A PROBLEM SIMILARITY APPROACH TO DEVISING HEURISTICS: FIRST RESULTS

John Gaschnig
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

Here we describe an approach, based upon a notion of problem similarity, that can be used when attempting to devise a heuristic for a given search problem (of a sort represented by graphs). Thé proposed approach relies on a change in perspective: instead of seeking a heuristic directly for a given problem P1, one seeks instead a problem P2 easier to solve than P1 and related to P1 in a certain way. The next step is to find an algorithm for finding paths in P2, then apply this algorithm in a certain way as a heuristic for P1. In general, the approach is to consider as candidates problems P2 that are "edge subgraphs" or "edge supergraphs" of the given problem P1. As a non-trivial application, we show that a certain restricted form of sorting problem (serving as P2) is an edge supergraph of the 8-puzzle graph (P1). A simple algorithm for solving this sorting problem is evident, and the number of swaps executed in solving an instance thereof is taken as a heuristic estimate of distance between corresponding points in the 8-puzzle graph. Using the A* algorithm, we experimentally compare the performance of this "maxsort" heuristic for the 8-puzzle with others in the literature. Hence we present evidence of a role for exploiting certain similarities among problems to transfer a heuristic from one problem to another, from an "easier" problem to a "harder" one.

1. Introduction *, **

Many combinatorially large problems cannot be solved feasibly by exhaustive case analysis or brute force search, but can be solved efficiently if a heuristic can be devised to guide the search. Finding such a heuristic for a given problem, however, usually requires an exercise of creativity on the part of the researcher.

Research to date on devising heuristics has spanned several problem-solving domains and several approaches. In some efforts, the objective has been to optimize, using some adaptation scheme over a number of trials, the values of coefficients determining the relative weighting of several preselected terms in an evaluation function, so as to maximize the overall performance (e.g., [Samuel 1959], [Samuel 1967], [Rendell 1977]). Other approaches to automatic generation of heuristics include [Ernst, et al., 1974] and [Rendell, 1976]. Related efforts have focused on what might be called "disciplined creativity", enunciating general principles or rules of thumb that a person may apply to the problem at hand (e.g., [Polya 1945], [Wickelgren 1974]).

The approach to devising heuristics proposed here differs in perspective from these previous efforts: instead of seeking a heuristic directly, one seeks instead a problem P2 that is easier to solve than the given problem P2 and is related to P2 in a certain way.* The next step is to find an algorithm for finding paths in P2, then apply this algorithm in a certain way as a heuristic for P1. As an elementary example, the rectilinear distance function is an efficient heuristic for finding paths in a "Manhattan street pattern" graph even when some (but not too many) of the streets have been blockaded (i.e., some edges are removed from the graph). Generalizing, the approach is to consider as candidates problems P2 that are "edge subgraphs" or "edge supergraphs" of the given problem P1. As a non-trivial application, we show that a certain restricted form of sorting problem (serving as P2) is an edge supergraph of the 8-puzzle graph (P1). A simple algorithm for solving this sorting problem is evident, and the number of swaps executed in solving an instance thereof is taken as a heuristic estimate of distance between corresponding points in the 8-puzzle graph. Using the A* algorithm, we experimentally compare the performance of this "maxsort" heuristic for the 8-puzzle with others in the literature.

The general class of problems to which the present approach can be applied are those state space problems that can be represented as graphs, in which the objective is to find a path from a given initial node in the graph to

* This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US. Government.

** Author's present address: Artificial Intelligence Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025

* This perspective is somewhat akin to that on which are based the results of backward error analysis (as defined in the numerical analysis literature), in which one asks of a matrix inversion algorithm, for example, not how accurate are the answers that it computes, but rather how different from the given problem is the problem for which the computed answers are the exact answers.

a given goal node. * Such problems can be solved using the A* best-first search algorithm, which iteratively grows a tree or graph of partial paths from the initial node, at each step expanding the node (along the edge of this tree or graph) that appears to be "best".** The definition of "best" is specified by assigning a number to each node generated, whose computation typically involves the value of a heuristic function K(s, t) used to estimate the distance in the graph from an arbitrary node s in the graph to another arbitrary node t (where t is taken to be the goal node in a particular instance of search). Devising heuristic functions that estimate distance between points in a given graph is the present practical objective.

Additional theoretical objectives motivated this work: to investigate how "similar" or "dissimilar" problems differ in structural properties, and how such structural differences relate to difference in difficulty to solve the problem. Investigations of relatively simple problems serve this purpose well. Note that we are not interested in the 8-puzzle (our principal example) per se, but as a concrete instance with which to investigate general principles.

Section 2 defines the notions of "edge subgraph" and "edge supergraph", and illustrates their relation to heuristic transfer using a simple example -- a "Manhattan street pattern" graph and variants thereof. Section 3 applies the basic approach to a larger problem -- the 8-puzzle. Section 4 discusses the generality of the results and poses future tasks.

