9

Bi-Directional Search

Ira Pohl*

Thomas J. Watson Research Center
IBM Corporation, New York

Abstract

A technique that has proved useful in shortest path and other discrete optimization computations has been bi-directional search. The method has been well tested in the two-node shortest-path problem providing substantial computational savings. A natural impulse is to extend its benefits to heuristic search.

In the uni-directional algorithms, the search proceeds from an initial node forward until the goal node is encountered. Problems for which the goal node is explicitly known can be searched backward from the goal node. An algorithm combining both search directions is bi-directional.

This method has not seen much use because book-keeping problems were thought to outweigh the possible search reduction. The use of hashing functions to partition the search space provides a solution to some of these implementation problems. However, a more serious difficulty is involved. To realize significant savings in bi-directional search, the forward and backward search trees must meet in the 'middle' of the space. The potential benefits from this technique motivates this paper's examination of the theoretical and practical problems in using bi-directional search.

INTRODUCTION

Problem-solving programs have principally worked in one direction – from the initial state to the goal state. In searching for a path to the goal state they have relied on estimators demonstrating progress to the goal. This they achieve by a computation that compares the current states with the goal state. GPS (Newell, Shaw and Simon 1959) and its derivatives (Quinlan and Hunt 1968) use 'means-ends' analysis; the Graph Traverser (Doran and Michie 1966) or HPA (Pohl 1969a,b) use a 'heuristic' distance estimator; Multiple Slagle and

* present address: Board of Studies Information and Computer Science, University of California, Santa Cruz.

Bursky 1968) uses a 'merit' function. Each of these problem-solving systems order the candidate states, expanding one with a 'best' value. The best state is the one estimated to require the least additional search to reach the goal state (*see* Nilsson 1969 for a general reference).

In many problems attacked by search procedures, the goal state is explicitly known. It is possible to conduct a search backward from the goal state to the initial state. The usual forward methods are 'turned around' with the goal and initial node interchanged and the directions of the steps reversed. Indeed both a forward and backward search can be independently attempted. Rather than two independent searches, it can be advantageous to combine the two searches into a bi-directional search with each contributing part of the solution. The motivation is that search trees grow exponentially and two shorter search diameters generate fewer states than a single longer diameter tree. This provides a strong incentive for developing bi-directional heuristic algorithms even in the face of more complex book-keeping and coordination. Moreover, good results have been achieved with these methods in the shortest-path problem and the network flow problem (Pohl 1969a).

BI-DIRECTIONALITY IN THE TWO-NODE SHORTEST-PATH PROBLEM

The two-node shortest-path problem is of fundamental importance to operations research. The history of computational methods for this problem and closely related problems may be found elsewhere (Dreyfus 1969).

A directed graph $G(X, E)$ is a set of nodes $X=\{x_1, x_2, \ldots\}$ and a set of edges $E=\{(x_i, x_j) | x_i, x_j \in X,\ x_j \in \Gamma(x_i)\}$, where Γ is the successor mapping, $\Gamma: X \to 2^X$, the mapping of X into its power set. Analogously there is a predecessor mapping $\Gamma^{-1}: X \to 2^X$ where $\Gamma^{-1}(x_j)=\{x_i | x_i, x_j \in X,\ x_j \in \Gamma(x_i)\}$.

A path from a to b is a sequence of nodes $(x_0, x_1, x_2, \ldots, x_r)$ where $x_0=a$, $x_r=b$ and $(x_i, x_{i+1}) \in E$. For the edges to have lengths is defined as a mapping $\ell: E \to R^+$ (the positive reals). The special mapping $\ell_1: E \to \{1\}$ is the cardinality length. The length function is extended to paths using as the definition, $\ell(\mu)=\sum_{i=0}^{r-1} \ell(x_i, x_{i+1})$. The two-node shortest-path problem is for a given G, ℓ, s and t, to find a path $\mu=(s, \ldots, t)$ which has minimum length over all such paths in G.

The basic algorithm used for this problem is Dijkstra's (1959). It and its predecessor, Moore's algorithm (Moore 1959), have been adapted for heuristic search and we will exhibit a bi-directional extension.

Dijkstra's algorithm F

The F signifies forward.

s=start (initial) node, t=terminal (goal) node.

S=set of nodes reached from s whose minimum distance from s is known.

$\tilde{S}$=set of nodes reached from S by one edge but are not in S.

$g_s(x)$=current shortest distance from s to x.

$wf(x)$=the immediate predecessor node of x along the path from s to x.

These sets and values will change with each iteration of the algorithm.

