# 10

# And-or Graphs, Theorem-proving Graphs and Bi-directional Search

R. Kowalski
Department of Computational Logic
University of Edinburgh

**Abstract**

And-or graphs and theorem-proving graphs determine the same kind of search space and differ only in the direction of search: from axioms to goals, in the case of theorem-proving graphs, and in the opposite direction, from goals to axioms, in the case of and-or graphs. Bi-directional search strategies combine both directions of search. We investigate the construction of a single general algorithm which covers uni-directional search both for and-or graphs and for theorem-proving graphs, bi-directional search for path-finding problems and search for a simplest solution as well as search for any solution. We obtain a general theory of completeness which applies to search spaces with infinite or-branching. In the case of search for any solution, we argue against the application of strategies designed for finding simplest solutions, but argue for assigning a major role in guiding the search to the use of symbol complexity (the number of symbol occurrences in a derivation).

## INTRODUCTION

This paper investigates some of the search spaces and search strategies which are applied to the representation and attempted solution of problems in artificial intelligence. Our main objective is to demonstrate that both and-or graphs and theorem-proving graphs determine the same kind of search space and that the difference lies only in the direction of search – forwards from axioms to goal in theorem-proving graphs and backwards from goal to axioms in and-or graphs. This initial observation suggests the construction of bi-directional algorithms which combine forward and backward searches within a given search space. The resulting bi-directional strategy generalises the one studied by Pohl (1969) for path-finding problems.

We investigate the notions of exhaustiveness and completeness for search strategies. For both of these properties it is unnecessary to insist that the

search space contain only finitely many axioms or alternative goals or that only finitely many inference steps apply to a fixed set of premises or to a given conclusion. We discuss the utility of constructing search spaces which violate such finiteness restrictions and show how these spaces can be searched completely by employing symbol complexity to guide the search. The removal of the finiteness restrictions makes it impossible to apply Pohl's (1969) cardinality comparison of candidate sets in order to determine which direction of search to choose at any given stage. The decision to choose instead the direction which has smallest set of candidates of best merit resolves this dilemma and can be effectively applied precisely in those circumstances when it is possible to search the space completely in at least one of two directions.

We distinguish between problems which require a simplest solution and problems which require any solution. For the first kind of problem we investigate the construction of search strategies which are guaranteed to find simplest solutions. These strategies involve the use of heuristic functions to estimate the additional complexity of a simplest solution containing a given derivation. Under suitable restrictions the search strategies find simplest solutions and do so by generating fewer intermediate derivations (both those which are relevant and irrelevant to the eventually found solution) than do the more straightforward complexity saturation strategies. The heuristic search strategies cover those previously investigated by Hart-Nilsson-Raphael (1968) for path-finding problems, by Nilsson (1968) for and-or trees, and by Kowalski (1969) for theorem-proving graphs. In the case of bi-directional path-finding problems, we propose a method of updating the heuristic function so that it reflects properly the progress of search in the opposite direction. The resulting search strategy overcomes much of the inefficiency, noted by Pohl (1969), of the original bi-directional strategy.

Having investigated search strategies which aim to find simplest solutions, we then argue against applying similar strategies in situations which require finding any solution, no matter what its complexity. We argue that such search strategies have the undesirable characteristic of examining and generating equally meritorious potential solutions in parallel. We propose no concrete alternative, but suggest that more appropriate search strategies will need to employ methods for 'looking-ahead' in the search space and will need to bear a greater resemblance to depth-first searches than do those which belong in the family with search strategies for simplest solutions. We argue moreover that symbol complexity should occupy a central role in guiding the search for a solution.

The problems investigated in this paper assume a solution to the representation problem: for an initial semantically defined problem, how to obtain an adequate syntactic formulation of the problem. In the case of theorem-proving, the semantically defined problem might have the form of showing that one set of sentences implies another. The original problem does not

specify a search space of axioms, goals and basic inference operators. Indeed, many different search spaces can represent the same original problem, as in the case of resolution systems where different refinements of the resolution rule determine different search spaces for a single problem of demonstrating the unsatisfiability of a given set of clauses. Viewed in this context, the name 'theorem-proving graph', for a particular direction of search in a particular kind of search space, can be misleading. A given theorem-proving problem can be represented by different search spaces, not all of which need be interpretable as theorem-proving graphs when searched from the direction which starts with axioms and works toward the goal.

Conversely, the abstract graph-theoretic structure of problems incorporated in and-or graphs and in theorem-proving graphs is not confined to the representation of theorem-proving problems. In the next section of this paper we demonstrate an application of such search spaces to the problem of enumerating all subsets of a given set of objects.

This paper assumes no prior acquaintance with the research literature about search spaces and search strategies. On the contrary, our intention is that it provide a readable, and not overly-biased, introduction to the subject. For the same purpose, the reader will probably find Nilsson's book (1971) useful.

## DERIVATIONS, THEOREM-PROVING GRAPHS, AND-OR GRAPHS

We observe first that and-or graphs and theorem-proving graphs determine the same kind of search space (see figure 1). What distinguishes the two is the direction of search: forwards from initially given axiom nodes towards the goal node in theorem-proving graphs, and backwards from the goal node towards the initially given solved nodes in and-or graphs. Both directions of search can be combined in bi-directional search strategies which generalise those investigated for path-finding problems.

Typically, theorem-proving graphs/and-or graphs are used for problems of logical deduction where the task is one of finding some solution derivation of an initially given goal sentence from initially given axioms. With theorem-proving graphs, the initially given set of solved problems (axioms) is repeatedly augmented by the addition of deduced theorems until the goal sentence is derived and recognised as having been solved. With and-or graphs, the solution of the initially given goal problem is repeatedly reduced to the solution of alternative sets of subgoal problems, some of which might be immediately recognised as being solved, others of which might need further reduction. A solution is obtained when some such reduction set of subgoals consists entirely of completely solved subproblems (axioms) of the original problem. In both cases the search strategy can be regarded as selecting some candidate derivation and then generating it by generating all of its previously ungenerated sentences and connecting inference steps.

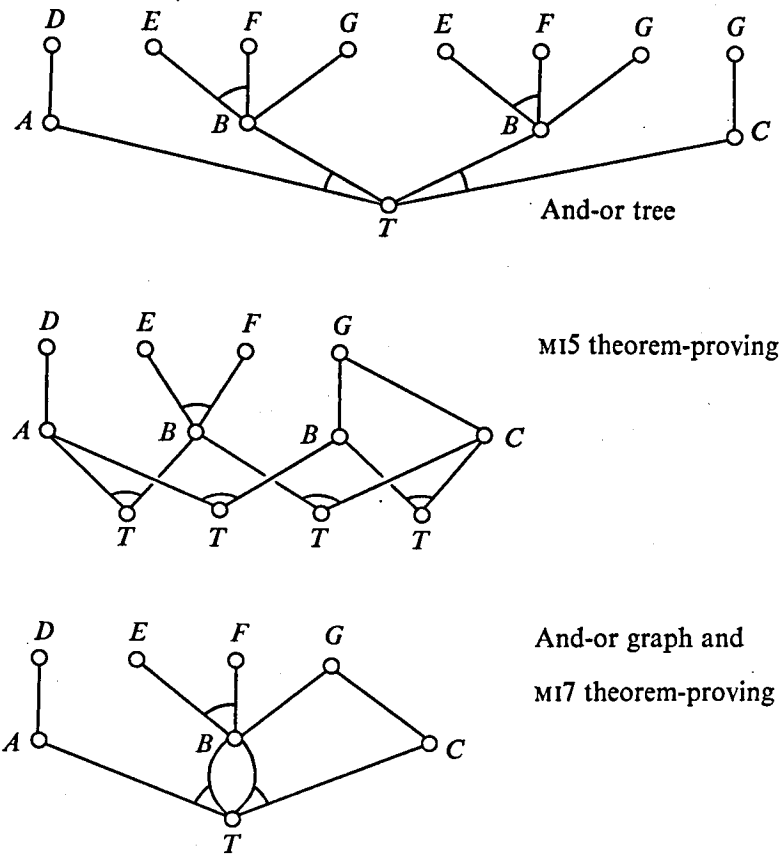
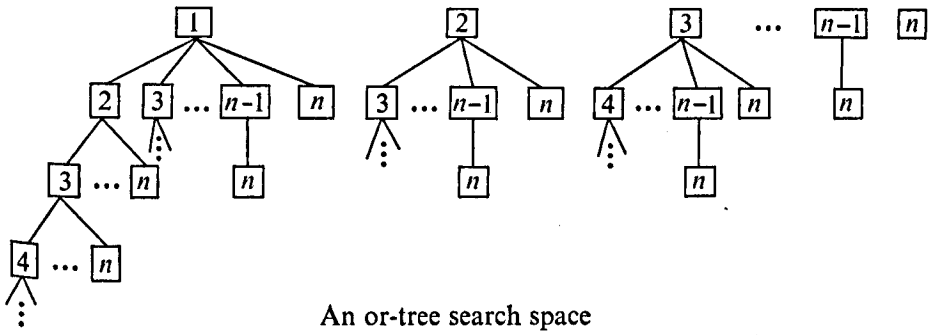