2. Problem Similarity: Edge Subgraphs and Supergraphs

The basic idea considered in this paper can be illustrated using a simple example. Consider the graphs depicted in Figures 1a, 1b, and 1c, which we shall refer to as MSUB44, M44, MSUP44, respectively. We note that these three graphs have identical numbers of nodes, and that the edges of MSUB44 comprise a subset of those of M44; similarly the edges of M44 comprise a subset of those of MSUP44. We shall say therefore that MSUB44 is an edge subgraph of M44 and that similarly M44 is an edge subgraph of MSUP44. Likewise we also say that M44 is an edge supergraph of MSUB44 and that MSUP44 is an edge supergraph of M44. These relations of edge subgraph and edge supergraph generalize formally in the obvious way for the class of problem graphs, which we define to be any finite, strongly connected graph G = (V, E) having no self-loops and no multiple edges. This definition includes the familiar problems cited in the preceding section. *

To illustrate how edge subgraph or edge supergraph similarity between problem graphs is related to heuristic transfer, we now consider three cases in turn: (1) that M44 is the given problem to be solved; (2) that MSUB44 is the given problem; and (3) that MSUP44 is the given problem. In general we wish to solve an arbitrary instance of a given problem graph P, that is to find a path from an arbitrary initial node s_r in P to an arbitrary goal node s_g. (We denote such a problem instance I of a problem graph P thus: $I = (s_r, s_g)$.) This trivial example serves as a vehicle for introducing several general concepts.

The task of finding a path between arbitrary initial and goal nodes in the M44 graph (or in some larger version of this "Manhattan street pattern" graph) is trivial. A simple algorithm for solving instances of this problem graph is readily evident: comparing the coordinates of the current node (starting with the initial node s_r), move iteratively up (or down as the case may be) until the vertical coordinate of the goal node is reached, then move right (or left) iteratively until the goal node is reached. Call this algorithm L (for "L-shaped solution path").

If MSUB44 is taken instead of M44 as the problem to be solved, the A* approach seems more attractive than attempting to devise an algorithm like algorithm L, because of the additional cases an algorithm of the latter sort must account for: besides comparing the coordinates of the current node with those of the goal node, it must also be prepared to make detours when necessary.

To use A* to solve MSUB44, one must supply a heuristic function K(s, t) that estimates distance from node s to node t. Let $h_P(s, t)$ denote the actual distance in P from s to t, assuming edges have unit weight. In the case K(s, t) = h(s, t) the distance estimate is exact; it is well known [Hart et al. 1968] that this case minimizes the number of nodes expanded -- the number is exactly $h(s_r, s_g)$, the distance from initial node to goal node.** For M44, it is evident that $h_{M44}(s, t) = |x_s - x_t| + |y_s - y_t|$, where x_s and y_s are the x and y coordinates, respectively, of node s. For MSUB44 a symbolic formula for h(s, t) is not so compact, since it must distinguish more cases. An alternative to enunciating h(s, t) for MSUB44 is to use $K(s, t) = h_{M44}(s, t)$, i.e., use the distance function of M44 to approximate the distance function of MSUB44.

It is easy to show, as follows, that solution paths found for an arbitrary problem instance of MSUB44 using

* Elementary examples of such problems include the 8-puzzle, The Tower of Hanoi, the water jug problem, missionaries and cannibals problem and other "toy" problems familiar in the literature [Nilsson 1971, pp. 39-41, 77-78], [Jackson 1974, pp. 81-84, 110-115], [Raphael 1976, pp. 79-86], [Wickelgren 1974, pp. 49-57, 78-80 cf.]. Somewhat less frivolous examples include certain algebraic manipulation problems [Doran & Michie 1966, pp. 254-255], [Doran 1967, pp. 114-115], and a version of the traveling salesperson problem [Doran 1968], [Harris 1974].

** The Graph Traverser algorithm [Doran & Michie 1966] and the HPA algorithm [Pohl 1970a] are essentially the same as A*.

* Note that the 8-puzzle graph consists of two disjoint components. For present purposes, we consider search within one such component. Also, we assume that a problem graph is specified in practice by a successors function: SUC(s) denotes, for any node s in the graph, the of nodes v_i for which there exists an edge from s to v_i. Typically SUC(s) is implemented by a set of operators, each of which transforms a given state s into another state v_i, provided the operator's precondition is satisfied. In the 8-puzzle context, one such operator might have the effect of moving the hole upward if it is not in the top row of the board.

** To achieve this bound, one must be careful in selecting a strategy for resolving ties among nodes for which equal values were assigned.

