Generation of Heuristics by Problem Transformation

Dennis Kibler

University of California at Irvine

Information and Computer Science Department

14 December 1983

Technical Report # 225

This research was supported by the Naval Ocean Systems Center under contract N00123-81-C-1165.

Abstract

We define problem transformations and show that each problem transformation induces an admissible and monotonic heuristic on the original problem. Furthermore we show that every admissible and monotonic heuristic is induced by some problem transformation. This result generalizes and unifies several approaches for heuristic formation reported on in the literature. We give four techniques for generating problem transformations and we apply these techniques to generate several heuristics found in the literature. We also introduce a variant of the relational representation framework which has some advantages.

Keywords: Heuristic generation, admissible heuristics, problem transformation, quotient spaces, problem representation.

CR CATEGORIES: I.2.8, I.2.6.

Introduction

If a straightforward attack on a problem leads to failure, Polya [14] suggested that one should consider simpler, analogous problems. Once these associated problems were solved, the insights gained were somehow to be translated back to apply to the original problem. Gaschnig [4] was the first to propose using an analogous problem to generate a heuristic for the original problem. Gaschnig chose to represent problems as graphs and analogous problems (which he called transfer problems) as supergraphs or subgraphs. This paper presents a general method for generating associated problems and the transformation between problem and associated problem. We show how

these associated problems lead to admissible and monotonic heuristics for the original problem. Special cases of these techniques have been studied by Gaschnig [4], Pearl [12], Kibler [7], and Valtorta [16].

Some advantages of changing the representation of a problem have been noted in the literature. Newell and Simon [9] outlined their "planning method" which consisted of abstracting essential relations or operators. Amarel [1] showed how to transform the representation of a generalized missionaries and cannibals problem. Korf [8] defined a language for representing representations and transformations. Using this framework he shows how large representational changes can be achieved by composing transformations which have small effects. Unfortunately he gives little guidance for selecting the appropriate transformations. In the latter papers a problem is transformed and a solution to the transformed problem provides a solution to the original problem. In our work, the transformed problem guides the solution process of the original problem by providing an admissible and monotonic heuristic.

We begin by formally defining state-spaces, state-space transformations, problems and problem transformations. Then we prove that each problem transformation induces an admissible and monotonic heuristic on the original problem. Furthermore we show that every such heuristic can be achieved by problem transformation. Then, for particular choices of problem representation, we define four types of problem transformations, namely inclusion maps, relaxation maps, quotient maps, and counting maps. We apply these techniques to the

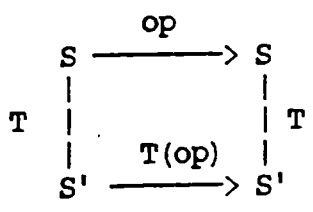
eight-tile puzzle, the cube slicing problem, the blocks world, and the mutilated checkerboard problem. Lastly we discuss the limitations and weaknesses of the approach.

Problems and Problem Transformations

The following definitions fit within the framework that Nilsson [11] provided for the A^* algorithm. They are also similar to those proposed by Georgeff [5], although he mapped states and strategies (as represented by program schemas and their interpretations) while we map states and operators.

A collection of states S and operators Op is called a state-space if each operator in Op is a partial function[1] from S to S. A cost function c on a state-space <S,Op> is a positive real-valued function on Op. If no cost function is explicit then, by default, the state-space has the cost function which assigns one to each operator.

A state-space transformation (without cost) T from one state-space into another is a mapping of states S and operators Op of the source state-space into the states S' and operators Op' of the image state-space such that the following diagram commutes:



To say the diagram commutes means that the following equation holds, for

---

[1] A partial function f from X to Y is function from a non-empty subset of X into Y. f may be many-to-one, one-to-one, into, or onto.

all states s and applicable operators op:

apply(T(op),T(s))=T(apply(op,s)) (equation 1)

where apply(op,s) is the new state arrived at by applying op to the state s. This definition is standard in the mathematical subarea of category theory.