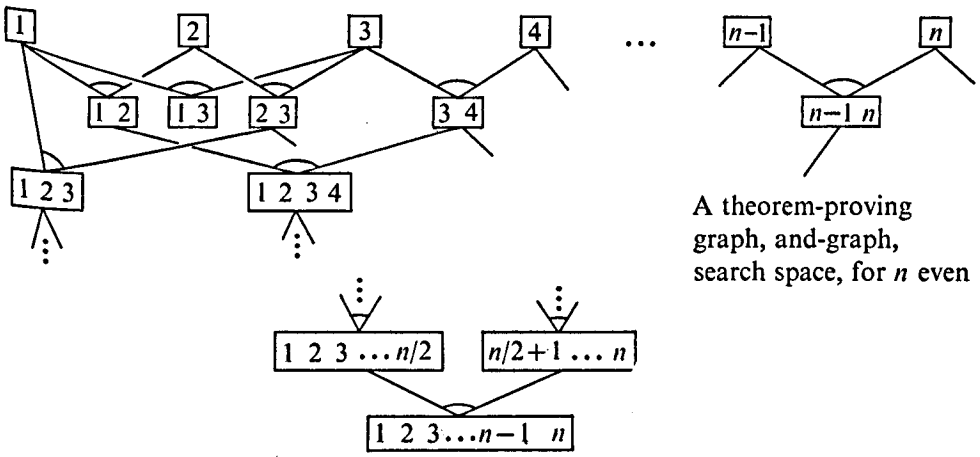

Figure 1. In the and-or tree, the identical problems *B* are encountered in different inference steps, working backwards from the goal *T*. Similarly *E*, *F* and *G* occur more than once in the and-or tree. All distinct occurrences of a sentence have distinct nodes associated with them. In the tree representation of theorem-proving graphs (used in *Machine Intelligence 5* (Kowalski 1969)), identical problems *T* are generated at different times, working forwards from the initial set of axioms {*D*, *E*, *F*, *G*}. Here too distinct occurrences of a sentence are associated with distinct nodes. In the and-or graph and in the graph representation of theorem-proving graphs (as defined in this paper) sentences are the nodes of the search space. Different ways of generating the same sentence are represented by different arcs connected to the sentence.

Theorem-proving graphs and and-or graphs can also be applied to the representation of search spaces in problems where the inference operation connecting premises with conclusion is not one of logical implication. Figure 2 illustrates just such an application where the 'inference operator' is a restricted form of disjoint union of sets. Figure 2 also illustrates a more conventional or-tree representation for the same problem of enumerating without redundancy all subsets of a given collection of *n* objects. Both search spaces contain the same number of nodes and achieve similar economies by

sharing a single copy of a subset when it is common to several supersets. For a particular application to a problem in urban planning of identifying a least costly collection of $m$ housing sites out of a total of $n$ possible, the use of the theorem-proving graph, and-graph, provides several advantages. In particular a certain bi-directional search strategy can be employed and




An or-tree search space




A theorem-proving graph, and-graph, search space, for $n$ even




Figure 2. The or-tree and the theorem-proving graph provide two different ways of enumerating all subsets of $n$ objects without redundancy. The theorem-proving graph is an and-graph, without any or-branching, because every subset can be derived in only one way; there are no alternatives. (Every subset is the conclusion of a single inference step. The two subsets which are premises of the inference step are disjoint. One of these subsets contains only elements which are less than all of the elements contained in the other premise subset. Either both premises have the same number of elements, or else the premise which contains the smaller elements contains one less element than the other premise.) Moving downwards in the space corresponds to generation of subsets in the theorem-proving graph direction, combining smaller subsets into larger ones. Moving upwards corresponds to generation in the and-graph direction, decomposing subsets into smaller ones.

interpreted as a model of the method of successive approximation. The search begins with aggregated collections of objects in the middle of the search space and works both forward combining smaller sets into larger ones and backwards splitting aggregates into smaller sets; this method of bi-directional search differs from the one which is discussed in this paper. Further details regarding the use of theorem-proving graphs and heuristic programming in the urban design problem are contained in the reports by Kowalski (1971) and du Feu (1971).

In the following definitions, the notions of sentence and inference operator should be interpreted liberally to include any kind of object and any operator which maps sets of objects onto single objects. A more formal and abstract treatment of these notions can be supplied along the lines detailed for MI 5 theorem-proving graphs (Kowalski 1969).

Let sentence $N$ follow directly from the *finitely* many sentences $N_1, \ldots, N_n$ by means of one application of an inference operator. Then $N$ is connected to each of $N_i$ by a directed *arc* (from $N_i$ to $N$), the collection of which is called an *and-bundle*. The *premises* $N_1, \ldots, N_n$, the *conclusion* $N$, and the connecting and-bundle constitute a single *inference step*. Corresponding to each axiom $N$ is a single inference step having $N$ as conclusion and an empty set of premises. (In the figures we have not drawn arcs associated with inference steps having axioms as conclusions.)

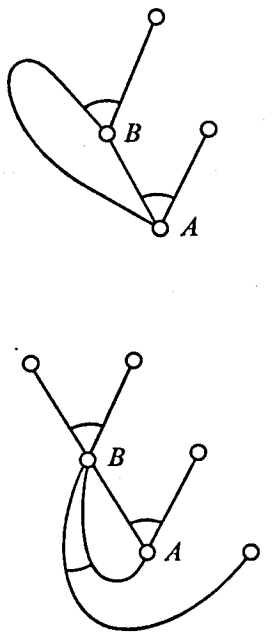



Figure 3. Some non-derivations.

A *derivation* $D$ is a finite set of sentences and connecting inference steps such that:

(1) Every sentence in $D$ belongs to at least one inference step in $D$.

(2) Exactly one sentence $C$ in $D$ is the conclusion of an inference step but premise of none.

(3) Every sentence in $D$ is the conclusion of at most one inference step in $D$.

(4) $D$ contains no infinite branches, where a *branch* in $D$ is a sequence of sentences $S_1, S_2, \ldots, S_n, S_{n+1}, \ldots$, where $S_1$ is in $D$ and $S_{n+1}$ is a premise of an inference step in $D$ which has $S_n$ as conclusion. (This condition guarantees that the graph associated with $D$ contains no cycles.)

Figure 3 illustrates some non-derivations.

The *premises* of $D$ are those sentences in $D$ which are conclusions of no inference steps belonging to $D$. The *conclusion* of $D$ is $C$. $D$ is *premise-free* if it has no premises. $D$ is a *reduction* if the conclusion it derives is the initially given goal sentence. $D$ is a *solution* if it is a premise-free reduction. All candidate derivations selected for generation in the forwards theorem-proving direction are premise-free. All candidates selected in the and-or direction are reductions. Every reduction derivation reduces the solution of the goal node to the solution of the premises of the derivation. Every premise-free derivation augments the initially given set of solved nodes by the conclusion it derives.

*Path-finding problems* constitute a special case of the derivation-finding problem: every inference step contains at most one premise. A derivation, therefore, consists of a single path connecting its premise with its conclusion.

## THE ALGORITHM

The bi-directional algorithm for and-or graphs/theorem-proving graphs described in this section includes as special cases

(1) uni-directional algorithms for theorem-proving graphs and for and-or graphs;

(2) bi-directional algorithms for path-finding problems;

(3) algorithms for finding simplest solutions, relative to a given measure of complexity, as well as algorithms for finding any solution, regardless of its complexity; and

(4) complexity saturation algorithms which use no heuristic information, as well as algorithms which do use heuristic information.

We do not claim that the general algorithm produces the most efficient algorithm for any of these special cases; our objective is, instead, to formulate an algorithm which abstracts what is common to many different search problems and search strategies. The intention is that best algorithms for special cases might be obtained by special additions or minor modifications to the general algorithm.

The algorithm proceeds by first selecting a direction of generation, direction **F** for forward generation or direction **B** for backward generation. If there are

no candidate derivations (containing exactly one inference step as yet ungenerated) the algorithm terminates without solution; otherwise it chooses a candidate which is of best merit among all candidates in the selected direction and then generates all previously ungenerated sentences and the one previously ungenerated inference step belonging to the chosen derivation. If an appropriate solution has not been generated (depending on the problem, either any solution or else one which is simplest), then the algorithm begins another cycle of selecting a direction of generation and then choosing and generating a candidate derivation. The algorithm terminates when an appropriate solution is generated. If the objective is to generate a simplest solution then the algorithm continues until it finds a simplest solution and identifies it as being simplest.

At any given time, the set of sentences and inference steps so far generated is stored in two sets **F** and **B**. **F** contains all sentences and inference steps belonging to already generated premise-free derivations. (Thus all sentences in **F** are solved sentences, or theorems.) **B** contains all sentences and inference steps belonging to already generated reduction derivations. The intersection of **F** and **B** need not in general be empty. A derivation is regarded as *generated* provided all its sentences and inference steps have been generated, even though they might have been generated as part of some other candidate derivation explicitly selected for generation. Thus some derivations are deliberately selected and generated, while others, with which they share inference steps, may be generated adventitiously. The following definitions specify the algorithm more precisely.