Forward algorithm

1. (initialize) $S=\{s\}$, $\tilde{S}=\{\Gamma(s)\}$, $g_s(s)=0$, $g_s(x)=\ell(s,x)$, $wf(x)=s$.
2. (find minimum) $n=\{x \mid x\in\tilde{S}\wedge\forall_{y\in\tilde{S}}[g_s(x)\leqslant g_s(y)]\}$.
3. (expand) If $n=t$ go to 4, else add n to S and remove n from $\tilde{S}$. Look at each $x\in\Gamma(n)$; (a) if $x\in S$ ignore it; (b) if $x\notin\tilde{S}$ then place it in $\tilde{S}$ with $g_s(x)=g_s(n)+\ell(x,n)$ and $wf(x)=n$; (c) if $x\in\tilde{S}$ then compare $g_s(x)$ with $g_s(n)+\ell(n,x)$ and if the latter is smaller change values of g_s and wf as in (b). Go to 2.
4. (halt) $g_s(t)$ is the length of the shortest path which is

$$(s, wf^k(t), \ldots, wf^2(t), wf(t), t).$$

where $wf^i(x)=\underbrace{wf(wf\ldots(wf(x)\ldots))}_{i \text{ times}}$

The algorithm is very simple and the reader can easily prove its correctness or convince himself by trying an example (*see* figure 1). The basic work is the iteration of Steps 2 and 3. This is done each time a node is added to s; thus the number of nodes in set S upon termination of the algorithm is a reasonable measure of the work done.

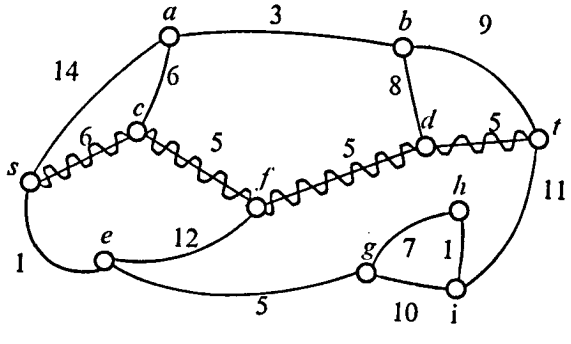

Figure 1. The zig-zag line traces the shortest path from s to t

The analogous algorithm working backwards from t would also solve the shortest-path problem. The sets of interest would then be:

T=set of nodes which have a path to t whose minimum distance is already found.

$\tilde{T}$=set of nodes one edge from T but which are not in T.

$g_t(x)$=current shortest distance from x to t.

$wt(x)$=the immediate successor node of x along the path from x to t.

Step 1 would now initialize $T=\{t\}$ and Step 3 would expand by looking at $x \in \Gamma^{-1}(n)$. The terminating condition would be 'encountering s'.

Call the forward algorithm F and the backward algorithm B; we now wish to combine them into a bi-directional algorithm.

Bi-directional shortest-path algorithm: BSPA

(1) (initialize) perform F_1 (the first step of the forward algorithm) and B_1.

(2) (strategy) decide to go forward (go to Step 3) or backward (go to Step 4).

(3) (forward expansion) perform F_2 and F_3. F_3 must check not 'if $n=t$', but if $n \in T$ go to 5 else it returns to Step 2.

(4) (backward expansion) perform B_2 and B_3 with the check for halting being 'if $n \in S$ go to 5 else go to 2'.