A state-space transformation T <u>with cost</u> of <S,Op,c> into <S',Op',c'> is a state-space transformation T of <S,Op> into <S',Op'> such that $c'(T(op)) \leq c(op)$ for all op in Op. If T is a state-space transformation of <S,Op> into <S',Op'> and c is a cost function on <S,Op> then define the <u>induced</u> cost function on Op' by:

if op' is in T(Op) then c'(op')=minumum{c(op) where T(op)=op'}
else c'(op') is any positive value.

The above definition is necessary since several operators in the source domain may map to the same operator in the image domain. Clearly if T is a state-space transformation of <S,Op> into <S',Op'> and c is cost function on <S,Op> then T is a state-space transformation with cost from <S,Op,c> into <S,Op,c'> where c' is the induced cost function.

A <u>problem</u> P is a five-tuple <I,S,F,Op,c> where <S,Op> is a state-space, I is an element of S called the <u>initial</u> state, F is a subset of S called the <u>final</u> states or <u>goal</u> states, and c the cost function. A sequence of operators $\langle op_1, op_2, \ldots op_n \rangle$ applies to a state s if the image of op_i is contained in the domain of op_{i+1}. Such a sequence is a solution to the problem if the image of the last operator belongs to F. An <u>optimum</u> solution is one whose cost is minimum over all solutions, where the cost of a solution is the sum of the cost of each

operator in the sequence. With the default cost function the minimum cost solution is the one with minimum length.

If P=<I,S,F,Op,c> is a problem (called the source problem) and T is a state-space transformation of <S,Op> into <S',Op'>, we define the image problem P'=<I',S',F',Op',c'> by the following means. Let I' be T(I) and F' be {T(f) where f belongs to F}. Let c' be the induced cost function.

Given the above definition the following observations are immediately apparent.

1. A solution of the source problem maps into a solution of the image problem.
2. If the image problem is unsolvable the source problem is unsolvable.
3. The cost(length) of a solution in the image state-space is not greater than the cost(length) of a solution in the source state-space. (This result depends on the definition of c'.)
4. The minimum cost solution in the image state-space is a lower bound on the cost of a solution in the source state-space.
5. The composition of two state-space transformations is a state-space transformation. The composition of two problem transformations is a problem transformation.

Nilsson [11] defined an admissible search algorithm as one which is guaranteed to find the optimum solution, whenever a solution exists. A (numeric) heuristic is non-negative function on the set of states. We say a heuristic h is admissible if it underestimates the cost from the node to the goal. Nilsson proved that the evaluation function $f(n)=g(n)+h(n)$, where $g(n)$ is the current cost to the node n, and h is admissible, defines an admissible algorithm. A heuristic is monotonic if for any node n and operator op, $h(n) \leq h(n')+c(op)$ where n' is the node

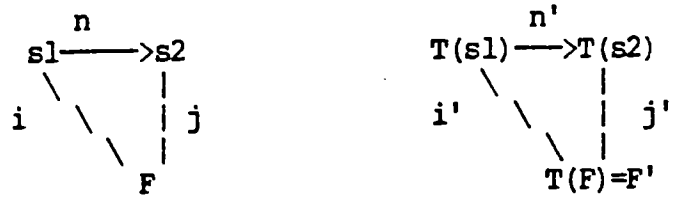
reached by applying op to n. Monotone heuristics are valuable because they reduce the updating required in the A^* algorithm. More specifically, applying the A^* algorithm to a problem sometimes requires updating the g-value (current cost of reaching node) of a CLOSED node. Whenever the g-value of a node is updated, one must check whether any descendant of the node requires updating. This is an expensive operation within the A^* algorithm. The advantage of monotonic heuristics is that they eliminate the need for updating the g-values of CLOSED nodes.

We need one more definition to simplify the statement of the next theorem. If T is a problem transformation from P=<I,S,F,Op,c> to problem P'=<I',S',F',Op',c'> then the function h defined on S by h(s)= cost of optimum solution of T(s) in P' is the <u>induced</u> heuristic.

The following theorem shows how to induce monotonic and admissible heuristics from problem transformations.

Theorem 1: If T is a problem transformation from problem P to problem P' then the induced heuristic is admissible and monotonic heuristic.