Suppose that there exists an inference step which does not belong to **F**, but all of whose premises do belong to **F**. Then any premise-free derivation which consists of

(1) this inference step,
(2) its conclusion, and
(3) for every premise of the inference step, exactly one premise-free derivation, contained in **F**, whose conclusion is the premise,

is a *candidate* for generation in direction **F**.

Suppose that $D_0$ is a reduction derivation, contained in **B**, and every premise of $D_0$ is the conclusion of some inference step not belonging to **B**. Then any reduction derivation which consists of

(1) just one such inference step,
(2) its premises, and
(3) $D_0$,

is a *candidate* for generation in direction **B**.

Assume that a notion of *merit* has been defined on derivations, such that for any collection of derivations it is always possible to select effectively a derivation of best merit, in the sense that no other derivation belonging to the collection has better merit. Assume there is given just one goal problem. This, without loss of generality, covers cases where there are finitely many

goals to be solved simultaneously or denumerably many (finite or infinite) alternative goals.

(1) *Initialize* both **F** and **B** to the null set.

(2) *Select* one of **F** or **B** as the *direction* **X** of generation.

(3) *Terminate without solution* if there are no candidates for generation in direction **X**. Otherwise continue.

(4) *Select and generate a candidate* derivation $D$ for **X** having best merit among candidates for **X**. Generate $D$ by adding to **X** the single inference step and all sentences in $D$ not already belonging to **X**.

(5) *Update* **F** and **B** by adding to both of them all inference steps and sentences belonging to any already generated premise-free derivation of a sentence in **B**.

(6) *Terminate with solution* if (a) some solution $D$ is contained in **F** or **B**, and (b) no candidate for direction **F** or **B** has better merit than $D$. Otherwise go back to (2).

Figure 4 illustrates the operation of a single cycle of the algorithm.

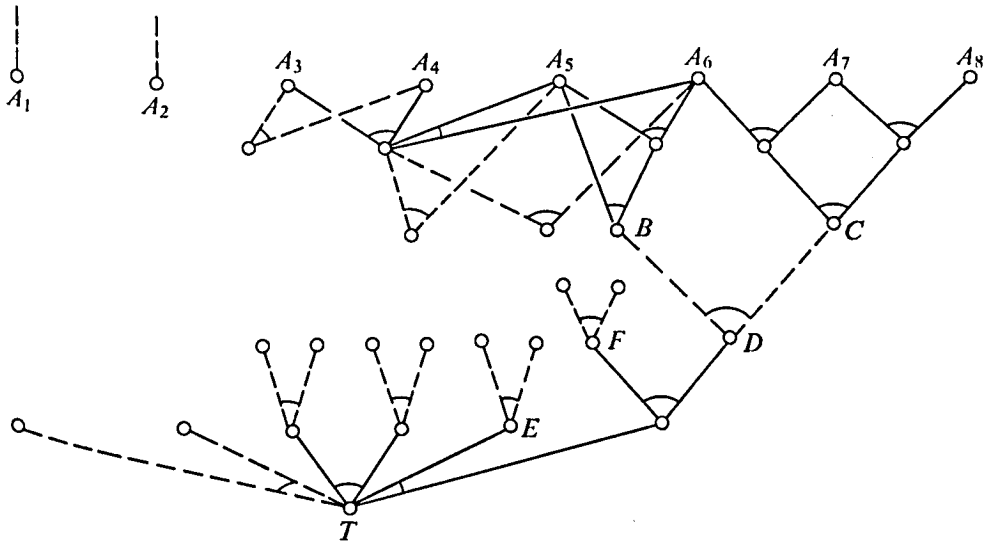



Figure 4. The state of the search space at the beginning of a cycle of the algorithm: Solid lines represent arcs which belong to inference steps already generated. Broken lines represent arcs which belong to inference steps which are candidates for generation. Inference steps which have not yet been generated and which are not candidates for generation are not represented in the figure. Of the axioms $A_1, \ldots, A_8$, the first two axioms have not yet been generated. The inference steps which have the remaining axioms as conclusions are not illustrated.

The inference step which has $B$ and $C$ as premises and $D$ as conclusion is a candidate for generation in both direction **F** and direction **B**. The sets **F** and **B** do not yet have any elements in common.

Suppose now that **B** is chosen as direction of generation and that the reduction derivation having premises $\{E, F, B, C\}$ is the candidate selected for generation. Then in step (4), the sentences $B$ and $C$ are added to **B** along with the inference step having them as premises. In the update step, the same inference step is also added to **F** along with its conclusion $D$. At the same time, all sentences and inference steps in the derivations of $B$ and $C$ are added to **B**.

**Remarks**

(1) The most complicated uni-directional case of the algorithm is the case of and-or graphs (where the direction **X** is always chosen to be **B**). The set **F**, which is initially empty, is augmented whenever a sub-problem is solved. The updating step corresponds, in the usual algorithm for searching and-or trees, to the operation of labelling sub-problems as solved. The effect of labelling sub-problems as unsolvable is obtained by not counting as a candidate derivation one whose premises are not conclusions of some (as yet ungenerated) inference step.

(2) The algorithm can be implemented easily in a list processing system, by associating with every inference step backward pointers from the conclusion to the premises. Such pointers correspond to the directed arcs connecting conclusion with premises in the and-bundle of the inference step. Derivations already generated can be accessed by following the pointers backwards.

(3) Unless the problem is one of obtaining a simplest solution, condition 6(b) is unnecessary and can be disregarded. 6(b) is used only when searching for simplest solutions and when employing certain kinds of merit orderings. But even then 6(b) is unnecessary for finding simplest solutions in uni-directional and-or tree search or in uni-directional theorem-proving graph search. This condition is necessary, however, for all bi-directional searches as well as for uni-directional and-or graph search. We shall investigate these matters in greater detail in the section concerned with admissibility.

(4) Our algorithm differs from others reported elsewhere in the literature most importantly in one respect. Ordinarily a search strategy is regarded as selecting derivations which are candidates for expansion. For path-finding problems, a selected derivation is fully expanded by extending it in all possible ways by the addition of a single inference step (arc). For and-or trees, a derivation is selected, one of its premises is chosen and then all inference steps are generated which have the given premise as conclusion. In our algorithm, however, a search strategy is always regarded as selecting derivations which are candidates for generation. A selected derivation is generated by generating its single previously ungenerated inference step.

Our algorithm degenerates when it is impossible to predict the merit of an ungenerated candidate derivation without a detailed examination which requires that the derivation be fully generated. For **X** either **F** or **B**, let $\bar{\mathbf{X}}$ consist of those sentences and inference steps not in **X** but belonging to derivations which are candidates for generation in direction **X**. The algorithm selects and generates an inference step in $\bar{\mathbf{X}}$ belonging to a candidate derivation of best merit. In the degenerate case, it is necessary to store the sets $\bar{\mathbf{X}}$ explicitly inside the computer. The algorithm then behaves indistinguishably from one which fully extends candidate derivations for expansion. **X** can then be re-interpreted as containing the set of fully expanded nodes and $\bar{\mathbf{X}}$ as containing the candidates for expansion.

In several important cases, the sets $\bar{\mathbf{X}}$ need not be explicitly stored in order

to select and generate candidate derivations of best merit. Such cases are more closely related to partial expansion (Michie and Ross 1969) than they are to full expansion. The algorithm does not however, as in partial expansion, first select previously generated nodes of best merit and then extend them by applying the best applicable operator; nor does it first pick an operator of best merit and then apply it to nodes of best merit. For in neither case does the corresponding derivation, selected for generation, necessarily have best merit among all candidates for generation. Our algorithm, in contrast, selects and generates a candidate of best merit even when it cannot be obtained from nodes and operator either of which individually has best merit.

The chief advantage of regarding the sets $\tilde{\mathbf{X}}$ as containing candidates for generation is that then the sets $\tilde{\mathbf{X}}$ need not be finite in order to admit the construction of exhaustive search strategies. In particular, it is quite useful to construct spaces which have *infinite or-branching*, in the sense of having infinitely many axioms, infinitely many alternative goals, or infinitely many inference steps which apply to a fixed set of premises or to a given conclusion. In fact, it seems apparent that such search spaces are often more appropriate for representing certain problems and can be searched more efficiently than alternative search spaces which involve finite or-branching. We shall continue our discussion of this topic in the section concerned with completeness.