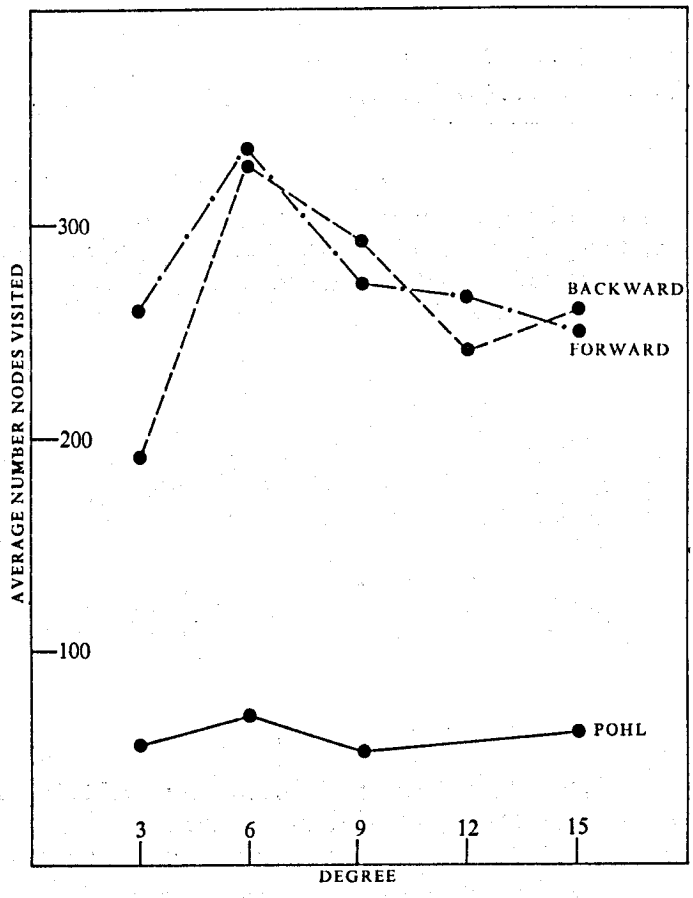

Figure 2. The results from 10 shortest-path problems in 500 node graphs. The graphs were generated randomly with a given average degree (the number of edges incident to a node). Backward and forward are uni-directional Dijkstra algorithms and Pohl is the bi-directional algorithm using cardinality comparison

(5) (halt) the shortest path is the path $(s, wf^j(x), \ldots, wf(x), x, wt(x), \ldots, wt^k(x), t)$ which minimizes

$$\forall_{x \in S \cap (T \cup \tilde{T})} [g_s(x)+g_t(x)].$$

(Note $\forall_{x \in T \cap (S \cup \tilde{S})}$ would also work.)

Two points are non-trivial in performing its extension from the Dijkstra algorithm. The complexity of the terminating condition, which is not the obvious $x \in S \cap T$, and the fact that any rule for deciding Step 2 gives a correct algorithm. These points are explained elsewhere (Pohl 1969a, Dreyfus 1969). Thus what is wanted is the most efficient decision rule for Step 2. If the decision rule is computationally complicated a heavy penalty is paid, for it is executed at each iteration. The simple rules of alternating between forward and backward direction or trying to move equidistantly from the end points have been suggested (Nicholson 1966). These rules implicitly hypothesize symmetry. However, a better strategy involves the size of $|\tilde{S}|$ and $|\tilde{T}|$ which reflects the density of the forward and backward neighborhoods. Going in the direction of the smaller number of candidates is making progress in a sparser region. This rule inserted in Step 2 of BSPA, i.e.,

(2) if $|\tilde{S}| \leqslant |\tilde{T}|$ then go to 3 else go to 4.

is called the *cardinality comparison* principle. It is analyzed in great detail in Pohl (1969a) and is supported by both experimental data and theoretical arguments. Some of these findings are summarized in figure 2. Bi-directionality has a general usefulness in other combinatorial problems. An especially important problem which immediately benefits from this improvement in path finding is the network-flow problem (*see* Appendix). Consequently, one would like to add this technique to the store of aids for heuristic search.

BI-DIRECTIONAL HEURISTIC SEARCH

Hart, Nilsson, and Raphael (1968) extended the Dijkstra algorithm into an algorithm, A^*, which used not g_s but g_s+h_s; where the function h_s is an estimate of the remaining distance from x to t along the shortest path. Doran and Michie (1966) in the Graph Traverser used h_s instead of g_s, and Pohl (1969b) used $(1-\omega)g_s+\omega h_s$ where ω is a constant $0 \leqslant \omega \leqslant 1$. The heuristic case is hopefully more efficient in the number of nodes looked at in finding a solution path.

Formally, the heuristic is an estimator of the shortest-path distance between two nodes.

$$h_s: X \rightarrow R^+ \text{ (the non-negative reals)}$$

an estimate of $\ell(\mu(x, \ldots, t))$ where μ is a minimum-length path from x to t.

$$h_k: X \rightarrow R^+$$

and is an estimate of $\ell(\mu(s, \ldots, x))$, the minimum-length path from s to x.

These heuristic functions are to be computer algorithms which, given the node and its associated state and the state description of the appropriate

endpoint (i.e. $g_s(x)$ uses t as its endpoint and $g_t(x)$ uses s), utilizes these descriptions to provide the estimate. Hence the problem of capturing appropriate features relates to a function on these features providing an accurate estimate of distance. As an example, to model GPS, some of its expositions (Ernst 1969, Sandewall 1969) indicate that a function it could utilize is

$$h_s(x) = \sum_{i=1}^{n} 10^i \cdot d_i(x)$$

where associated with x is a state vector $(x^{(1)}, x^{(2)}, \ldots, x^{(n)})$ and with the goal node t its state vector $(t^{(1)}, t^{(2)}, \ldots, t^{(n)})$

$$d_i(x) = \begin{cases} 0 & x^{(i)} = t^{(i)} \\ 1 & x^{(i)} \neq t^{(i)} \end{cases} \quad \text{the Boolean difference vector.}$$

This leads to attempts to reduce differences according to their 'difference ordering'. This h_s is not necessarily an accurate distance estimator, but corresponds to a Hamming-like metric. Moreover, discussing the efficiency of GPS style search functions is not the point, rather this is one example of a heuristic function. Our principal concern is to utilize bi-directionality which was beneficial in the shortest-path problem. (The graphs there did not have associated states and will be called an uninterpreted domain.) Following Hart, Nilsson, and Raphael (1968) we first wish to produce algorithms which find the shortest path.