$K(s, t) = h_{M44}(s, t)$ will be of minimal length. If P1 and P2 are any problem graphs such that P1 is an edge subgraph of P2 (denoted $P1 \subseteq_e P2$), then the distance between arbitrary nodes s and t in P2 is never more than the distance between the corresponding points in P1, i.e., $h_{P2}(s, t) \leq h_{P1}(s, t)$. Hence using the heuristic function $K(s, t) = h_{P2}(s, t)$ for solving P1 it follows that $K(s, t) \leq h_{P1}(s, t)$. Hence by the A* admissibility theorem ([Hart, et al. 1968], [Gelperin 1977]) the solution path found is always of minimum length when using $K(s, t) = h_{P2}(s, t)$ in solving P1.*

As a third case, take MSUP44 as the problem to be solved. Now $h_{MSUP44}(s, t) \leq h_{M44}(s, t)$, hence taking $K(s, t) = h_{M44}(s, t)$ overestimates the distance from s to t in MSUP44. Instead of using A*, however, note that algorithm L can be used to find paths in MSUP44, although they will not necessarily be of minimum length. (Whether this is important depends on the application.) In general, if $P2 \subseteq_e P1$ then an algorithm that finds paths in P2 can also be used to find paths in P1 (assuming the same encoding scheme is used for identifying corresponding nodes in P1 and P2).

These simple examples illustrate a general approach: given a problem graph P1 to be solved, first identify a problem graph P2 that is an edge subgraph or edge supergraph of P1. We call P1 the <u>given problem</u> and P2 the <u>transfer problem</u>. Then find a way of computing $h_{P2}(s, t)$. (In the preceding examples a simple formula for h_{M44} was readily evident. When a formula for h_{P2} is not evident, we instead attempt to devise an algorithm for computing the values of $h_{P2}(s, t)$ by finding a path in P2 from s to t and counting the number of steps in the path. See section 3.) Then use $h_{P2}(s, t)$ as a heuristic function for solving instances of P1: given a goal node s_g, whenever A* calls for a value to be assigned to a node s in P1, compute $h_{P2}(s,s_g)$.

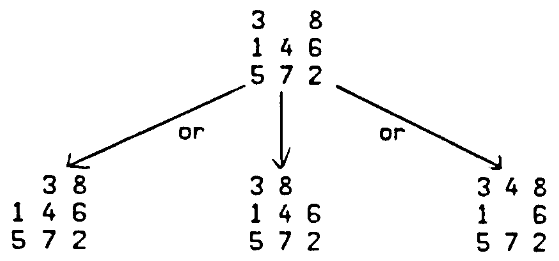
3. Example: A Sorting Algorithm Used as Heuristic for the 8-Puzzle

Section 3.1 defines the "9MAXSWAP" graph and shows it to be an edge supergraph of the 8-puzzle graph. Section 3.2 defines an algorithm, called MAXSORT, for finding paths in the 9MAXSWAP problem graph. Section 3.3 reports experimental results measuring the performance of MAXSORT when used as the basis of a heuristic distance-estimating function for the 8-puzzle, and compares this observed performance with those of other known heuristics for the 8-puzzle. This section also proposes and tests another related heuristic for the 8-puzzle. Section 3.4 reports experimental measurements of how closely the heuristic function based on MAXSORT approximates the distance function of the 8-puzzle. This larger example serves as a non-trivial application and as a vehicle for introducing additional concepts.

* A* is analyzed mathematically in [Hart, et al. 1978], [Pohl 1970a], [Pohl 1970b], [Nilsson 1971], [Harris 1974], [Vanderbrug 1976], [Munyer 1976], [Munyer & Pohl 1976], [Ibaraki 1977], [Pohl 1977], [Gelperin 1977], [Martelli 1977], [Gaschnig 1979], and elsewhere. Attempting to apply such analytic results is not a major concern here; in Section 3.3 we measure heuristic performance experimentally.

3.1. The MAXSWAP problem

The 8-puzzle is a one-person game the objective of which is to rearrange a given configuration of eight tiles on a 3x3 board into another given configuration by iteratively sliding a tile into the orthogonally adjacent empty location, like so:



To apply the present approach (which might be called the "edge subgraph/supergraph transfer" approach) to the 8-puzzle, first we must identify a suitable edge subgraph or edge supergraph of the 8-puzzle. Toward this end we note that an 8-puzzle tile configuration can be considered a permutation of the sequence 1,2,...,9, letting 9 correspond to the hole in the 8-puzzle, and numbering the nine locations of the 8-puzzle board in left to right, top to bottom manner, say. Hence the topmost of the four configurations depicted above corresponds to the sequence 398146572.

Now we define the "N-swap" graph to have N! nodes, corresponding to the N! permutations of the sequence 1,2,...,N. An edge connects two vertices of this graph iff the corresponding permutations differ by a single swap. In an obvious way, the behavior of any sorting algorithm which operates by sequentially interchanging pairs of array elements can be mapped into sets of paths in this graph. Taking N = 9, there is evident a one-to-one correspondence between the nodes of 9-Swap (but not the edges) and those of the 8-puzzle graph.