## DELETION OF REDUNDANCIES

The use, in our algorithm, of and-or graphs, as opposed to and-or trees, eliminates the redundancy which occurs when identical sub-problems are generated, not identified, and consequently solved separately and redundantly as though they were distinct. This use of and-or graphs has been necessitated by wanting to regard generation of derivations in direction **B** as backward search in a theorem-proving graph. Similarly, in order to regard generation in direction **F** as 'backward' search in an and-or graph, we have had to abandon the tree representation of theorem-proving graphs (Kowalski 1969). The graph representation we use here can be obtained from the tree representation by identifying nodes having the same label. The graph representation eliminates the redundancy which arises when the same sentence is derived by distinct premise-free derivations, which can be used interchangeably in all super-derivations containing the given sentence. In both graph representations, and-or graph and theorem-proving graph, maximal use is made of sharing common sub-derivations. In practice the recognition and identification of regenerated sentences requires certain computational overheads. These overheads can be reduced by using the hash-coding methods discussed by Pohl (1971).

The present version of the algorithm admits another kind of redundancy: when a sentence in **F** is derived by more than one premise-free derivation contained in **F**. It is unnecessary to save all of these derivations by keeping all backward pointers associated with each of the inference steps which have

the given sentence as conclusion. It suffices instead to preserve at any given time only those pointers associated with some single distinguished inference step. In the case where one is interested in finding any solution no matter what its complexity, virtually any decision rule can be employed to select the distinguished inference step. However, completeness and efficiency are ordinarily served best by choosing the inference step which belongs to the simplest premise-free derivation of the sentence. If a search strategy is *admissible*, in the sense that it always finds a simplest solution whenever one exists, then the use of this decision rule, to remove redundant inference steps, preserves admissibility.

The analogous redundancy in **B** occurs when distinct derivations contained in **B** have the same set of premises. The different derivations represent different reductions of the original problem to the same set of sub-problems. It suffices to preserve only one such reduction at a time. Chang and Slagle (1971) incorporate into their algorithm for and-or graphs a test for just such redundancy. More generally, redundancy occurs when the premises of one derivation are a subset of those belonging to another. In such a case the first derivation represents a reduction of the original problem to a subset of the sub-problems associated with the second derivation. Unfortunately it seems virtually impossible to eliminate this redundancy in an efficient way, because of the complicated manner in which derivations in **B** share substructure. The situation does not improve even if consideration is limited to the case where distinct derivations have identical sets of premises. In general, virtually every inference step belonging to a derivation in **B** is shared with some other derivation belonging to **B**. For this reason a derivation cannot be removed from **B** by the simple removal of one of its inference steps, since this would almost certainly result in the unintended removal of other derivations. Perhaps the single case where redundant derivations can easily and efficiently be eliminated is the case of path-finding problems. Here, eliminating redundant derivations in **B** can be accomplished exactly as it is done in **F**.

## COMPLETENESS

A search strategy is *complete* if it will find a solution whenever one exists. It is *exhaustive* for direction **X** if it will eventually, if allowed to ignore the termination-with-solution condition, generate all sentences and inference steps which can be generated in direction **X**. Clearly, *every exhaustive search strategy is complete.* That *a search strategy can be complete without being exhaustive* is illustrated in figure 5.

It is of practical value to find useful conditions under which search strategies can be certified to be complete. $\delta$-finiteness is such a condition. As defined below, this condition is a straightforward extension of the one given in Kowalski (1969) and is more general than the one considered elsewhere in the literature. An important feature of our definition is that it applies to cases where or-branching is infinite. A derivation is considered

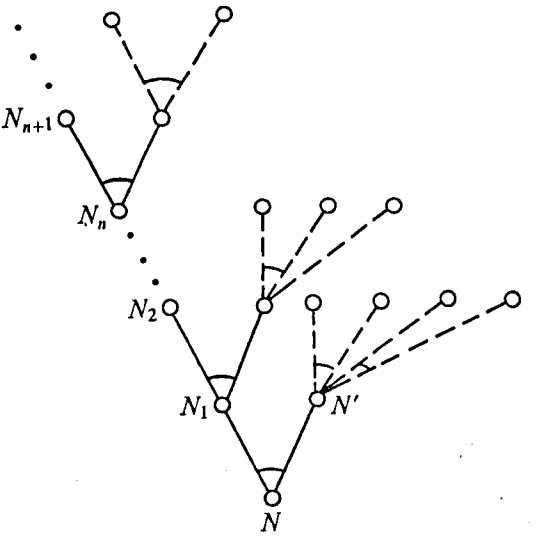



Figure 5. The search space contains an infinite chain of reductions: $N$ can be solved only if $N_1$ can be solved; $N_1$ can be solved only if $N_2$ can; $N_n$ can be solved only if $N_{n+1}$ can. A search strategy which is depth-first along and-branches can be complete without ever generating any of the search space above node $N'$. In general, depth-first search along and-branches, although it sometimes violates exhaustiveness, in no way adversely affects completeness. (A strategy is *depth-first along and-branches* if from among any pair of premises in an inference step, it selects one premise and determines its solvability or unsolvability before deciding that of the other premise.)

*generatable* in direction **F** if it is premise-free and is considered *generatable* in direction **B** if it is a reduction derivation of the goal sentence. A merit ordering of derivations is *δ-finite* for direction **X** if for any derivation $D$ generatable in that direction there exist only finitely many other derivations, generatable in the same direction which have merit better than or equal to that of $D$. A good example of a δ-finite merit ordering is the one measured by the total number of distinct occurrences of symbols in a derivation and called its *symbol complexity*. For denumerable languages constructed from finite alphabets, symbol complexity has the property that, for each natural number $n$, only a finite number of derivations has complexity $n$. It follows that merit measured by symbol complexity is δ-finite, even in search spaces which have infinite or-branching. Size and level are δ-finite for search spaces which have finite or-branching but are not δ-finite for spaces with infinite or-branching. A δ-finite search strategy need not be exhaustive but always is complete.

*Theorem*

Any bi-directional search strategy using a δ-finite merit ordering for some direction **X** is complete, provided that at no stage does it become uni-directional in the opposite direction $\mathbf{Y} \neq \mathbf{X}$.

*Proof.* Suppose that $D^*$ is some solution which is not generated by the search strategy. Then there exists some sub-derivation $D$ of $D^*$ which at some stage

becomes a candidate for generation in direction X but at no stage ever gets generated. Unless the search strategy eventually terminates by finding some solution other than $D^*$, it must otherwise select and generate in direction X candidate derivations $D_1, \ldots, D_n, \ldots$ without termination. But then at some stage, because of $\delta$-finiteness, one of the selected derivations $D_n$ has merit worse than $D$; and this is impossible.//

It seems to be a common misconception that finite or-branching is a necessary condition either for completeness or for efficiency. In particular, it is sometimes believed that resolution theorem-proving inference systems have the advantage over alternative inference systems that only a finite number of inference steps can have a given set of sentences as premises. This belief appears to be without foundation. Similarly, the condition often required in pure logic that it be effectively decidable whether a given conclusion follows directly from a set of premises by one application of an inference rule, is yet another restraint which has no apparent relevance to the construction of complete or efficient proof procedures.

It is especially important to bear in mind that infinite or-branching poses no hindrance to efficient theorem-proving when evaluating proof procedures for higher-order logic or for inference systems like those investigated by Plotkin (1972). The practical utility of constructing search spaces with infinite or-branching and of employing exhaustive search strategies is well illustrated by the efficient theorem-proving programs written by Siklossy and Marinov (1971). It is interesting to note that the notion of merit which guides their search strategy is similar to that of symbol complexity.

Another relaxation of constraints which can usefully be employed to improve the performance of search strategies is that the relative merit of derivations be allowed to depend upon the state of computation, that is, upon the entire set of derivations generated up to a given cycle of the algorithm. This liberalized notion of merit, applied to evaluation functions, has been investigated by Michie and Sibert (1972). A useful application to bi-directional heuristic search is described in the section concerned with bi-directionality.

## COMPLEXITY, DIAGONAL SEARCH, AND ADMISSIBILITY

Let $f$ be a real-valued, non-negative function defined on derivations. $f$ is an *evaluation function* if it is used to define the merit of derivations:

$D_1$ has *better merit* than $D_2$ if $f(D_1) < f(D_2)$,

$D_1$ and $D_2$ have *equal merit* if $f(D_1) = f(D_2)$.

$f$ is *monotonic* if no derivation has merit better than any of its sub-derivations, that is, if

$f(D') \leqslant f(D)$ whenever $D' \subseteq D$.

A *complexity measure* (or cost measure) is any monotonic evaluation function. Various measures which have been used to compute the complexity of derivations are

(1) *size*, the number of sentences in a derivation,
(2) *level*, the largest number of sentences along any one branch of a derivation,
(3) *sum cost*, the sum of all costs, for all sentences in a derivation, where every sentence has an associated cost (or complexity), and
(4) *max cost*, the largest sum of all costs, for all sentences along any one branch of a derivation.

Symbol complexity is the special case of sum cost which is obtained when the complexity of an individual sentence is measured by the number of occurrences of distinct symbols in the sentence. Size is the special case of sum cost obtained when each sentence has associated with it a unit cost. Similarly level is max cost where individual sentences have unit costs. Useful variant definitions of complexity are obtained by associating costs with inference steps rather than with individual sentences. On the whole, most of the discussion of complexity below is independent of the exact details of the definition involved. The only important property which we need require of complexity measures is that they be monotonic.