Bi-directional Heuristic Path Algorithm: BHPA

The extension of BSPA is accomplished by using $f_s = g_s + h_s$ and $f_t = g_t + h_t$ for evaluation functions and changing the termination step.

BHPA: The notation has mostly been encountered in the previous algorithms.

$a_{\min}$ is equal to the minimum distance for paths already found.

$f_s(x) = g_s(x) + h_s(x)$ where $g_s(x)$ is calculated as in the forward algorithm.

$f_t(x) = g_t(x) + h_t(x)$ where

$g_t(x) = g_t(n) + \ell(x, n)$, $x \in \Gamma^{-1}(n)$.

1. Place s in S and calculate $f_s(x)$ for $x \in \Gamma(s)$ placing them in $\bar{S}$ and make the corresponding calculations for t. Set $a_{\min} := inf$; where inf is a number larger than the length of any path you will encounter.
2. Decide to go forward (Step 3) or go backward (Step 4).
3. Select $n = \{x \mid x \in \bar{S} \wedge \forall_{y \in \bar{S}}[f_s(x) \leqslant f_s(y)]\}$.

Place n in S and check $x \in \Gamma(n)$ for the following possibilities.

(a) $x \in \Gamma(n) \cap \bar{S} \wedge f_s(x) > g_s(n) + \ell(n, x) + h_s(x)$ then replace f_s by this new smaller value.

(b) $x \in \Gamma(n) \cap S \wedge f_s(x) > g_s(n) + \ell(n, x) + h_s(x)$ then place x into $\bar{S}$ with the new value at the same time removing it from S.

(c) $x \in S \cup \bar{S} \wedge x \in \Gamma(n)$ then place x in $\bar{S}$.

(d) Otherwise no change in S and $\bar{S}$. Go to Step 5.

4. Select $n=\{x \mid x \in \bar{T} \wedge \forall_{y \in T}[f_t(x) \leqslant f_t(y)]\}$ and perform the same computations as in Step 3 with respect to $x \in \Gamma^{-1}(n)$ using T and $\bar{T}$. Go to Step 5.
5. If $n \in S \cap T$ then

$$a_{\min} := \min\,(a_{\min}, g_s(n)+g_t(n))$$

If $$a_{\min} \leqslant \max\left[\min_{x \in S}(f_s(x)), \min_{x \in \bar{T}}(f_t(x))\right]$$

then halt and the path that gave $a_{\min}$ is the shortest path. Otherwise go to Step 2.

As in Hart, Nilsson, and Raphael (1968), we prove for the appropriate choice of h_s and h_t, BHPA will find a shortest path.

Let $h_p(x, y)$ = the minimum distance to get from x to y. Then the heuristic functions are required to satisfy

$$h_s(x) \leqslant h_p(x, t),\ h_t(x) \leqslant h_p(s, x) \tag{A}$$

$$h_p(x, y)+h_s(y) \geqslant h_s(x),\ h_p(x, y)+h_t(x) \geqslant h_t(y) \tag{B}$$

Then by lemma 2 of Hart, Nilsson, and Raphael (1968) the nodes placed in set $S(T)$ have had their shortest path from s (to t) found. Consider BHPA halting with path $\mu=(s, x_1, x_2, \ldots, x_k, t)$ when in fact the shortest path is $\lambda=(s, y_1, \ldots, y_j, t)$.

The algorithm halted with $x_i \in S \cap T$, x_i the intersection node. If $x_i=s$ or $x_i=t$ then we have essentially the undirectional case with theorem 1 of Hart, Nilsson, and Raphael (1968) holding.

Otherwise, $a_{\min}=g_s(x_i)+g_t(x_i)=\ell(\mu)>\ell(\lambda)$

By induction, there exists

$$y_l \in \bar{S},\ y_{l-1} \in S \text{ and } y_{r+1} \in T,\ y_r \in \bar{T}$$

with $i-1<r+1$ or else the path λ would have been found. But

$$f_s(y_l)=g_s(y_l)+h_s(y_l) \leqslant \ell(\lambda) < a_{\min}$$

and $f_t(y_l) \leqslant \ell(\lambda) < a_{\min}$.

