C50

Proceedings of the 29th Conference
on Decision and Control
Honolulu, Hawaii • December 1990

ON THE USE OF SEARCH METHODOLOGIES IN SYSTEM IDENTIFICATION *

Marc HAEST, Georges BASTIN, Michel GEVERS and Vincent WERTZ†
Department of Automatic Control - University of Louvain
Place du Levant 3, B-1348 Louvain-La-Neuve, Belgium

Abstract - The purpose of this paper is to investigate ways in which advanced search methodologies can contribute to solve the identification problem. Several such methods are reviewed with techniques that are introduced to increase their efficiency. Their applicability to the determination of the model structure that best describes sampled data is analysed at the same time. Finally, it is shown on examples that considerable savings in time can result if they are used during an identification exercise.

1. INTRODUCTION

Since a model structure is specified by a set of integer numbers, say q, the quest for a model structure that best describes sampled data can be thought of as a way of travelling through the points of the space $\mathbf{Z}_+^q$ until an element is obtained for which the corresponding model is considered "good enough" or "best".

If upper bounds on the structure indices can reasonably be assumed, one way of handling this search through the space $\mathbf{Z}_+^q$ would be to use an *exhaustive search* approach, which means estimate the models for all structures the indices of which do not exceed the upper bounds. However, this procedure could lead us to the investigation of a horrendous number of models. Therefore, at Louvain-La-Neuve, we started to develop an *expert systems* approach (Haest *et al.* 1988, 1990a) in which a set of rules is used to mimic the behaviour of human experts during an identification exercise. Our primary objective was to develop a software tool that would be able to identify a reasonably good model on most industrial data, thereby allowing us to save the considerable amount of time that is normally spent identifying models from data. ESPION, our expert system for process identification, has already achieved this objective to a large degree, but one can ask what is gained when producing programs that do "intelligent" things in the same way fallible people do. In particular, aren't we in danger of producing an expert system that fails when human experts do? For this reason, it seemed interesting to investigate which contribution advanced search techniques from *artificial intelligence* could provide us with when faced with the above mentioned problem (Haest *et al.* 1990b).

The paper is organized as follows. After having defined what is meant by search and introduced the necessary terminology and basic concepts in Section 2, we will illustrate various search methodologies and elaborate upon their usefulness when one is left with the problem of determining which model structure should be used to describe data. Simple procedures are presented in Section 3. These are used when an optimal solution is not required, but they are not guaranteed to find a solution particularly quickly. The only way to speed up those methods is to add to them some domain specific knowledge leading to the heuristic search methods of Section 4. Having briefly described these, we will then study more complicated procedures that find optimal solutions. Branch-and-bound is analysed in Section 5, while Section 6 is devoted to the so-called A^* procedure. Finally, we show in Section 7 how these procedures can be used during an identification exercise, while some typical performances are illustrated in Section 8.

2. WHAT IS SEARCH ?

Search is one of the major problem solving paradigms used in artificial intelligence and other areas of computer science such as integer programming and game theory (see for example Rich 1983, Pearl 1984, Winston 1984, Charniak and McDermott 1985, Hillier and Lieberman 1989 and Chabris 1989).

Every search procedure can be described as the crossing of a graph in which each node represents one particular state of the problem being solved and each arc represents a relationship between the states defined by the nodes it connects. One can consider the nodes as being different knowledge levels of the understanding process and the arcs can be interpreted as different available operators that can be used when attempting to progress from the current state of the solution process closer to a node that represents the completely solved problem. So, in order to succeed, a search procedure should find a sequence of operators, defining a path through the graph, starting at the node representing the initial state and going from node to node until a node representing a goal or final state is reached.

However, before solving a problem by using this approach, we must decide what are the states and define what are the operators. We already spoke about the identification loop as if it were a way of travelling through the points of the space $\mathbf{Z}_+^q$ in which the states merely represent model structures. In what concerns the operators, the simplest way one can think of a movement from one model structure to another is simply by adding or deleting one parameter at a time. With these two elementary operations, we are able to go everywhere in our space no matter where the starting model structure is in that space. Just to give a very simple example, let us assume that our search is restricted to pure autoregressive model structures of order no greater than 4 and with no more than 3 parameters. Further assume that our search is restricted to compact model structures, which means that we do not want to consider model structures with missing parameters between the first and the last nonzero parameters in the autoregressive polynomial. Then, the graph of our search space can be depicted as in Figure 1. Here, the notation i-j has been used to characterize autoregressive model structures of the form $A(z^{-1})\,y(t) = e(t)$ where $y(t)$ is the sampled signal, $e(t)$ is a noise driving term and $A(z^{-1})$ is a polynomial in the backward shift operator z^{-1}:

$$A(z^{-1}) = 1 + \sum_{k=i}^{j} a_k z^{-k}$$

with $1 \leq j \leq 4$, $i \leq j$ and $0 \leq j-i \leq 2$. When $i = j$, which means that the structure contains only one parameter, a single number has been used to indicate its position in the autoregressive polynomial. For example, adding one parameter to the model structure 2-3 gives us the structures 1-3 or 2-4, depending on the place where the parameter is appended in the autoregressive polynomial. Similarly, deleting one parameter of the same model structure gives us either 2 or 3.