It often occurs, especially in path-finding problems, that a particular measure of complexity is given and it is required to find a simplest solution (one having least complexity). Perhaps more often no such measure is given and it is required only to find any solution, no matter what its complexity. In the latter case it is plausible that some measure of the complexity of partial solutions can usefully be employed to improve the efficiency of the search. Such a measure might be employed in a depth-first search both to extend a given derivation along simpler lines in preference to ones more complex as well as to suspend the further extension of a derivation when it has become intolerably complex. Alternatively, it may be decided to replace the original problem of finding an arbitrary solution by the related, but different, problem of finding a simplest or most elegant one. We will delay a more thorough discussion of this problem until we have first dealt with the problem of finding a simplest solution.

#### Diagonal search strategies

These strategies (or more precisely the special case of bounded diagonal search) are equivalent to the branch-and-bound algorithms employed in operations research (Hall 1971), to the algorithms of Hart-Nilsson-Raphael (1968) for path-finding problems, to those of Nilsson (1968) for and-or trees, and to those of Kowalski (1969) for theorem-proving graphs. These algorithms are well suited for problems where it is required to find a simplest solution. Some authors, including Hall (1971), Kowalski (1969), Nilsson (1971) and Pohl (1970), have recommended using such algorithms (or minor variations) for problems requiring any solution, regardless of its complexity.

For a given complexity function $g$, a search strategy employing an evaluation function $f$ is a *diagonal search strategy* if

$$h(D)=f(D)-g(D)\geqslant 0 \text{ for all derivations } D.$$

$h$ is called the *heuristic function* and $f$ is often written as the sum $g+h$. The evaluation function $f$ of a diagonal search strategy can often be interpreted as providing an estimate $f(D)$ of the complexity of a simplest solution $D^*$ containing the derivation $D$. The heuristic function $h$ then estimates that part of the complexity of $D^*$ which is additional to that of $D$. A diagonal search strategy is a *complexity saturation strategy* if the complexity measure $g$ is used as evaluation function $f$, that is, if $f \equiv g$ and therefore $h \equiv 0$.

A diagonal search strategy is an *upward diagonal* strategy if, whenever two candidate derivations for the same direction have joint best merit but unequal complexity, the candidate derivation selected for generation is the one having greater complexity. The use of upward diagonal strategies helps to avoid the simultaneous exploration of equally meritorious partial solutions.

The geometric intuition underlying the terminology 'diagonal' and 'upward diagonal strategies', is illustrated in figure 6. Inference steps in the search space are treated as points in a two-dimensional space. An inference step has co-ordinates (g, h) if it is generated by the search strategy as part of a selected candidate derivation $D$ with complexity $g(D)=$g and heuristic value $h(D)$

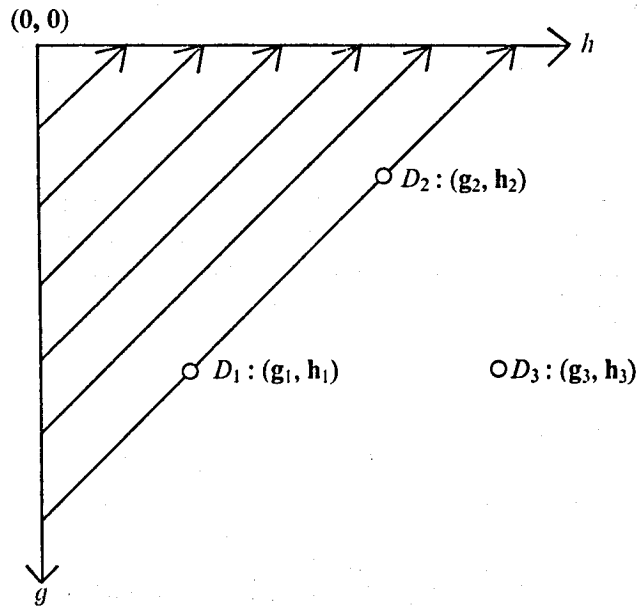



Figure 6. Whenever possible, diagonal search generates $D_1$ before $D_2$ and $D_2$ before $D_3$. It generates $D_2$ before $D_1$ only if $D_2$ is a sub-derivation of $D_1$ and generates $D_3$ before $D_1$ or $D_2$ only if $D_3$ is a sub-derivation of $D_1$ or $D_2$. If the evaluation function is monotonic then $D_3$ could not be a sub-derivation of $D_1$ or $D_2$, because, if it were, it would have larger evaluation. Therefore, for monotonic evaluation functions, diagonal search generates all derivations on a given diagonal before generating any on a longer diagonal. Notice that a diagonal search strategy is bounded if its evaluation function is monotonic and if $h(D^*)=0$ whenever $D^*$ is a solution. All solution derivations lie on the $g$-axis.

$=\mathbf{h}$. A diagonal search strategy generates inference steps along diagonals, moving away from the origin $(\mathbf{0}, \mathbf{0})$, generating inference steps on shorter diagonals (having smaller $\mathbf{g}+\mathbf{h}$) in preference to inference steps on longer diagonals (having greater $\mathbf{g}+\mathbf{h}$). Within a diagonal, upward diagonal search moves up the diagonal whenever possible, generating inference steps lower on the diagonal (having greater $\mathbf{g}$) before inference steps higher on the diagonal (having smaller $\mathbf{g}$).

A diagonal strategy is *bounded* if for all candidate derivations $D$, $f(D)$ is less than or equal to the complexity of any solution containing $D$, that is, if

$$f(D) \leqslant g(D^*) \text{ whenever } D \subseteq D^* \text{ and } D^* \text{ is a solution.}$$

Notice that any complexity saturation strategy is bounded.

#### Admissibility

A search strategy is admissible if it terminates with a simplest solution, whenever the search space contains any solution.

*Theorem*

If a bounded diagonal strategy terminates with a solution derivation $D^*$ then $D^*$ is a simplest solution.

*Proof.* Suppose $D_0$ is a simpler solution than $D^*$ and suppose that termination has occurred because no candidate in direction $\mathbf{X}$ has better merit than $D^*$. Then at time of termination some sub-derivation $D_0'$ of $D_0$ is a candidate for generation in direction $\mathbf{X}$. Moreover $D_0'$ has merit better than $D^*$ because

$$f(D_0') \leqslant g(D_0) < g(D^*) = f(D^*).$$

But this is in violation of the termination condition.//

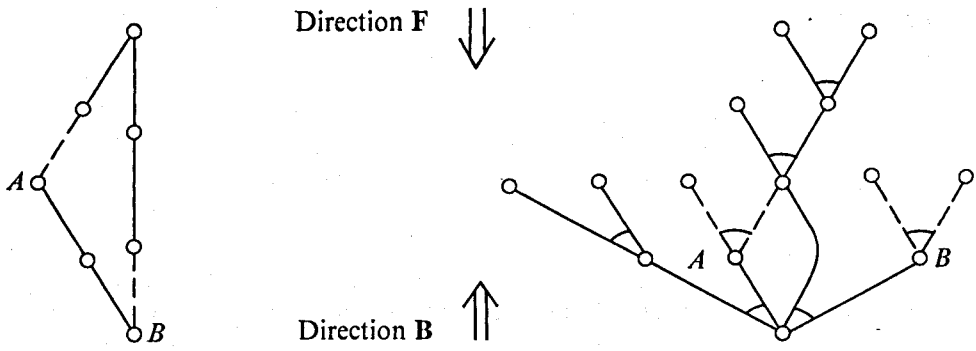
*Corollary*

If a bounded bi-directional diagonal search strategy is $\delta$-finite for some direction $\mathbf{X}$, then it is admissible, provided that at no stage does it become uni-directional in direction $\mathbf{Y} \neq \mathbf{X}$.

In the case of uni-directional search either in and-or trees or in theorem-proving graphs, it can be shown that when a bounded diagonal strategy first generates a solution $D^*$, then no candidate for generation has merit better than $D^*$. Thus condition 6(b) is automatically satisfied and need not be tested explicitly. In these cases, it suffices to terminate, therefore, as soon as a first solution has been generated. Figure 7 illustrates the need for 6(b) when searching for simplest solutions in bi-directional path-finding problems and in uni-directional and-or graph search.

An interesting application of bounded diagonal search can be made in certain cases where the problem is one of enumerating all nodes in a finite search space and of verifying that they all have some property $P$. Under certain conditions it is possible to verify that all nodes have property $P$ without generating all of the search space. Reformulate the problem as being that of finding some derivation of a node having property not-$P$. Assume that for some notion of complexity $g$, it is possible to construct a bounded diagonal search strategy. Assume also that the maximum complexity of any

solution is g. Then under these conditions, if a node having property not-$P$ is not found before generating a candidate $D$ with value $f(D)>g$, then it suffices to terminate the search and to conclude that no node in the search space has property not-$P$ and that all nodes, therefore, have property $P$.