$\therefore$ Contradiction, Step 5 of BHPA was not satisfied and halting would not have occurred.

This argument demonstrates:

Theorem

BHPA with its heuristic functions satisfying (A) and (B) will find a shortest path.

When not concerned with finding a shortest path BHPA can use (*see* Pohl 1969a,b)

$$f_s=(1-\omega_s)g_s+\omega_s h_s, \text{ and}$$
$$f_t=(1-\omega_t)g_t+\omega_t h_t,\ 0 \leqslant \omega_s, \omega_t \leqslant 1$$

without concern for h_s and h_t satisfying (A) and (B). While no longer insuring a shortest path, these functions often find a solution path looking at fewer nodes and consequently performing less work. Additionally, the termination step can be simplified to '(Step 5) halt when $x \in S \cap T$.'

REDUNDANCY AND INTERSECTION

In using BHPA, several book-keeping problems occur which were not of concern (or not as serious) in uni-directional heuristic search. Two of the implementation problems of BHPA are *redundancy* and *intersection*. The problem spaces searched have many alternate paths to the same node causing redundant encounters. In BHPA when the shortest path is required it is necessary to check if an improved path has been found, and even when this is unnecessary the addition of extra copies of nodes into S, $\bar{S}$, T, and $\bar{T}$ add to storage requirements and work. The intersection problem is the need to recognize that a node has appeared in both the forward and backward trees ($x \in S \cap T$). Both *redundancy* and *intersection* are the same problem, namely, a record-searching problem.

It is well known from sorting theory that hashing functions are the best techniques available for unordered record searches (*see* Morris 1968). The evaluation function, when suitably normalized, immediately presents itself as a candidate. Otherwise a suitable function can be concocted using the state description.

Using the evaluation function of a node may not be possible because it is dependent on how it was reached (its path history). Also it may be advantageous as we shall see later for BHPA to modify the evaluation function. It is often the case that h_s and h_t are invariant and can be used when appropriately normalized. However, $h_s(x) \neq h_t(x)$ except accidentally, thus the two trees require separate hashing functions. Finally searches often are stuck in an area whose nodes have approximately the same evaluation value. If this occurs the heuristic functions would not adequately distribute the nodes to unique addresses in memory – of prime importance in efficient hashing searches.

With these objections to the use of the evaluation function, we turn to a concocted function on the state description. A particular node has its associated state description which may be thought of as a vector

$$\vec{v}_i = (v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(k)}).$$

(*see* examples in Doran and Michie 1966, Nilsson 1969, Pohl 1969b). As an idealized example, consider that we wish to hash into a size 2^M address space where the $v_i^{(j)}$ take on uniformly-distributed values from 0 to $2^{n_i}-1$. A simple hashing function is

$$hash\,(\vec{v}_i) = \left(\sum_{j=1}^{k} v_i^{(j)} \cdot 2^{\sum_{r=1}^{j-1} n_r} \right) \bmod 2^M$$

which can be computed by adds and shifts. More generally,

$$hash\,(\vec{v}_i) = \sum_{j=1}^{k} c_j v_i^{(j)} \text{ could be used,}$$

where the c_j are chosen to make the computation economical and to uniformly distribute states among address space.

Each hash address becomes a class of states chained together. A check for redundancy or intersection consists of computing the hash value and doing a chained search of all nodes with this value. Experimentally, a function of the above form was used with the fifteen puzzle – dividing the 16!/2 positions into 680 equivalence classes. The search was limited to 1000 nodes and hashing produced approximately two orders of magnitude decrease in running time in comparison to simple linear searches for redundancy and intersection. The hash chains were between 0–30 long for a tree of size 1000, which is a considerable improvement over searching the whole tree. The behavior of heuristic search procedures is to perform local searches where much of the state is left alone. In these instances the notions of Hamming distance and separability of nearly-identical states should be considered, if savings warrant the more refined coding theory calculations (*see* Hanan and Palermo 1963).

The importance of this computational idea should not be underestimated. The idea recurs throughout combinatoric and enumerative programming. In some sense the hash provides a semi-canonical form. The uses of a hash places the bi-directional search inner loop almost on a par with the uni-directional search inner loop.

STRATEGIES IN BI-DIRECTIONAL SEARCH

In the ordinary shortest-path problem the best rule for deciding on the direction of search is cardinality comparison. The same idea is plausible in the heuristic search case. Often the path back from the goal node is more easily derived than the path forward from the start node. Cardinality comparison attempts to take advantage of any asymmetry in difficulty as measured by the number of current alternatives. However, unlike the uninterpreted shortest-path problem, this is not enough to improve upon uni-directional heuristic search.