By noting that the effect of moving a tile in the 8-puzzle graph is to exchange two elements of a 9 element vector, it is easy to see that the 8-puzzle graph is an edge subgraph of 9SWAP. Hence the 8-puzzle can be viewed as a restricted version of permutation sorting, such that every swap must include the largest element (i.e., 9), and that certain swaps are forbidden, depending on the location of the largest element.

If we impose the restriction on sorting N-element permutations that every swap must exchange the largest element, N, with some other element, we obtain what might be called the "N-MAXSWAP" graph. For illustrative purposes, Figure 2 depicts the 4MAXSWAP graph. Note that 8-puzzle $\subseteq_e$ 9MAXSWAP $\subseteq_e$ 9SWAP, hence for arbitrary nodes s and t, $h_{9SWAP}(s, t) \leq h_{9MAXSWAP}(s, t) \leq h_{8\text{-puzzle}}(s, t)$. Hence the distance functions of both 9SWAP and 9MAXSWAP underestimate the distance function of the 8-puzzle, but that of 9MAXSWAP is a closer approximation of the

8-puzzle's distance function than that of 9SWAP. * We will use 9MAXSWAP as the transfer problem for the 8-puzzle in this example; next, we must demonstrate the transfer.

3.2. The MAXSORT algorithm

We describe now what seems to be a most peculiar way of solving the 8-puzzle. Given a problem instance, A* iteratively expands nodes in the 8-puzzle graph and requires that a number be assigned to every new node generated, as the basis for deciding which node to expand next. To obtain each such number we propose to solve an entire instance of 9MAXSWAP using an algorithm defined below, and return a number measuring its performance (in fact, the number of swaps it executes). If solving instances of 9MAXSWAP itself required search, this approach might consume more time than if the 8-puzzle were solved using breadth-first search. However, the point of this example is that the 9-Maxswap problem is simpler than the 8-puzzle --- one can readily devise an efficient algorithm for it, as follows.

The basis for an algorithm which we call "MAXSORT" (defined by a SAIL procedure in the test) for finding paths in an N-MAXSWAP graph is the observation that the largest element, N, in the permutation can always be swapped with the element whose proper place in the permutation it occupies, except when N is in the N'th position of the permutation. We illustrate using N = 4. To sort the permutation P = 2341 (into 1234), the algorithm first swaps 3 and 4 producing 2431, in which the element 3 now occupies its proper place. Next swapping 2 and 4 puts 2 in its proper place and then swapping 4 and 1 puts both those elements in their proper places and the algorithm terminates. To implement this policy of simply swapping the largest element 4 with the element whose proper place 4 is occupying, MAXSORT uses an internal auxiliary array B, and begins by assigning to B[i] the location of element i in the input permutation, i.e., B[1:4] = 4123 for the above example. Then MAXSORT simply swaps iteratively P[B[4]] with P[B[B[4]]], updating both P and B at each iteration, until the sort is complete, with the followng exception to this rule.

Sorting 4321 using MAXSORT, the first swap produces 1324, but now 4 is in its proper place, and so we must do something other than swap 4 with itself indefinitely. Instead, the algorithm swaps 4 with the leftmost element of P that is not in its proper place, i.e., 3 in this case, and we proceed as before. (Determining this leftmost element is accomplished efficiently -- using the variable called avail for bookkeeping, the block named "available-spot-found" is executed at most N/3 times.) So each swap executed by MAXSORT moves zero or one new element, other than N, into its proper place and never moves an element, other than N, from its proper place. Table 1 shows the current permutation, the contents of the elements of B, and the value of the variable avail for each step in the above example.

(Trace this sequence as a path in the graph in Figure 2.) *

iteration	permutation	b array	avail
	4 3 2 1	4 3 2 1	1
1			
	1 3 2 4	1 3 2 4	2
2			
	1 4 2 3	1 3 4 2	2
3			
	1 2 4 3	1 2 4 3	3
4			
	1 2 3 4	1 2 3 4	4

Table 1. MAXSORT trace for permutation 4321

To apply MAXSORT as a heuristic for the 8-puzzle, we take N = 9. Let $K_{MAXSORT}(p)$ denote the number of swaps executed by MAXSORT given permutation p. Then given an 8-puzzle instance with arbitrary initial node p and goal node q = 123456789, the function $K_{MAXSORT}$ can serve as a heuristic for the 8-puzzle: we take $K(s) = K_{MAXSORT}(s)$, obtained for any state s by executing MAXSORT on s and counting the number of swaps it executes. For example if p = 296134758 and q = 123456789, then MAXSORT executes seven swaps when sorting p, hence K(p) = 7. (It happens that $h_{8\text{-puzzle}}(p,q) = 9$ in this case.) For other goal nodes q, we compute $K_{MAXSORT}(p, q)$ by simply using MAXSORT to sort p into q by first transforming the instance (p, q) into the canonical form (p', 123456789) and then sorting p' by MAXSORT. The permutation p' is the image of p under the transformation that maps q into 123456789, e.g., if p = 258491367 and q = 485297361, then 4 in p becomes 1 in p', 8 in p becomes 2 in p', and so on, so that p' = 4 32159786. Note that every swap must include as one of its elements not 9, but rather the image of 9 under this transformation, i.e., 5 in this example. (The trivial generalization of the MAXSORT code to accomplish this transformation is omitted here for brevity.)