Bi-directional search for a path-finding problem

Uni-directional, and-or graph, search for a derivation-finding problem

Figure 7. For both of these problems, all inference steps in the search space are illustrated and either have already been generated or else are contained in candidates for generation. For the bi-directional search, F is the chosen direction of generation. In both cases complexity is measured by size. Assume that, at the stage of the algorithm illustrated here, both candidate derivations in each of the two search spaces have heuristic value zero. Then, in both spaces, the candidate derivation containing the node $A$ is the candidate having best merit. When this candidate is selected and generated, a solution derivation containing $A$ is adventitiously generated. In neither case is this solution a simplest one and its merit is worse than that of the remaining candidate derivation containing the node $B$.

## OPTIMALITY

The Optimality Theorem of Hart, Nilsson and Raphael (1968) states that, unless it is better informed, no admissible algorithm generates fewer nodes before termination than does a bounded diagonal strategy. They define one admissible algorithm to be *better informed* than another if it admits the calculation of a better heuristic function, that is, if the heuristic information of the first strategy is represented by $h_1$ and if the heuristic information of the second is represented by $h_2$, then $h_1(D)\geqslant h_2(D)$ for all derivations and $h_1(D)>h_2(D)$ for some derivation implies that the first strategy is better informed than the second. With these definitions, Hart, Nilsson and Raphael prove the Optimality Theorem for uni-directional path-finding. Under the assumption that both admissible strategies are bounded diagonal search strategies, Kowalski (1969) proves the Optimality Theorem for uni-directional theorem-proving graph search, and Chang and Slagle (1971) prove the theorem for an uni-directional and-or graph search strategy similar to, but different from, diagonal search. Since the authors of the original Optimality

Theorem do not specify what it means for a non-diagonal strategy to employ a heuristic function, it is not clear that they have proved a stronger Optimality Theorem, which is free of the assumption that both search strategies are diagonal.

Even the weak version of the Optimality Theorem fails for bi-directional path-finding and for uni-directional and-or graph search. In both cases the attempted proof fails for the same reason. The proof requires it to be shown that if an inference step is generated by the better informed strategy then it is also generated by the other, worse informed strategy. The argument proceeds by showing that, for the latter strategy, the inference step in question belongs to a candidate derivation of merit better than that of a simplest solution and is therefore generated before the search strategy terminates. This proof works for uni-directional theorem-proving search and for the special case of uni-directional path-finding, because, in these cases, once a derivation becomes a candidate for generation it remains a candidate of unchanging merit until it is selected for generation. This is not the case, for instance, for and-or tree search. A derivation which is candidate for generation at one stage may cease to be a candidate without ever being generated at a later stage. Its ungenerated inference step may remain ungenerated as part of a new candidate derivation which contains the original one. The new derivation may have merit worse than the old. The ungenerated inference step, which once belonged to a candidate derivation having better merit than a simplest solution, may now belong only to candidates whose merit is worse than that of a solution.

Chang and Slagle rescue the proof of the Optimality Theorem by altering the definition of candidate reduction derivation. Initially, before the goal sentence has been generated, any single inference step which has the goal as conclusion is a *candidate* for generation. Afterwards, a derivation $D$ is *candidate* for generation if some reduction derivation $D_0$ is a sub-derivation of $D$ which has already been generated and $D$ contains only generated inference steps belonging to $D_0$ and, for each premise of $D_0$, a single ungenerated inference step whose conclusion is the given premise. The effect of this definition is to assure that once an ungenerated inference step belongs to a derivation which is a candidate for generation then that derivation remains a candidate of constant merit until the inference step is generated. But it is just this property which was needed to save the proof of the Optimality Theorem.

The Chang-Slagle algorithm can be regarded as a special case of the general bi-directional algorithm and, for and-or tree search, as a special case of Nilsson's algorithm (1968). Viewed in this way, their algorithm amounts to a strategy of *breadth-first search along and-branches*, which given a reduction derivation investigates the solvability or unsolvability of all its premises in parallel. Such a strategy does not seem optimal in any intuitive sense and is demonstrably inefficient in many cases.

### BI-DIRECTIONALITY

Various methods have been suggested for determining, in path-finding problems, which direction of generation should be selected in a given cycle of the algorithm. Among these are the method of alternating between **F** and **B** and of choosing that of **F** and **B** which possesses a candidate derivation of best merit. Pohl discusses these and related methods in his papers (1969, 1971). In particular, he provides theoretical and experimental arguments to support his method of *cardinality comparison* which chooses that of **F** or **B** for which the set of candidates has least cardinality. Since we are especially interested in the case where the sets of candidates are infinite, we need to formulate a different criterion for the choice of direction. Our criterion, in fact, can be regarded as a refinement of Pohl's:

*Choose that of* **F** *or* **B** *which has the fewest candidates of best merit.*

Notice that if a merit ordering is $\delta$-finite for **F** or **B** then in that direction, although the number of candidates for generation may be infinite, the numbers of candidates of best merit is always finite.

A more serious problem with bi-directional searches, independent of the method for choosing direction of generation, is that the two directions of search may pass one another before an appropriate solution derivation is found and verified. The resulting bi-directional search may then generate more derivations than either of the corresponding uni-directional searches. Such a situation may occur with bounded diagonal strategies and even occurs with our present formulation of bi-directional complexity saturation strategies. As the algorithm is presently formulated a bi-directional complexity saturation strategy must both generate a simplest solution (of complexity **g**) and also verify that the solution is simplest by generating all candidates of complexity less than **g** for one of the directions **X**. Such a bi-directional strategy does all the work of a uni-directional strategy for direction **X** and, in addition, generates extra derivations in the opposite direction. The algorithm can be modified so that bi-directional complexity saturation behaves as it should: generating a simplest solution of complexity $\mathbf{g_F}+\mathbf{g_B}$ (where $\mathbf{g_X}$ is the complexity of that part of the solution which is contained in direction **X**) and generating all candidates in direction **X** of complexity less than $\mathbf{g_X}$. Most importantly, the modifications necessary to achieve this effect apply equally to the more general case of bi-directional bounded diagonal strategies. The resulting improvement in efficiency should significantly increase the utility of such search strategies. To minimize the complications involved in the following discussion, we limit ourselves to consideration of bounded diagonal search strategies for path-finding problems.

Suppose that every partial path maximally contained in direction **X** has complexity greater than or equal to **c**. (A derivation is *maximally contained* in **X** if the addition of some single ungenerated inference step makes it a candidate for generation in direction **X**.) Then no partial path *D* which is candidate for generation in the opposite direction **Y** can be contained in a

solution path whose additional complexity (in addition to that of $D$) is less than **c**. Therefore we may require that

$h(D) \geqslant \min(g(D'))$, for $D'$ maximally contained in **X**.

If $h(D)$ is less than $\min(g(D'))$ then we should reset its value to $\min(g(D'))$.

Suppose that every partial path $D'$ maximally contained in direction **X** has value $f(D')$ greater than or equal to **e**. Then **e** is a lower bound on the complexity of a simplest solution. We may require, therefore, that for all candidates $D$ for generation in the opposite direction **Y**

$f(D) \geqslant \min(f(D'))$ for $D'$ maximally contained in **X**.

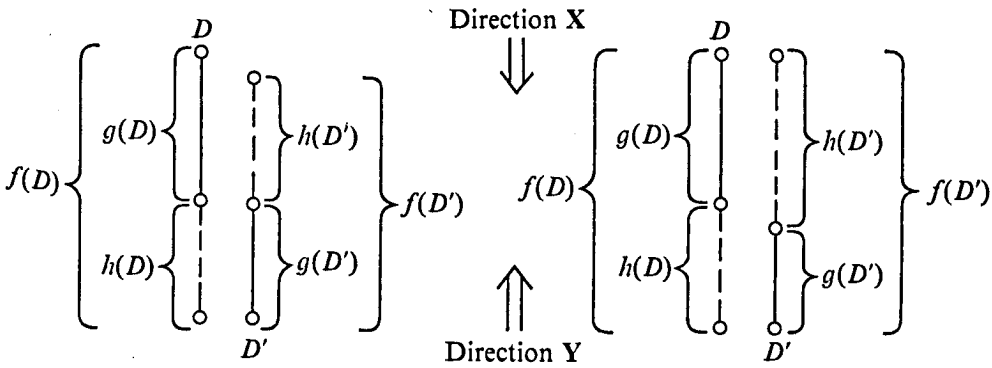
If $f(D)$ is less than the right-hand member of the inequality then we reset its value so that it is equal.

These two ways, of updating the heuristic function for direction **Y** by keeping track of $h$ and $f$ values for direction **X**, can be combined to obtain even better-informed heuristic functions. For $D$, candidate for generation in direction **Y**, we may insist that

$h(D) \geqslant \min[g(D') + (h(D') \dot{-} g(D))]$ for $D'$ maximally contained in **X**, where

$$a \dot{-} b = \begin{cases} 0, \text{ if } a \leqslant b \\ a - b, \text{ otherwise.} \end{cases}$$

Figure 8 illustrates a proof that such a heuristic function always yields a bounded diagonal search strategy.