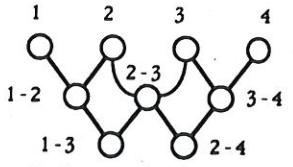

Figure 1: A very simple search graph.

Of course, our search procedure should not allow itself to cycle in the graph. When cyclic paths are forbidden, graphs become trees. The tree of Figure 2 is made from our graph in Figure 1 by following each possible path from a starting model structure, 3-4 in this particular case, until it runs into a place already visited.

*The results presented in this paper have been obtained partially within the framework of the Belgium Program on Concerted Research Actions and on Interuniversity Attraction Poles initiated by the Belgium State, Prime Minister's Office, Science Policy Programming, and partially within the framework of the "Programme FIRST de Formation et d'Impulsion à la Recherche Scientifique et Technologique du Ministère de la Région Wallonne". The scientific responsibility rests with its authors.

†Chercheur qualifié FNRS.

CH2917-3/90/0000-3182$1.00 © 1990 IEEE

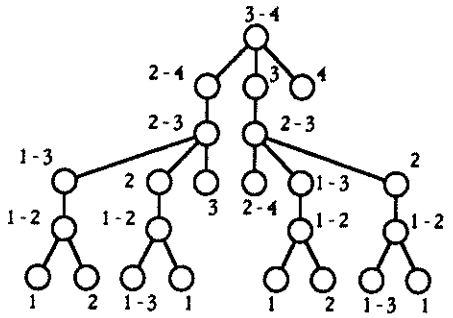

Figure 2: A very simple search tree.

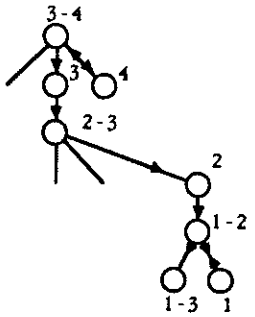

Figure 3: Depth-first search.

We also assume that some model structures exist somewhere, in the tree or in the graph, that could satisfy us. The problem is to find them: not an easy task, since we do not know beforehand where they are and what they look like. In the next four Sections, we present several procedures that have been developed to solve the riddle and discuss their applicability in system identification.

3. BASIC PROCEDURES

Generate-and-test, the simplest of the approaches one can find in the literature on artificial intelligence, consists of the following three steps:

1. Generate a possible solution,
2. Test to see if it is actually a solution,
3. If not, repeat from 1. until a solution has been found.

As simple as it is, generate-and-test already illustrates two aspects of search methodologies: the problem of generating a solution on one hand and the problem of recognizing that a solution has been found on the other. To test whether a solution has been encountered, one usually compares some properties of the chosen model structure to see if they are close to those that are thought to characterize either the data generating process or a "good enough" model.

Depending on what properties are chosen to enter the test criterion, the problem may thus have more than one solution. For example, we could be content with overparametrizations of the data generating process, which means all model structures that include the poles and zeros of the data generating process. In cases where the data have been generated by an ARARX model structure, which is more expensive to identify than ARX model structures are, we could accept any equivalent ARX overparametrization. In other circumstances, we could be interested only by the data generating process itself, introducing then the concept of an optimal solution. Now, if the investigation of admissible model structures is done randomly in the state space, there will be few guarantees that a solution, if one exists, will ever be encountered. We already prevented repeats by excluding cyclic paths. Now, if the investigation of admissible model structures is conducted systematically, then the generate-and-test procedure will encounter a solution eventually, provided one exists and we are able to recognize it. For the moment, we will consider that these two assumptions are satisfied.

Depth-first search is the most straightforward way to implement generate-and-test systematically. The idea is to pick the same operator at every node expanded and to go on forward from that choice. Other alternatives are ignored as long as there is hope of finding a solution using the original choice. Assuming that the structure of the generating process is 1-3 and using the convention that the alternatives are tried clockwise in the search tree, the model structures that are explored before reaching the goal state are shown in Figure 3.

