL.

Although WU succeeds in pruning five intermediate links from the network, it does not terminate until expansion of node 10, saving no iterations compared to A^*. WL, however, does better. It eliminates two more links than does WU when node 7 is expanded. Both algorithms eliminate links $(5,9)$ and $(7,9)$; thus the optimal path must go through either node 6 or node 8. The decision then hinges on whether or not node 8 should be reached through node 5, because both algorithms have by this point identified link $(1,2)$ as optimal to node 5 and link $(1,3)$ as optimal to nodes 6 and 7. However, the stronger pruning power of SL succeeds in eliminating $(2,5)$ from the set of links which may lie on the fastest path from node 1 to node 10, and therefore link $(1,3)$ must be the optimal first decision. (The optimal node sequence is 1,3,7,8,10.) Therefore, algorithm SL saves two complete node expansion steps (nodes 8 and 6).

7 Conclusions and future research directions

We have demonstrated that the $1A^G$ algorithm as proposed by Lark and White [1] can find the first decision on a fastest path in fewer iterations than are required for the more well-known A^* algorithm. We have further demonstrated that the heuristic information required by the algorithm can be generated rapidly to satisfy a monotonicity condition which substantially reduces the effort per iteration of $1A^G$ (or WL, as we have called it.)

However, WL generally requires more work per iteration than A^*, because of the effort required to keep updating the bounds ℓ_k and u_k in WL. Therefore, computational experiments

are needed to determine which algorithm is favored by the tradeoff between iterations and work per iteration. The updated bounds also suggest stopping rules for ϵ-optimal solutions, that is, first decisions on paths which have duration at most ϵ more than the fastest path. Computational experiments would also determine which approach is faster for ϵ-optimal first decisions, for use when drivers are willing to accept slightly suboptimal paths in exchange for reduced time waiting for routing advice.

References

[1] Lark, J.W., III, and White, C.C., III. 1993. A best-First Search Algorithm Guided by a Set-Valued Heuristic. Working paper, The University of Michigan.

[2] Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company, Reading, Mass.