In the shortest-path problem the forward and backward search trees expand as spherical wave fronts about their respective endpoints (*see* figure 3). The search trees when heuristics are not used must touch with the combined search radii $\tau_s+\tau_t=\tau^*$, the shortest-path distance. A heuristic search proceeds in a more directed fashion, often searching some narrow conical region. The size of the spaces searched are very large and the two cones typically have a combined search radius exceeding τ^*. Gains in efficiency occur if the two search cones have the sum of their search radii equal to approximately the single heuristic search radius. The payoff is to be able to solve problems of length $2k$ for only twice the cost of uni-directionally solving problems of length k. The crucial problem is to attempt to coerce the search trees into an intersection midway between the endpoints.

In working with the fifteen-puzzle (Doran and Michie 1966, Pohl 1969a), a first attempt was to make no special changes, but to allow each heuristic search tree to aim at its opposite endpoint. The typical result when this was

tried with the fifteen-puzzle was to have both search trees growing almost complete but separate solution paths, with intersection occurring near one or the other of the endpoints.

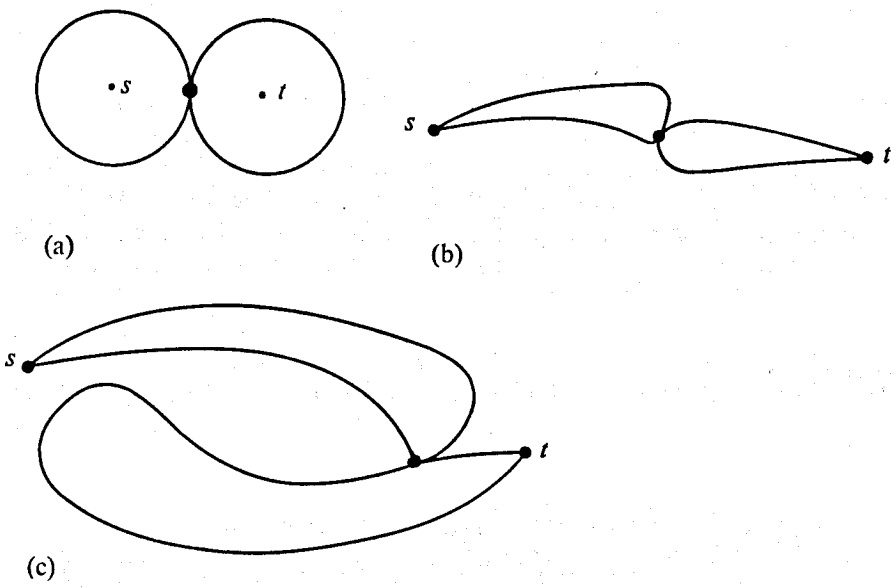

Figure 3. Characteristic bi-directional searches: (a) expansion in the uninterpreted problem, $h=0, f_s=g_s, f_t=g_t$; (b) ideal bi-directional heuristic search; (c) typical bi-directional heuristic search.

This is not surprising; many alternate solution paths exist in this space. Once a tree is making progress along one of these routes, it will not investigate adjacent paths. This is the key to efficiency in heuristic search and at the same time the difficulty in making two searches collide. Two strategies were employed to cope with this difficulty in the fifteen-puzzle: however, neither gave improvement.

Intermediate state conjecture

Suppose the initial state is $\vec{v}_s=(v_s^{(1)}, \ldots, v_s^{(k)})$ and the termin*al* state is $\vec{v}_t=(v_t^{(1)}, \ldots, v_t^{(k)})$, then a conjectured intermediate state would be some

$$\vec{v}_i=(v_i^{(1)}, \ldots, v_i^{(k)}) \text{ such that } h(\vec{v}_s, \vec{v}_i)\approx h(\vec{v}_i, \vec{v}_t)\approx 1/2h(\vec{v}_s, \vec{v}_t).$$

The two search trees are now directed toward the conjectured intermediate state. In the fifteen-puzzle, a simple strategy is to first order the top half of the puzzle and then, leaving the top half alone, order the remainder. A component-additive estimator could produce this effect by having a heuristic function $h(\vec{v})=\sum_{i=1}^{k} h_i(v^{(i)})$ restricted to $\hat{h}(\vec{v})=\sum_{i=1}^{[k/2]} h_i(v^{(i)})$.

The conjectured intermediate position provides a modified estimator or restriction on the original function over the full state vector. This does not mean that intersection has to occur at the conjectured position, but the modified estimator will direct both search trees to an appropriate middle ground. The idea of planning or proving lemmas involves an actual subdivision of the problem into two, going from s to i and from i to t. This is an important idea in reducing the search. As Minsky (1961, page 442) states: 'Only schemes which actively pursue an analysis toward obtaining a set of *sequential goals* can be expected to extend smoothly into increasing complex problem domains.' The heuristic distance model often provides means for constructing such sequential goals.