Proof: The admissibility of h follows immediately from result 4. To show monotonicity consider the following diagrams:



where s1 and s2 are states of S, n is the cost of the operator from s1 to s2, i is h(s1), j is h(s2), and i', j', and n' are the minimum costs between states in the image state-space. We must show that $n+j \geq i$. By

definition i=i' and j=j'. Also n'+j'≥i' by definition. Since n≥n', we have n+j≥i, and monotonicity is proven. Q.E.D.

We call the heuristic generated by process described in the theorem above the induced heuristic.

The next theorem shows that every admissible and monotonic heuristic can be induced by a problem transformation. This gives a precise understanding of the power of problem transformations. Gaschnig [4] hypothesized a similar version of the following theorem.

Theorem 2: Any admissible and monotonic heuristic on a problem can be induced by problem transformation.

Proof: Let h be an admissible and monotonic heuristic on a problem P=<I,S,F,Op,c>. We will define a problem P'=<I',S',F',Op,c'> and problem transformation T of P into P' such that the induced heuristic h' equals h.

Let I'=I and S'=S. For each state s in S, T(s)=s. For each s in S define op_s as the partial function from {s} to f where f is any element of F, the final states. Let Op'=Op#U#{op_s where s belongs to S}. Intuitively we have added a short-cut, one-step solution in the image domain. For each op in Op, let T(op)=op. Obviously T is a domain transformation. Let c' be the cost function defined by:

if op is in Op, then c'(op)=c(op).
if s is in S, then c'(op_s)=h(s).

With this definition of c', T is clearly a domain transformation with cost. Consequently there is an induced heuristic h' on P.

We now prove that h' is identical to h. Recall that, for s in S,

h'(s) is the minimum cost of solution of T(s) in S'. It suffices to show that a minimum cost solution is found by applying the single operator op_s (whose cost is h(s)) to s.

The proof breaks into two cases. Let s belong to S and let $\langle op_1, op_2, \ldots op_n \rangle$ be the sequence of operators of Op' that yield a minimum cost solution in S'.

Case 1: n=1

If n=1 then op_1 is either in Op or is op_s, in which case there is nothing to show. If op_1 is in Op then $h(s) \leq c(op)$ since h is admissible. But then op_s is at least as cheap a solution as op_1.

Case 2: n>1

If n>1 then op_1 must belong to Op. But now the monotonicity of h demands that $h(s) \leq c(op_1) + h(op_1(s))$. The admissibility of h implies $h(op_1(s)) \leq$ cost of solution from $op_1(s)$. Hence op_1 is again as cheap a solution.

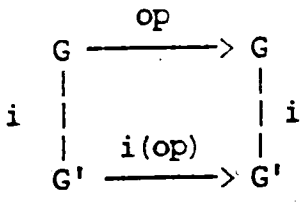
In either case a minimum cost solution in S' is achieved by applying op_s whose cost is h(s). Therefore h'(s), the induced heuristics, is h(s). Q.E.D

The weakness in the above theorem is that it does not tell us how to construct useful problem transformations. We now consider some specific problem representation schemes and give some general techniques for constructing problem transformations.

GENERAL STATE-SPACE TRANSFORMATIONS

Graphical Representation

Gaschnig [4] defined a problem graph as a finite, strongly connected graph with no multiple edges or self-loops. In this context a problem is defined by a start node s and a final node f from G. Gaschnig defined a transfer problem as one which is either a subgraph or supergraph. He noted that a subgraph transfer problem need not induce an admissible heuristic but that a supergraph transfer problem will induce an admissible heuristic. The result for supergraph transfer problem is a special case of theorem 1, which also concludes that the heuristic is monotonic - a point not noted by Gaschnig. This is probably the simplest application of theorem 1 for it follows once we verify that the following diagram commutes, where G is the set of nodes of the graph and an operator is an edge:



where i is the inclusion[2] map of states(nodes) and operators(edges) into nodes and edges of the contained graph. Let us call this type of state-space transformation an inclusion map. Gaschnig applies this technique to the 8-tile puzzle by embedding the graph in a subset of the graph that allows sorting by swapping a pair of values. This does not lead to a very good heuristic possibly because it is not intrinsic to