*Case* (1) $h(D') \dot{-} g(D) = 0$ implies $g(D') + (h(D') \dot{-} g(D)) = g(D')$.

*Case* (2) $h(D') \dot{-} g(D) > 0$ implies $g(D') + (h(D') \dot{-} g(D)) = f(D') - g(D)$.

Figure 8. Suppose that $D$ is a candidate for generation in direction **X** and suppose that it is contained in some simplest solution $D^*$. Then $D^*$ contains some partial path $D'$ which is maximally contained in **Y**. There are two cases:
(1) if $h(D') < g(D)$ then $h(D)$ should not be less than $g(D')$.
(2) If $h(D') > g(D)$ then $f(D)$ should not be less than $f(D')$ (which is already less than or equal to $g(D^*)$). But this means that $h(D) = f(D) - g(D)$ should not be less than $f(D') - g(D)$.
In both cases $h(D)$ should not be less than $g(D') + (h(D') \dot{-} g(D))$.

## FINDING ARBITRARY SOLUTIONS

It has sometimes been assumed that the problem of finding any solution can be adequately replaced by the problem of finding a simplest solution. Pohl (1970) and Nilsson (1971) adopt a more liberal position favouring the use of diagonal search strategies with an evaluation function of the form $g+\omega h$ where $h$ is a heuristic function and $0 \leqslant \omega$. We shall argue here that such search strategies are not adequate to deal efficiently with the kind of search spaces that arise in artificial intelligence problems.

Our main objection to diagonal search strategies is that they investigate equally meritorious alternatives in parallel. For this reason, bounded diagonal strategies are even better suited for finding all simplest solutions than they are for finding any simplest solution. (Just change the termination condition so that termination takes place when all candidates for generation have merit *worse* than any solution found so far. By that time the algorithm will have generated all simplest solutions.) The point is that once a bounded diagonal strategy has found one simplest solution then it has either generated all other simplest solutions or it very nearly has. More generally when two candidates are tied for best merit, diagonal strategies typically generate one soon after the other.

Typical of the search spaces and of the heuristic functions that can be constructed for problem domains in artificial intelligence are ones where the eventually-found solution derivation is obtained from candidate sub-derivations which at some stage look worse (as viewed from the heuristic function) before looking better again. More precisely, if $D^*$ is the eventually-found solution, then more often than not there exist sub-derivations $D_1$ and $D_2$ selected and generated before $D^*$ (where $D_1 \subset D_2 \subset D^*$) which are such that the heuristic value of $D_2$ is not better than that of $D_1$, that is, $h(D_1) \leqslant h(D_2)$. In such a case (no matter how large an $\omega$ is used to weight the heuristic function), if no other extension of $D_1$ looks better than $D_2$, then a $g+\omega h$ search strategy will turn away from further exploration of extensions of $D_1$ to an exploration of any alternatives which look equally meritorious to $D_1$. The search strategy eventually returns to generate $D_2$ and then continues generating extensions of $D_2$ until some further extension has heuristic value no better than $D_2$. The upwards diagonal variant of diagonal search is useful for dealing with the case where $h(D_1)=h(D_2)$, but is unable to deal appropriately with the more usual case where $h(D_1)<h(D_2)$. The trouble with diagonal searches is that they have no persistence. They abandon a line of approach as soon as things look bad. They are short-sighted and incapable of bringing long-term objectives to bear on the evaluation of short-term alternatives.

Another more technical problem arises when we seek to apply diagonal strategies to the finding of arbitrary solutions in and-or/theorem-proving graph search spaces: what measure of complexity $g$ should be employed by the evaluation function $g+\omega h$? The arguments in favour of using $g+\omega h$

carry no indication of how $g$ should be measured. In the case of path-finding problems the alternatives are few in number. In the more general case we have to choose between level, size, symbol complexity and various kinds of sum costs and max costs. The choice, for instance, between level and size has an important effect on the resulting behaviour of the corresponding $g+\omega h$ strategy.

In fact, for resolution theorem-proving problems, when size is used to measure complexity and when number of literals in the derived clause is used to measure heuristic value, the resulting $g+\omega h$ strategies display intolerable inefficiency. The problem is most acute when a solution of least size contains subderivations of contradictory unit clauses of nearly equal size. Although the solution can be generated as soon as the two units have been generated, the search strategy must wait until the candidate solution becomes a candidate of best merit. In most cases this will not happen before the program has exceeded pre-assigned bounds of computer space and time. With the same heuristic function, the situation would seem to be more satisfactory when level is used to measure complexity. Whenever a unit clause is generated, an immediate attempt is made to resolve it with all other previously generated units, because any resulting solution derivation would automatically have the same merit as the unit clause just generated (or better merit in the case of upward diagonal searches). But other objections apply to using level as a measure of complexity. For a fixed amount of computer time and space a diagonal search strategy, treating level as complexity, generates fewer derivations per unit of time (or spends less time evaluating each clause it generates) than a diagonal strategy using size. By the same measure, symbol complexity is still more efficient than size. We shall further elaborate upon this theme in the following section.

It would be too lengthy to outline here some of the concrete alternatives we envisage for the construction of more efficient search strategies. We remark only that in the case of theorem-proving problems we favour strategies which seek to minimise the additional effort involved in generating a solution. Such strategies employ look-ahead and resemble depth-first strategies as much as they do diagonal ones.

## SOME ARGUMENTS FOR SIMPLICITY PREFERENCE

Our arguments against the employment of diagonal search for finding arbitrary solutions are not arguments against the use of complexity for guiding the order in which alternatives are explored by an intelligent search strategy. On the contrary we believe that the complexity of candidate derivations is one of the most important factors which search strategies can employ in order to increase efficiency. For this purpose, level and max costs are misleading measures of complexity, size is more appropriate than level, and symbol complexity is more adequate than size. The arguments which support symbol complexity are both empirical and theoretical.

**Empirical arguments**

The increased efficiency contributed by the use of *ad hoc* heuristics in resolution theorem-proving programs has been an unmistakable, and unexplained, phenomenon. We shall argue that the most important of these heuristics are disguised first approximations to a more general principle of preference for small symbol complexity. Assuming that the argument succeeds we will have an *a priori* case for believing that the use of a single strategy of preference for small complexity will result in improved efficiency on the order of that contributed by the *ad hoc* heuristics. In fact, we will have grounds for believing more. If the use of symbol complexity unifies, subsumes and subjects to a common unit of measure otherwise diverse heuristics, and if such heuristics have a positive effect on efficiency, then we might expect an even greater improvement to result from an application of a unifying simplicity preference strategy. Such a theory could be tested by subjecting it to experiment.

It can be argued in fact that such an experiment has already been performed by Siklossy and Marinov (1971). Their program for proving theorems in rewriting systems employs a search strategy which is very nearly symbol complexity saturation. The statistics they have collected for a large number of problems substantiate the thesis that a general-purpose, exhaustive, symbol complexity preference strategy outperforms many special-purpose heuristic problem-solving programs. What is additionally impressive about their results is that they use a complexity saturation strategy which might be improved further by extending it to a bounded bi-directional diagonal search – this despite the arguments against diagonal search for finding arbitrary solutions. We conclude that the advantages obtained by generalising and extending the *ad hoc* heuristics compensate for the inefficiencies inherent in saturation and diagonal search. It remains for us now to argue our case that many of the *ad hoc* heuristics can be interpreted as first approximations to some variation of simplicity preference. We shall limit our attention to heuristics used in resolution theorem-proving programs.

(1) *Unit preference* (Wos *et al.* 1964). This heuristic gives preference to resolution operations which have a unit clause (one containing a single literal) as one of the premises. It behaves somewhat like the bounded diagonal search strategy which measures complexity by level and measures the heuristic value $h(D)$ of a derivation $D$ by the number of literals in the derived clause. Until recently this seemed like an adequate, theoretically justified substitute for unit preference. The more radical preference for units involved in the original heuristic strategy seemed to be unjustified. However, if we measure complexity by symbol complexity, and heuristic value by the symbol complexity of the derived clause, then the resulting bounded diagonal strategy exhibits a behaviour significantly more like that of the unit preference strategy. The diagonal strategy has the advantage that it does not depend on the initial specification of an arbitrary level bound,

within which unit preference is employed, and outside of which it is not employed.

The argument here is not intended as a one-sided attack against unit preference. On the contrary, the utility of the unit preference strategy is widely recognised and is not in dispute. Indeed, we take the usefulness of unit preference as an argument for the use of complexity, and of symbol complexity more particularly, in search strategies for theorem-proving problems.

(2) *Function nesting bounds* (Wos *et al.* 1964). This heuristic preserves all clauses which have function nesting less than some initially prescribed bound and deletes those which have nesting greater than or equal to the bound. It is most effective when the user supplies a bound which is small enough to filter out a great number of useless possibilities and large enough to include all sentences in some tolerably simple solution. Unfortunately the kind of knowledge necessary for obtaining such bounds is usually nothing less than a knowledge of some simple solution and of the maximal function nesting depth of all clauses contained in the solution. In other cases, the reliability of user-prescribed function nesting bounds would seem to be highly precarious and without foundation.