Since 8-puzzle $\leq_e$ 9MAXSWAP it follows that $h_{9MAXSWAP}(s, t) < h_{8puzzle}(s, t)$. Hence if $K_{MAXSORT}(s, t) = h_{9MAXSWAP}(s, t)$ it follows from the A* admissibility theorem that solution paths found using $K_{MAXSORT}(s)$ as heuristic function for the 8-puzzle are

* If the number of edges in 9MAXSWAP were close to the number in the 8-puzzle we could infer that the distance function of the former is a close approximation of that of the latter, but such is not the case: The 8-puzzle, 9MAXSWAP, and 9SWAP each has 9! = 362,880 nodes. The 9MAXSWAP graph has 4•9! = 1,451,520 edges. (The number of edges in the N-MAXSWAP graph is N! (N-1)/2.) Nodes in the 8-puzzle graph are incident to 2, 3, or 4 edges, in the proportion 4:4:1, respectively. Hence the number of edges is 9!(2•4/9 + 3•4/9 + 4•1/9) / 2 = 9!•4/3 = 483,840, a factor of 3 fewer than in 9MAXSWAP. By comparison, the 9SWAP graph has 18•9! = 6,531,840 edges, a factor of 3.375 more than in 9MAXSWAP. (The number of edges in N-SWAP is N! N(N-1)/4.)

* The general behavior of MAXSORT for 4MAXSWAP can be depicted graphically as follows. Since the execution of MAXSORT for an arbitrary 4 element permutation corresponds to a path in the 4MAXSWAP graph (see Figure 2), the union over all 4-element permutation of the execution paths of MAXSORT (i.e., to sort into ascending order) is a subgraph of 4MAXSWAP. In fact this subgraph is a spanning tree of 4MAXSWAP (see Figure 3.)

of minimal length. * In words, to prove the premise is to show that for any 9-element permutations p and q, the number of swaps executed by MAXSORT in sorting p into q is minimal, i.e., it equals the minimum distance from p to q in the 9-MAXSWAP graph. Instead of attempting to prove this (its relevance beyond the example at hand being minimal), we note that it has been observed that heuristic functions that overestimate distance in a graph often are more efficient than similar functions that always underestimate the true distance ([Doran & Michie 1966], [Nilsson 1971], [Gaschnig 1977], [Gaschnig 1979]). Hence it may advantageous for efficiency in solving the 8-puzzle if $K_{MAXSORT}$ does not find minimal length paths in 9MAXSWAP, provided that a guarantee of minimal length solution paths in the 8-puzzle is not mandatory.

3.3. Experimental Performance Measurement Results

How efficient is the function $K_{MAXSORT}$ as a heuristic function for the 8-puzzle? To find out, we selected randomly a set of over 800 problem instances of the 8-puzzle, and solved each of these with A* using $K_{MAXSORT}$ as the heuristic function. As a measure of search efficiency, for each such problem instance (s_r, s_g) we measured the value of $X(s_r, s_g)$, the number of nodes expanded before a solution path is found to problem instance (s_r, s_g) when using $K_{MAXSORT}$ as a heuristic function with A*. This sample of problem instances is identical to the one used in [Gaschnig 1977] and [Gaschnig 1979], as is the experimental procedure followed, hence those results and the present results are comparable. The problem instances (s_r, s_g) in this sample are grouped together according to the value of $h(s_r, s_g)$, the length of a minimum length path in the graph from s_r to s_g.

Figure 4 plots the observed values of XMEAN(N), the mean number of nodes expanded as a function of depth of the goal node (i.e., the mean over all instances (s_r, s_g) in the sample such that $h(s_r, s_g) = N$). Superimposed in Figure 4 for comparison purposes are the analogous experimental data reported in Figure 1 of [Gaschnig 1977] and [Gaschnig 1979] for three particular 8-puzzle heuristic functions there called K_1, K_2, and K_3. Figure 4 shows that $K_{MAXSORT}$ is slightly better than $K_1(s)$, which computes the number of tiles out of place in tile configuration s with respect to the goal node. Figure 4 shows also that $K_{MAXSORT}$ usually expands many more nodes than do K_2 or K_3.** So it turns out that $K_{MAXSORT}$ is a relatively inefficient heuristic for the 8-puzzle, compared with these other known heuristic functions, but inspection of Figure 4 reveals that it expands far fewer nodes in solving the 8-puzzle than does breadth-first search.

* As a matter of convenience, we sometimes write K(s) instead of K(s, t) when t is understood implicitly. For example, in any given A* search node t is understood to be the goal node.