---

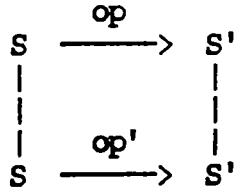
[2]An inclusion function i from X to Y is defined by i(x)=x.

the problem in the sense that no direction is given for choosing the "right" supergraph. For a graph with n nodes there are about 2^{n*n} supergraphs with the same nodes. In particular for the 8-puzzle there are around 180,000 (9!/2) nodes in the graph. Therefore the number of supergraphs (with the same nodes) is roughly $10^{10,000,000}$. Finding the right supergraph is a gargantuan task.

We now consider state-space transformations which are intrinsic to the problem, i.e. they are constructed from the original problem.

Relational Representation

Perhaps the most widely used representational scheme is that described by Fikes and Nilsson [2] in their STRIPS system. In this approach each operator is defined by three lists of relations, namely the preconditions list, the delete list, and the add list. In a later work Fikes, Hart and Nilsson [3] showed how to build a hierarchy of abstraction spaces by disregarding some preconditions of each operator. The solution in the abstraction space provided a skeleton solution, to be elaborated upon in the original space. Using the same representation scheme, Pearl [13] showed that one could generate admissible and monotonic heuristics. If we define the "forgetful" transformation which is the identity on states but forgets some of the preconditions of operators, we get a state-space transformation, as is readily apparent. All that need be shown is that the following diagram commutes:



where op' is formed by removing some of the preconditions of op.

Following Pearl we call such transformations, relaxation maps. A simple application of theorem 1 is that relaxation maps induce monotonic and admissible heuristics.

Now we will consider a different state-space representation scheme which seems to have some advantages over the standard STRIPS-like approach.

Multiset Representation

In this section we first describe the multiset representation and then define two types of problem transformations.

We use multisets or bags[3] of relations rather than sets of relations to describe the states and the operators. This representation has been used by Kibler and Morris [6] to analyze and repair plan inefficiencies. In particular we define a relational operator by specifying two multisets of conditions, one called the preconditions and the other the postconditions. An operator is applicable in a given state if each of its preconditions is satisfied in the state. The preconditions of an operator are satisfied if there exists a substitution for the variables such that the instantiated preconditions are contained, as a multiset, in the state description. One applies an applicable operator by deleting each of the instantiated preconditions from the state and adding each of the instantiated postconditions.

The multiset approach avoids the problem, discussed by Vere [17],

---

[3] A bag can be represented as a list in which elements may appear more than once.

of composing relational operators to form a "macro-operator". Vere showed that, in general, relational operators do not have a well defined composition and that one must augment their descriptions in order that composition would make sense. Multiset relational operators have a well-defined composition, in the same sense as presented by Vere, but we forgo the demonstration as it is not needed for the rest of this paper.

A multiset representation allows a simpler and more natural description of operators. For example, in the blocks world with the common definition of relational operator as used by Vere or Nilsson, the definition of the operator move(x,y,z) (meaning move x from y onto z) has the preconditions: clear(x),clear(z),on(x,y),x=/=z. Using multisets the precondition x=/=z is superfluous since x and z cannot be bound to the same object. Intuitively preconditions are resources rather than assertions of truth.

Quotient maps

A simple useful class of state-space transformations are quotient maps. These are very similar in effect to relaxation maps but do not entail the additional overhead of storing unneeded relations. A quotient map simply forgets about some of the relations in the source state-space, for both operators and states. More precisely a quotient map Q is defined by a set S of relations as follows:

Q(state)={n| member(n,state) and not member(n,S)}
Q(operator)=newop where
preconditions(newop)=Q(precondition(op)) and
postcondition(newop)=Q(postcondition(op)).

Again to show that the map is a state-space transformation one need only check that the appropriate diagram commutes. Notice that if the operators and states can be described with n relations, then the number

of quotient maps is 2^n. This includes two degenerate cases. One where the quotient set is empty so we reachieve the original problem and the other where the quotient set consists of all the relations so that the initial state and the goal state become equivalent.