Despite these reservations, the uniform application of consistently small nesting bounds has been exceptionally successful for obtaining solutions to a large number of problems (see, for some examples, Allen and Luckham (1969)). But Siklossy and Marinov (1971) make the same observation – that simple problems have simple solutions – and obtain efficient results with a graded simplicity preference strategy which does not involve the employment of some initial, arbitrary, function nesting bound. Since degree of function nesting can be regarded as an approximation to symbol complexity, we can regard the employment of function nesting bounds as an approximation to a strategy of simplicity preference which measures complexity by symbol complexity.

(3) *Preference for resolution operations involving functional agreement.* (This heuristic has been discovered independently by Isobel Smith, Donald Kuehner and Ed Wilson, at different times in the Department of Computational Logic. It has probably been noticed elsewhere with similar frequency.) Given a clause containing a literal $P(f(y))$, preference is given, for example, to resolving it with a clause containing $\bar{P}(f(t))$ rather than with one containing $\bar{P}(x)$. This heuristic generalises to one which prefers resolving pairs of literals which have many common function symbols in preference to others which have fewer common symbols. Such a heuristic favours the resolution operation which involves the least complicated unifying substitution and therefore the least complicated resolvent as well. The operational effect of this heuristic can be obtained by preference for small symbol complexity.

(4) *Other heuristics*, such as preference for equality substitutions which simplify an expression in strong preference to those which complicate it, are

obvious and direct applications of the principle of preference for least symbol complexity.

### Theoretical arguments

Arguments for the use of symbol complexity can be obtained from entirely theoretical considerations.

(1) We consider, as an argument for the use of simplicity preference, the organising and simplifying effect which such search strategies have in a theory of efficient proof procedures. Thus, for example, we count in its favour the fact that symbol complexity provides a means for constructing complete search strategies for search spaces with infinite or-branching. The assumption that search strategies generate simple proofs in preference to more complex ones is necessary for a formal justification of the intuitive conviction that deletion of tautologies and subsumed clauses increases the efficiency of resolution theorem-proving programs (Kowalski 1970, Meltzer 1971). A greater improvement in efficiency can be demonstrated when complexity is measured by size rather than by level. Still greater improvement follows when symbol complexity is used instead of size. Similar assumptions about the employment of simplicity preference strategies are necessary for the proofs of increased efficiency applied to linear resolution and SL-resolution (Kowalski and Kuehner 1971). In all of these cases, the assumption that search strategies prefer simple proofs to ones more complex is a prerequisite for all proofs of increased efficiency. We regard the unavoidability of such an assumption in a theory of efficient proof procedures as an argument in favour of the implementation of such strategies.

(2) Assume that, within a given search space, all derivations have the same probability of being a solution derivation. Assume that, on the average, all symbol occurrences occupy the same storage space and require the same amount of processing time. Then, for a fixed finite quantity of space and time, symbol complexity saturation generates more derivations and therefore is more likely to generate some solution than is any other search strategy for the same search space. To the extent that not all derivations have equal probability of being a solution, and to the extent that such information can be made available to the search strategy, symbol complexity needs to play a role subordinate to such knowledge. None-the-less, the point of our argument stands. Other considerations being equal, a search strategy which generates derivations containing few symbol occurrences, in preference to others which contain more, maximises the probability of finding a solution derivation within fixed bounds on computer time and space.

(3) Symbol complexity is seemingly a most obvious example of the kind of purely syntactic concept which has come under increasingly more vigorous attack in the community of artificial intelligence workers. The argument seems to be that emphasis upon uniform, general-purpose, syntactic notions is detrimental to, and perhaps even incompatible with, progress in com-

municating special-purpose, problem-dependent semantic and pragmatic information to problem-solving programs. It is not our intention to wage here a lengthy counter-attack against this position. We shall instead outline a counter-proposal which aims to reconcile the opposing sides of this dispute. We shall argue that special-purpose, domain-dependent semantic and pragmatic information is inevitably reflected in the syntactic properties of sets of sentences. Among the most important of these properties is symbol complexity.

Suppose that in a given language, some definable concept occupies a semantically-important and pragmatically-useful role for solving problems posed in the language. Unless the concept is given a name, its definition in terms of other named concepts may be exceedingly complex. The data processing of propositions about the concept will be similarly complicated and consuming both of space and time. In order that data processing be made as efficient as is necessary to reflect the importance and utility of the concept, the concept is given a name, which serves the effect of lowering its symbol complexity as well as the complexity of propositions concerned with the concept.

What can be argued about concepts can also be argued about propositions. Suppose that some derivable proposition occupies a centrally-important semantic and pragmatic role for solving problems posed in the language. Then the more important this role, the more accessible the proposition needs to be made to the problem-solver. It will not do if a useful proposition needs to be re-derived in a complicated and time-consuming way every time it needs to be applied. In order that data processing be done efficiently, axioms are chosen and readjusted so that important and useful facts are accessed by simple derivations.

To summarise, our argument has been that, in order for important and useful concepts and propositions to be stored and processed efficiently, it is necessary that they be associated with small symbol complexity. We do not claim that symbol complexity is the only way of communicating useful, special purpose information to a problem-solving system. We do maintain, however, that it is one of the most important ways. This is not to argue that symbol complexity has been used as we suggest it be used. Indeed practice has been totally unconcerned with specifying languages whose syntax reflects their pragmatics. Perhaps it is an argument against this practice that is one of the more important and useful contributions of those who, otherwise, seem to be arguing against reliance on general-purpose, syntactic methods in problem-solving systems.

**Acknowledgements**

This research was supported by a Science Research Council grant to Professor B. Meltzer. Additional support came from an ARPA grant to Professor J. A. Robinson, during a visit to Syracuse University, as well as from a French Government grant to Dr A.

Colmerauer, during a visit to the University of Aix-Marseille. Thanks are due, for their useful criticism, to my colleagues, Bob Boyer, Pat Hayes and Ed Wilson, in the Department of Computational Logic, and to Mike Gordon and Gordon Plotkin in the Department of Machine Intelligence.

## REFERENCES

Allen, J.R. & Luckham, D. (1969) An interactive theorem-proving program. *Machine Intelligence 5*, pp. 321–36 (eds Meltzer, B. & Michie, D.) Edinburgh: Edinburgh University Press.

Chang, C.L. & Slagle, J.R. (1971) An admissible and optimal algorithm for searching and-or graphs. *Art. Int.*, **2**, 117–28.

du Feu, D. (1971) An application of heuristic programming to the planning of new residential development. *Department of Computational Logic Memo No. 49*, University of Edinburgh.

Hall, P.A.V. (1971) Branch-and-bound and beyond. *Proc. Second Int. Joint Conf. on Art. Int.*, pp. 941–50. The British Computer Society.

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

Kowalski, R. (1969) Search strategies for theorem-proving. *Machine Intelligence 5*, pp. 181–201 (eds Meltzer, B. & Michie, D.) Edinburgh: Edinburgh University Press.

Kowalski, R. (1970) Studies in the completeness and efficiency of theorem-proving by resolution. Ph.D. Thesis, University of Edinburgh.

Kowalski, R. & Kuehner, D. (1971) Linear resolution with selection function. *Art. Int.*, **2**, 227–60.

Kowalski, R. (1971) An application of heuristic programming to physical planning. *Department of Computational Logic Memo No. 41*, University of Edinburgh.

Meltzer, B. (1971) Prolegomena to a theory of efficiency of proof procedures. *Artificial Intelligence and Heuristic Programming*, pp. 15–33. Edinburgh: Edinburgh University Press.

Michie, D. & Ross, R. (1969) Experiments with the adaptive Graph Traverser. *Machine Intelligence 5*, pp. 301–18 (eds Meltzer, B. & Michie, D.) Edinburgh: Edinburgh University Press.

Michie, D. & Sibert, E.E. (1972) Some binary derivation systems. *Department of Machine Intelligence Internal Report*, University of Edinburgh.

Nilsson, N.J. (1968) Searching problem-solving and game-playing trees for minimal cost solutions. *IFIPS Congress preprints*, H125–H130.

Nilsson, N.J. (1971) *Problem-solving Methods in Artificial Intelligence*. New York: McGraw-Hill.

Plotkin, G.D. (1972) Building-in equational theories. *Machine Intelligence 7*, paper no. 4 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

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

Pohl, I. (1970) Heuristic search viewed as path-finding in a graph. *Art. Int.*, **1**, 193–204.

Pohl, I. (1971) Bi-directional search. *Machine Intelligence 6*, pp. 127–40 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Siklossy, L. & Marinov, V. (1971) Heuristic search vs. exhaustive search. *Proc. Second Int. Joint Conf. on Art. Int.*, pp. 601–7. The British Computer Society.

Wos, L.T., Carson, D.F. & Robinson, G.A. (1964). The unit preference strategy in theorem-proving. *Proc. AFIPS 1964 Fall Joint Comp. Conf.*, **25**, 615–21. Washington DC: Spartan Books.