** K_2 computes the number of tiles out of place, each weighted by the orthogonal distance the tile must move to its desired spot, assuming no other tiles block the path; K_3 equals K_2 plus another term measuring the degree to which the outer tiles in s agree in rotational order to those in the goal node. See [Doran & Michie 1966], [Nilsson 1971], [Gaschnig 1977], or [Gaschnig 1979] for additional details.

4. Analysis of Distance Function Approximation

The experimental results in Figure 4 pose a challenge: why is $K_{MAXSORT}$ a relatively inefficient heuristic for the 8-puzzle, and is it coincidental that its performance is comparable to that of K_1? One approach to answering such questions is to determine how closely $K_{MAXSORT}(s, t)$ approximates $h_{8\text{-puzzle}}(s, t)$. Figure 5 plots experimental measurements of the range of $K_{MAXSORT}$'s estimates of distance to the goal node in the 8-puzzle vs. the actual distance to the goal. KMIN(i) is defined with respect to $K_{MAXSORT}$ to be the minimum value of $K_{MAXSORT}(s, t)$ over all node pairs (s, t) such that h(s, t) = i, and similarly KMAX(i) is the maximum of these values. Hence KMIN(i) and KMAX(i) bound the values computed by $K_{MAXSORT}$ for nodes that happen to be distance i from the goal. The data plotted in Figure 5 were recorded during the same experiment represented in Figure 4, in the following way. The value of $K_{MAXSORT}(s_r, s_g)$ was recorded for each problem instance (s_r, s_g) in the sample set. The value of $h(s_r, s_g)$ equals the length of the solution path found, which is of minimum length since $K_{MAXSORT}$ underestimates distance in the 8-puzzle graph. In addition we measured the value of $K_{MAXSORT}(s, s_g)$ for each node s along the solution path found for each problem instance in the sample set. Since the solution path found is of minimum length, the value of $h(s, s_g)$ is known for each such node s. In all, Figure 5 represents over 10,000 distinct observations of $K_{MAXSORT}(s, t)$ vs. $h_{8\text{-puzzle}}(s, t)$. Superimposed for comparison in Figure 5 are analogous experimental measurements of KMIN(i) and KMAX(i) for K_1, taken from Figure 2 in [Gaschnig 1977] or [Gaschnig 1979].

Figure 5 shows both that $K_{MAXSORT}$ is a poor approximation of $h_{8\text{-puzzle}}$, and that the approximation is only slightly better than that of K_1.

5. Discussion

We have described a general principle of problem similarity for path-finding state-space problems -- the edge subgraph and edge supergraph relations. We have demonstrated, using the 8-puzzle as case study, how this principle of "edge subgraph/supergraph transfer" can be exploited to aid in devising a heuristic function for a given problem, by transferring a heuristic from an "easier" problem to a related "harder" one. We have measured experimentally the efficiency of the resulting heuristic function devised for the 8-puzzle, demonstrating at least a limited practical utility.

These first results leave many questions unanswered. Our intention here has been merely to introduce the idea for further consideration. Hence the remainder of this discussion focusses on extensions to the present efforts.

Additional insight might be obtained by applying the transfer approach to additional problem graphs, including some in which the transfer problem is an edge subgraph of the given problem instead of an edge supergraph, as in the 8-puzzle case study. In particular, we note of the 8-puzzle case study that a transfer problem (9MAXSWAP) was readily apparent, that an efficient algorithm for 9MAXSWAP was readily devised, and that this algorithm proved to be a relatively inefficient heuristic function for the 8-puzzle. How formidable these hurdles are for other problems remains to be determined.

Another objective is to attempt to devise for the 8-puzzle another transfer problem which is a closer approximation to it than is the 9MAXSWAP graph, that is to identify a problem graph that is an edge supergraph of the 8-puzzle and an edge subgraph of 9MAXSWAP.

It may be interesting to apply this transfer concept in reverse fashion: given a particular heuristic function for a given problem, identify the graph to which it is equivalent. For example, one might attempt to determine whether a graph corresponding to the K_1 function is an edge supergraph of the 9MAXSWAP graph, or to identify a graph corresponding to the function K_2.* A theory about the equivalence of problem graphs and heuristic functions along these lines is conceivable.

Acknowledgement. Richard Korf offered helpful comments on an earlier draft of this paper.

6. References

Doran, J., "An Approach to Automatic Problem Solving," in Machine Intelligence 1, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.

Doran, J., "New Developments of the Graph Traverser," in Machine Intelligence 2, E. Dale and D. Michie (eds.), American Elsevier Publ. Co., New York 1968.

Doran, J., and D. Michie, "Experiments with the Graph Traverser Program," Proc. Royal Society of London, Series A, Vol. 294, 1966, pp. 235-259.