Counting maps

Another useful class of natural transformations are counting maps. Roughly a counting map replaces a collection of relations by the number of instances of each type of relation. Intuitively, quotient maps forget about some relations, while counting maps forget about particular objects. More precisely, if the state-space is described by n relations, say r_1, r_2, through r_n, then the counting map will map each state into an n-vector of integers, where the integer of the i^{th} component corresponds to the number of relations of type r_i in the state. Equation 1 determines the definition of the image of an operator. Again this map is clearly a state-space tranformation. Note that counting maps would not fit into the relational framework for operators (without some violence).

How can we know which state-space transformations will be useful? This is the most important unanswered research question. The following gives a mild test to guarantee that the image problem will be simple to solve. A collection of operators Op is commutative if for any applicable sequence $\langle op_1, op_2 \rangle$, $\langle op_2, op_1 \rangle$ is applicable and yields the same result. Note that the image operators under a counting map are commutative. This guarantees that the problem will be easy to solve for it implies one can solve the goals of image problem in any order. A problem in which the goals can be solved in any order has been called

decomposable by Pearl [13].

For convenience we denote the number of occurrences of a relation or the multiplicity of the relation by writing multiplicity*relation. Consequently when we write the preconditions of a counting operator as $n*r_1$, $m*r_2$, etc. we mean that the operator deletes n relations of type r_1, m relations of type r_2, etc. Postconditions and states are denoted similarly.

Notice that state-space transformations have the effect of focusing attention on some aspect of the problem. Focusing on the wrong aspect of a problem leads to no insight. Focusing on the right aspect can lead to appropriate heuristics or a quick proof that the problem is unsolvable.

EXAMPLES

We illustrate the power of heuristic formation from problem transformation in the following examples. In all of these examples we use the multiset representation to demonstrate some of its advantages.

Tile puzzle

Nilsson analyzed the eight-tile puzzle and defined a number of heuristics for its solution [10]. This puzzle can be defined as a relational production in the following way. Let the board size be 3X3 and label the positions as in the diagram below.

position labels	goal state
a b c	1 2 3
d e f	4 5
g h i	6 7 8

A state, such as the goal state depicted above, could be described as:

```
pos(a),pos(b),...,adj(a,b),adj(b,a),adj(a,d),...
on(a,1),on(b,2),on(c,3),on(d,4),blank(e),on(f,5),
on(g,6),on(h,7),on(i,8).
```

where pos stands for position and adj stands for adjacent. There is only one relational operator, defined by:

```
preconditions: adj(X,Y),pos(X),pos(Y),blank(Y),on(X,V)
postconditions: adj(X,Y),pos(X),pos(Y),blank(X),on(Y,V).
```

A number of different quotient spaces can be constructed from this problem. If we demand that the quotient space maintain the "on" relation, then there are seven (2^3-1) candidate quotient spaces. We will look at three of them.

Heuristic P(n)

If we forget about the requirement that a tile be blank, then we get a new problem state-space with an operator defined by:

```
preconditions: adj(X,Y),pos(X),pos(Y),on(X,V)
postconditions: adj(X,Y),pos(X),pos(Y),on(Y,V).
```

This corresponds to a tile-puzzle where you are allowed to pile up tiles. In this state-space it is easy to reason that the minimum number of moves to change from one state to another is exactly the sum of the city-block distances between each tile's current position and its destination. This is exactly heuristic P(n) of Nilsson [10].

Heuristic W(n)

If we forget about both the blank constraint and the adjacency constraint we get a new problem state-space with an operator whose preconditions are pos(V),pos(X),on(X,Y) and whose postconditions are pos(X),pos(V),on(X,V). This corresponds to a tile-puzzle where you are allowed to pick up any tile and move it to any desired position. In

this puzzle, the minimum distance between two states is the number of tiles out of position. Hence the distance of the minimum solution for this quotient problem defines the admissible heuristic W(n) [10].

Notice that the operators in the above two quotient spaces had the property that they were commutative, allowing us to easily solve the image problem.