Shaping

Another possibility is to continually update the heuristic function in use. Since collision between the wave fronts is wanted, use the heuristic function to estimate the distance of a node to the opposite wave front. This can be done by assuming the last node expanded represents the target node for the opposite tree. The frequency of updating the heuristic function can be parameterized and experimented with. The extreme case of never updating from the original start and goal nodes produces our first algorithm. Highly frequent updating when tried was the worst in behavior. The heuristics are approximations in a very large space. To constantly readjust the 'targets' of the two search trees produces highly circuitous and lengthy search trees. Possibly if the heuristic is reliable within certain limits and the search has definitely reduced the remaining distance to the goal, then updating the target would be reasonable.

A plausible suggestion (Doran 1966) is to pairwise evaluate nodes expanded and expand both trees from the pair of minimum separation. However, the state space search is squared without any guarantee of commensurate search reduction. The effect in the uninterpreted shortest-path problem would be to mimic a policy of strict alternation. In this example, $h=0$ and the pair of nodes $\{x, y\}$ with $x \in \bar{S}$ and $y \in \bar{T}$, which have minimum evaluation using g_s and g_t is just

$$x \; s.t. \; \forall_{w \in \bar{S}}(g_s(x) \leqslant g_s(w)) \text{ and}$$
$$y \; s.t. \; \forall_{w \in \bar{T}}(g_t(y) \leqslant g_t(w))$$

which are those nodes which would separately have been expanded when the decision strategy was alternation.

While BSPA on the fifteen-puzzle had poor success, only limited experimental attempts were made using shaping or intermediate positions. These ideas and the suggestion of Doran need further testing, in particular in the form proposed by Michie (personal communication) where the states of the original problem are re-coded as state-pairs, thus implementing bi-directional search in single-tree form with corresponding reduction of the number of evaluations.

FINAL REMARKS

The question of bi-directional search is not distinct from other problems in heuristic search. Subgoal generation and pruning present similar problems in how to appropriately use distance estimators.

Bi-directional search is useful in standard combinatorial problems. It is a technique used by human problem-solvers. Potentially, its benefits derive from adding only a linear cost with possible exponential savings. Therefore, a large incentive exists for continuing work on the practical and theoretical problems associated with this device.

REFERENCES

Dijkstra, E. (1959) A note on two problems in connection with graphs. *Numerische Mathematik*, **1**, 269–71.

Doran, J. (1966) Doubletree searching and the Graph Traverser. *Research Memorandum EPU-R-22*, Edinburgh: Department of Machine Intelligence and Perception, Edinburgh University, Scotland.

Doran, J. E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc. A*, **294**, 235–59.

Dreyfus, D. (1969) An appraisal of some shortest path algorithms. *Operations Research*, **17**, 395–412.

Ernst, G. (1969) Sufficient conditions for the success of GPS. *J. Ass. comp. Mach.*, **16**, 517–33.

Ford, L. & Fulkerson, D. (1962) *Flows in Networks*. Princeton, N.J.: Princeton University Press.

Hanan, M. & Palermo, F. (1963) An application of coding theory to a file address problem. *IBM J. Res. & Dev.*, **7**, 127–9.

Hart, P., Nilsson, N., & Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. & Cyber.*, **4**, 100–7.

Minsky, M. (1961) Steps toward artificial intelligence. *Proc. IRE*, **49**, Also in *Computers and Thought*, pp. 406–50 (eds Feigenbaum, E. & Feldman, J.) New York: McGraw-Hill, 1963.

Moore, E. (1959) The shortest path through a maze. *Proc. of an Int. Symp. on Theory of Switching, Part II*, April 1957, pp. 285–92. Cambridge, Mass.: Harvard University Press.

Morris, R. (1968) Scatter storage techniques. *Comm. Ass. comput. Mach.*, **11**, 38–44.

Newell, A., Shaw, J. & Simon, H. (1959) Report on a general problem-solving program. *Proc. Int. Conf. Inf. Processing*, pp. 256–64, Paris: UNESCO House.

Nicholson, T. (1966) Finding the shortest route between two points in a network. *Comput. J.*, **9**, 275–80.

Nilsson, N. (1969) Problem-solving methods in artificial intelligence. *SRI Report*. Stanford Research Institute, Menlo Park, California.

Pohl, I. (1969a) Bi-directional and heuristic search in path problems, *SLAC Report No. 104*. Stanford, California.