Ernst, G.W., Banerji, R.B., Hookway, R.J., Oyen, R.A., and Shaffer, D.E., "Mechanical Discovery of Certain Heuristics", Report 1136-A, Jennings Computing Center, Case Western Reserve Univ., Cleveland, Ohio, January 1974.

Gaschnig, J., "Exactly How Good are Heuristics?: Toward a Realistic Predictive Theory of Best First Search", Proc. Intl. Joint Conf. on Artificial Intelligence, Cambridge, Mass., August 1977, pp. 434-441.

Gaschnig, J., Performance Measurement and Analysis of Certain Search Algorithms, Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., May 1979.

Gelperin, D., "On the Optimality of A*," Artificial Intelligence, Vol. 8, 1977, pp. 69-76.

Harris, L., "Heuristic SEarch under Conditions of Error," Artificial Intelligence, Vol. 5, No. 3, North Holland Publ. Co., Amsterdam, 1974, pp. 217-234.

Hart, P., N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans. Sys. Sci. Cybernetics, Vol. 4, No. 2, 1968.

Ibaraki, T., "Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms," International Journal of Computer and Information Sciences, Vol. 5, No. 4, December 1976, pp. 315-344.

Jac[illegible]n, P. Introduction to Artificial Intelligence, Petrocelli Books, New York 1974.

Martelli, A., "On the Complexity of Admissible Search Algorithms," Artificial Intelligence, Vol. 8, 1977, pp. 1-13.

Munyer, J., "Some Results on the Complexity of Heuristic Search in Graphs," Technical report HP-76-2, Information Sciences Dept., U. Cal. Santa Cruz, September 1976.

Munyer, J., and I. Pohl, "Adversary Arguments for the Analysis of Heuristic Search in General Graphs," Technical report HP-76-1, Information Sciences Dept., U. Cal. Santa Cruz, July 1976.

Nilsson, N., Problem Solving Methods in Artificial Intelligence, 1971.

Pohl, I., "First Results on the Effect of Error in Heuristic Search," Machine Intelligence 5, B. Meltzer and D. Michie (eds.), Edinburgh University Press, Edinburgh 1970 (1970a).

Pohl, I., "Heuristic Search Viewed as Path-Finding in a Graph", Artificial Intelligence, Vol. 1, 1970 (1970b).

Pohl, I., "Practical and Theoretical Consideration in Heuristic Search Algorithms," in Machine Intelligence 8, E. Elcock and D. Michie (eds.), Ellis Howard Ltd., Chichester, England 1977.

Polya, G., How to Solve It, Princeton University Press, Princeton, N. J. 1945.

Raphael, B., The Thinking Computer: Mind Inside Matter, W. H. Freeman & Co., San Francisco 1976.

Rendell, L., "A Method for Automatic Generation of Heuristics for State-Space Problems", Report CS-76-10, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, February 1976.

Rendell, L., "A Locally Optimal Solution of the Fifteen Puzzle Produced by an Automatic Evaluation Function Generator," Report CS-77-36, Dept. of Computer Science, University of Waterloo, December 1977.

Samuel, A., "Some Studies in Machine Learning Using the Game of Checkers," in E. Feigenbaum and J. Feldman (eds.), Computers and Thought, McGraw-Hill 1963, pp. 71-105.

Samuel, A., "Some Studies in Machine Learning Using the Game of Checkers II. Recent Progress", IBM J. Research and Development, Vol. 11, No. 6, (1967), pp. 601-617.

Schofield, P. "Complete Solution of the Eight Puzzle," in Machine Intelligence 1, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.

Slagle, J., and Farrell, C., "Experiments in Automatic Learning for a Multipurpose Heuristic Program," Communications A.C.M., Vol. 14, No. 2, pp. 91-99.

Swinehart, D., and B. Sproull, SAIL, Stanford Artificial Intelligence Laboratory Operating Note 57.2, January 1971.

Vanderbrug, G., "Problem Representations and Formal Properties of Heuristic Search," Information Science, Vol. 11, No. 4, 1976.

Wickelgren, W., How to Solve Problems, W. H. Freeman and Co., San Francisco 1974.

*Given a heuristic function K for a given problem graph P1, one can define another graph P2 thus: the nodes in P2 correspond to the nodes in P1; an edge connects two nodes s and t in P2 iff K(s, t) = 1. Whether or not P2 satisfies the conditions of problem graph depends on the value computed by K.