Heuristic almost W(n)

If we forget about the adjacency and position constraint we get a new operator with preconditions: blank(Y),on(X,V) and postconditions: blank(X),on(Y,V). This corresponds to allowing a tile to be moved into a blank position regardless of its position. The resulting heuristic is closely related to W(n).

We note that the total number of induced heuristics using quotient maps is 14 (2^4-2). If we compose this with counting maps we get another 14 potential heuristics. As we have already noted, the graphical technique gave a space of heuristics with about $10^{10,000,000}$ elements.

Pearl [13] used relaxation maps to achieve results similar to the ones above.

Another heuristic defined by Nilsson is defined to be P(n)+3*S(n) where S(n) measures the "sequence score" of tile position [10]. Although a better heuristic than either W(n) or P(n) it is not admissible and so cannot be found by any state-space transformation. Georgeff [5] has a strategy for solving this puzzle which generates fewer nodes than the above heuristic, but finds a longer solution. The search space of potential strategies seems to be infinite.

Cube Slicing

Since admissible heuristics provide lower bounds on the number of operations required to fulfill a goal, they can be used to determine that some problems have no solution. An instance of such a problem is the question of whether a cube can be sliced into 27 equal cubelettes with only five slices. By insight one sees that it is necessary to slice each of the six faces of the inner cube, so the problem is impossible.

This conclusion can also be reached by applying state-space transformations. A detailed relational description of the original problem would be tedious. We give only the image state-space and the corresponding operators. By applying quotient maps and counting functions, one reduces the problem to one whose initial state is {block<27>}, where the 27 refers to the volume in terms of smaller cubes. The goal state is {27*block<1>}. The slicing operations generate a collection of relational operations. Each relational operator deletes a set of blocks and replaces each block of the subset by two blocks, where the new blocks are of equal size or one is twice as large as the other. Some of the relational operators are:

op1: delete conditions: {block<9>,block<6>}
add conditions: {3*block<3>,block<6>}

op2: delete conditions: {block<9>,block<6>}
add conditions: {block<3>,block<6>,block<2>,block<4>}.

Without worrying about the entire search tree that would be generated, we can reason as follows. If at each slicing the largest block is subdivided then following only the largest block at each slice would

give the following chain.

block<27>->block<18>->block<9>->block<6>->
->block<3>->block<2>->block<1>.

This chain has six operations in it so the original problem requires at least six operations. This example indicates one of the weaknesses with this approach. It requires that a person see a computationally easy way of solving the transformed problem or, equivalently, of computing h.

Blocks World

We will only sketch the application of state-space transformations to the blocks world. Using any of the usual relational descriptions of the blocks world [11], one can define a quotient map which forgets about all relations but the "on" relations. The admissible heuristic generated is the number of different "on" instances between the current state and the goal state. Such a heuristic has the effect of making the search somewhat similar to that generated by STRIPS [2], in that the search is directed towards achieving the "on" goals sequentially for each possible ordering of the goals. However, the admissible heuristic approach guarantees finding a solution. We note that goal ordering fails for STRIPS [15] because the goals only partially specify the goal state. If the goals had been "completed" or extended to give a full specification of the final state, then goal ordering would allow STRIPS to solve all blocks worlds problems easily. In particular if we add the goals so all the desired "on" conditions form stacks starting at the table, then ordering the goals from the table to the top of the stack will allow a straightforward solution.

Mutilated checker board

The problem is: if a checkerboard has the white corners removed, can it be covered by dominoes, where each domino covers two adjacent squares. We will use a state-space transformation to show that the problem is impossible. Below is an abbreviated relational description of the initial state, the single operator, and the goal state:

```
initial state: adj(a,b),....
               white(a),...
               red(b),...
               free(a),free(b),...
operator: Pre: adj(X,Y),red(X),white(Y),free(X),free(Y).
          Post: adj(X,Y),red(X),white(Y),covered(X),covered(Y).
goal: covered(a),covered(b),...
```

We define a state-space tranformation into states described simply by the number of free white(fw) and free red(fr) squares. Consequently we have:

```
image initial state: 30*fw,32*fr.
image operator:  preconditions: fw,fr.   {deleted relations}
                 postconditions: empty. {added relations}
image goal:        empty. {i.e. (0*fw,0*fr)}
```

It is easy to check that this mapping is a state-space transformation. Moreover in the new space only one operator is applicable to any state, so one quickly sees that there is no way to reach the state {0*fw,0*fr}. Counting maps yield simple image problems so they can serve as a quick check for possibility.