Pohl, I. (1969b) First results on the effect of error in heuristic search. *Machine Intelligence 5*, pp. 219–36, (eds Meltzer, B. & Michie, D.). Edinburgh: E.U.P.

Quinlan, J. & Hunt, E. (1968) A formal deductive problem solving system. *J. Ass. comput. Mach.*, **15**, 625–46.

Sandewall, E. (1969) A planning problem solver based on look-ahead in stochastic game trees. *J. Ass. comput. Mach.*, **16**, 364–83.

Slagle, J. & Bursky, P. (1968) Experiments with a multipurpose theorem-proving heuristic program. *J. Ass. comput. Mach.*, **15**, 85–99.

APPENDIX: BI-DIRECTIONAL SEARCH FOR FLOW-AUGMENTING PATHS

An important class of optimization problems is the network flow problems. The graph has nodes designated as source and sink and the edges labeled with given capacities. The problem is to maximize the flow in the network from the source to the sink. This formulation follows the treatment in Ford and Fulkerson (1962).

We have a graph $G(X, E)$

$s \in X$ is the source

$t \in X$ is the sink

$c: E \rightarrow R^+$ are the capacities, and the flow in any particular edge cannot exceed its capacity. A flow of value v from s to t is a function $f: E \rightarrow R^+$ satisfying the following constraints:

$$\sum_{y \in \Gamma(x)} f(x, y) = \sum_{y \in \Gamma^{-1}(x)} f(y, x) \qquad x \neq s, t \quad \text{conservation equations}$$

$$\sum_{y \in \Gamma(s)} f(s, y) = v \qquad \text{flow out of source}$$

$$\sum_{y \in \Gamma^{-1}(t)} f(y, t) = v \qquad \text{flow into sink}$$

and $0 \leqslant f(x, y) \leqslant c(x, y)$ for $\forall (x, y) \in E$.

Flows satisfying these conditions are feasible flows and the problem is to find the maximum feasible flow.

The standard algorithm for this problem is the Ford-Fulkerson network flow algorithm. This procedure finds a 'flow-augmenting path' using a path-searching procedure and then updates the network's current feasible flow by this augmented flow. This is iterated until no such change can be made. This occurs when some cut-set of edges is saturated, i.e., when $c(x, y) = f(x, y)$, edge (x, y) is saturated; and a cut-set of edges satisfying this saturation condition is a set of edges which when removed from G leaves s disconnected from t.

Uni-directional procedure for finding flow augmenting paths [Routine A from Ford and Fulkerson (1962) page 18].

1. s is labeled $(-, \textit{excess}\ (s) = \infty)$

The source is now labeled but unexpanded. All other nodes are unexpanded and unlabeled.

2. Select any labeled but unexpanded node and expand. If no such node exists terminate with the maximal flow.

3. $\forall y \in \Gamma(x)$, where x was the node selected in (2), with y unlabeled and such that $f(x, y) < c(x, y)$ assign the label $(x+, \textit{excess}\ (y))$, where $\textit{excess}\ (y) = \textit{min}(\textit{excess}\ (x), c(x,y) - f(x, y))$.

$\forall y \in \Gamma^{-1}(x)$ which are unlabeled and such that $f(y, x) > 0$ assign the label $(x-, \textit{excess}\ (y))$, where $\textit{excess}\ (y) = \textit{min}\ (\textit{excess}\ (x), f(y, x))$.

4. If t is labeled in this step perform a flow augmentation with the path just found, updating the edge flows appropriately and return to (1). Otherwise go to (2).

The flow-augmenting path can be traced back from t and the additional flow *excess* (t) may be added to each edge along this path. This process can be accomplished bi-directionally (Pohl 1969a) with the termination condition being that x is labeled in both the forward and backward search.

Bi-directional search for augmenting path

1. s is labeled $(-, \infty)$, t is labeled $(+, \infty)$.
2. Decide on which direction to expand.

If no node exists as a candidate for expansion in either the forward or backward direction halt with the maximal flow.

3. Labeling proceeds as in the uni-directional case.
4. Halt if some node x labeled in (3) is labeled in both the forward and backward direction. Otherwise go to Step 2. The flow augmentation is the smaller of the two *excess* (x) numbers.

This method has two advantages over the uni-directional procedure. First, the path search ordinarily involves fewer node labelings as in the shortest-path problem. Secondly, the bi-directional search tends to 'randomize' the search more which leads to faster saturation. An augmentation pursues the same path until it reaches the first previously unsaturated edge which has just been saturated. Then it diverges with a likelihood of not producing a large augmentation. The randomization provided by bi-directionality avoids, more easily than uni-directional search, covering large portions of the previously saturated paths. Some experimental evidence with a bi-directional network flow algorithm has demonstrated these gains.