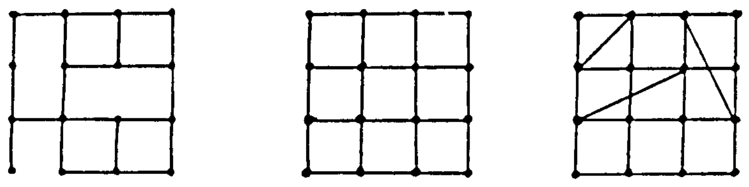

(a) MSUB44 (b) M44 (c) MSUP44

Figure 1. A "Manhattan street pattern" graph and two variants thereof

Note that MSUB44 $\leq_t$ M44 $\leq_t$ MSUP44

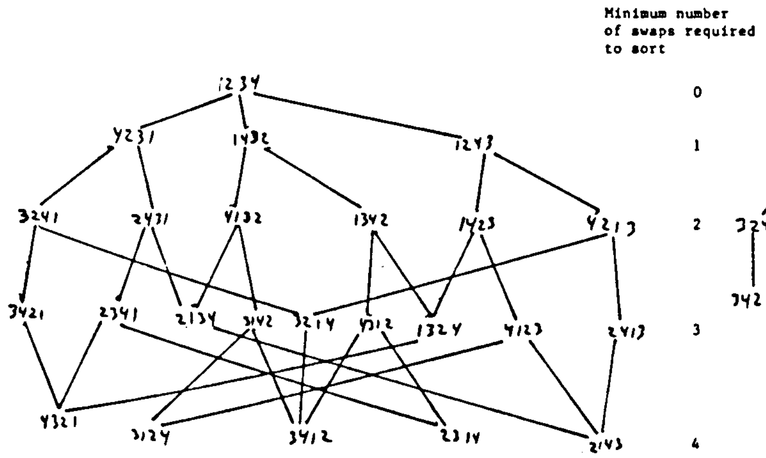


Figure 2 4MAXSWAP graph

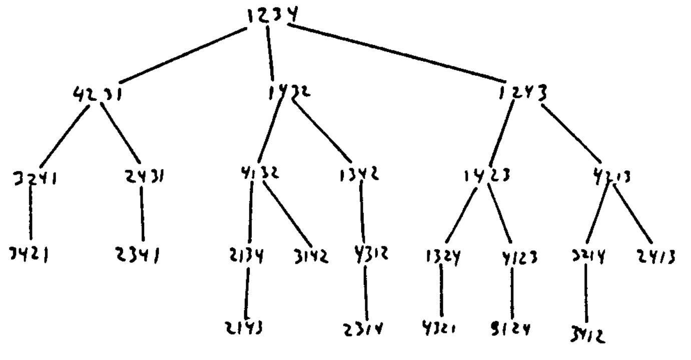


Figure 3. The union of the paths taken by MAXSORT in sorting any 4-element permutation is a spanning tree of 4MAXSWAP (and hence an edge subgraph of 4MAXSWAP; compare with Figure 2)

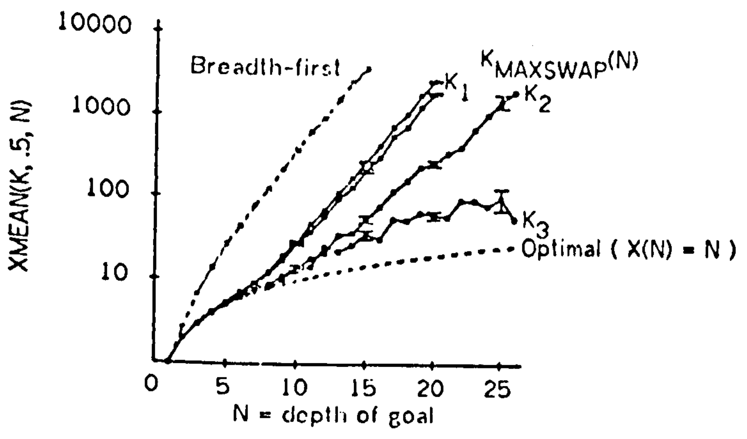


Figure 4 Mean number of nodes expanded vs. depth of goal
A* search of the 8-puzzle, comparing heuristic K_{MAXSAP} and $K_{WMAXSWAP}$ with heuristics K_1, K_2, and K_3
40 problem instances for most values of N
760 to 895 algorithm executions per heuristic function

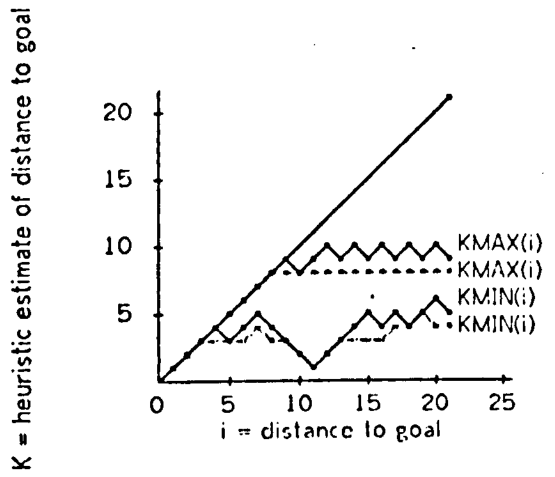


Figure 5
Heuristic estimate of distance to goal vs. actual distance
8-puzzle heuristic $K_{MAXSWAP}$ (solid) compared with heuristic K_1 (dash)