DISCUSSION

Limitations and Weaknesses

This approach always generates admissible heuristics, but not all admissible heuristics are useful. Applied to the tower-of-hanoi problem, the approach yields no useful heuristic. The quotient map may

fail to help because it either trivializes the problem or does not reduce the difficulty sufficiently so that the calculation of the minimum cost solution is simple. There are two major problems to overcome before a successful implementation can be achieved. First there is no guidance as to which state-space transformation to try. Pearl [13] has described several qualities that would make the image problem easy to solve, but has not given methods for creating such problems. The open research question is how to determine which state-space transformations are useful without searching a large space of transformations. This problem might be amenable to both analytical or heuristic approaches. Possibly from an analysis of the operators one could determine which relations need to be factored out so that the resulting quotient problem would be easy (perhaps too easy) to solve. The second problem is that the technique requires a person to optimize the computation of h. This difficulty is unavoidable according to the results of Valtorta [16].

Conclusions

We have shown that state-space transformations induce admissible and monotonic heuristics. Moreover every admissible and monotonic heuristic can be induced by an appropriate problem transformation. Since state-space transformations are composable this technique yields a large set of heuristics. These heuristics can be used to guide the search for a solution or to demonstrate the impossibility of finding a solution. Moreover, four types of state-space transformations, namely inclusion maps, relation maps, quotient maps, and counting maps, were defined and shown to be equivalent to special cases found in the

literature. The latter three maps are intrinsic to the original problem, i.e. they do not require a "eureka" form of insight to construct. While not leading to as efficient search strategies as those proposed by Georgeff, the automatic generation of admissible heuristics is better developed than the automatic discovery of strategies. Finally, by using quotient transformations we generated a number of standard heuristics. In addition, by composing quotient and counting maps we proved the impossibility of solving some problems.

REFERENCES

1. Amarel, S. On the representations of problems of reasoning about actions. Machine Intelligence III, 1968.

2. Fikes, R.E., and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. AI 2 (1971), 189-208.

3. Fikes, R.E., Hart, P.E., and Nilsson, N.J. Learning and Executing Generalized Robot Plans. AI 3, 4 (1971), 251-288.

4. Gaschnig, J. A problem similariy approach to devising heuristics. Proceedings of IJCAI 6 (1979), 301-307.

5. M.P. Georgeff. Strategies in Heuristic Search. AI 20, 4 (1983), 393-426.

6. Kibler, D.F, and Morris, P.H. Plan Variants and a Plan Refinement Algorithm. 178a, University of California, Irvine, 1982.

7. Kibler, D.F. Natural Generation of Admissible Heuristics. 188, University of California, Irvine, 1982.

8. Korf, R.E. Toward a Model of Representation Changes. AI 14 (1980), 41-78.

9. Newell, A. and Simon, H. Human Problem Solving. Prentice-Hall, 1972.

10. Nilsson, N.J. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, 1971.

11. Nilsson, N.J. Principles of Artificial Intelligence. Tioga, 1980.

12. Pearl, J. On the Discovery and Generation of Certain Heuristics. UCLA-ENG-CSL-8234, Univ. of California, Los Angeles, 1982.

13. Pearl, J. On the Discovery and Generation of Certain Heuristics. AI Magazine (1983), 23-33.

14. G. Polya. How to Solve it. Princeton University Press, 1945.

15. Sacerdoti, E.D. Planning in a hierarchy of abstraction spaces. AI 5 (1974), 115-135.

16. Valtorta, M. A Result on the Computational Complexity of Heuristic Estimates for the A* Algorithm. University of North Carolina, 1981.

17. Vere, S.A. Relational Production Systems. AI 8 (1977), 47-68.