As can be seen, the procedure generates or expands explicitely only the parts of the tree that it decides to investigate. Indeed, the state space is usually too large and most of it need never be visited. However, if the model structures represented by nodes 4 and 1 were gateways to vast subtrees instead of short blind alleys, depth-first search could slip past the level at which the data generating process appears and waste incredible energy in exhaustively exploring parts of the tree lower down.

One cure is to use *breadth-first search*, the essence of which is to look for the data generating process among all the states at a given level before exploring lower levels. The procedure is illustrated in Figure 4 where the model structures encountered before reaching the data generating process are indicated.

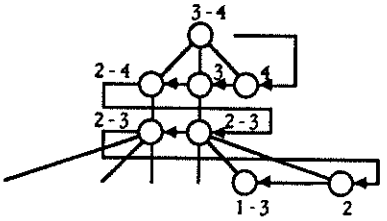

Figure 4: Breadth-first search.

As can be seen, breadth-first search will work even in the case of infinite trees. It is also clear that breadth-first search finds the shortest sequence of operators that starts from the initial model structures to the goal structure. However, one can ask whether the number of elementary operations required before reaching the solution is really what matters. And obviously, it could be that one of the paths leading to an admissible solution runs across many, easy to estimate, ARX model structures, while another crosses only a few ARMAX model structures that are far more expensive to estimate.

On the other hand, the number of elementary operations that solve the problem is not equivalent to the total number of model structures investigated during the whole search process. So, we have to worry about the overall performances of the final model structure versus the total amount of time and work needed to find it. It turns out that, even from this point of view, depth-first search and breadth-first search are of little practical use since, if they will find a solution eventually, this may take a very long time depending on the size of the search space. The only way to speed up those methods is to add to them some domain specific knowledge leading to heuristic search methods.

4. HEURISTIC METHODS

Better efficiency can be achieved when it is possible to rank the operators according to some heuristic measure of the remaining distance to a goal. A heuristic is any trick that helps discovery. For example, in the *shortest-route problem*, where it is wanted to go from a starting place to a goal along the shortest path through a net of roads, straight line distances to the goal could be used as a heuristic measure of the remaining distance. If the available moves are ordered according to this criterion and if the most promising ones are explored first, then it is natural to expect some useless work will be spared.

When moving through a tree, *hill climbing* proceeds as in depth-first search, except that the moves are ordered according to the mechanism just mentioned. After having expanded the root node, the child node with the best value for the heuristic function is selected at level one to be expanded further. Then, the process is restarted at lower levels until a solution is encountered somewhere.

In *best-first search*, forward motion is restarted from the best open node so far, no matter where it is in the partially developed tree. Note however that, if best-first search is more likely to find a solution with less work than other methods, one should never forget that those procedures are normally not appropriate when

an optimal solution is desired since they do not ensure that such a point will be encountered. The heuristic function is simply used in an attempt to minimize the number of nodes expanded during the search process.

By far the most important characteristic of heuristic search methods is that their performance varies directly with the accuracy of the heuristic function used. When estimating models from data, the variance of the prediction errors could seem to be the simplest available heuristic measure of what can still be gained. However, this constitutes a very optimistic overbound, since a zero variance for the prediction errors is certainly the best we can hope. A better estimate could be the difference between the prediction error variance of the model and that of any overparametrization of the data generating process. A big model structure with many parameters in each polynomial could be used for this purpose but still constitutes an overbound. Unfortunately, this is a drawback rather than being an advantage, as we will see in the next Section.

Note also that hill climbing and best-first search behave in the same way when this heuristic is used. This is so because the prediction error variance cannot increase with model dimension provided the models are hierarchically nested, which means that models in higher dimensions are obtained by adding parameters to models in lower dimensions. Therefore, the procedure will waste time to explore the far bottom of the tree. A simple way to prevent this phenomenon is to use a statistical test to stop forward motion, at least temporarily, along one path as soon as the prediction error variance ceases to decrease significantly (Ljung 1987, Haest *et al.* 1988, Söderström and Stoica 1989, Haest *et al.* 1990a, 1990b).

5. FINDING OPTIMAL SOLUTIONS

In this Section, we continue to investigate search methodologies but, from now on, we will pay attention to the optimality of the solution. Step by step, we will introduce several techniques to do the work more efficiently and, finally, we will bring together those ideas to give a general formulation of the so-called A^* procedure.

One way to find an optimal solution with less work than an exhaustive search is by using *branch-and-bound*. This technique allows considerable flexibility when designing a specific algorithm for the problem at hand and variations of it have been applied with success to a wide variety of operations research and artificial intelligence problems and are especially well known for their application to integer programming problems.

In its basic general form, the essence of the technique is the following. During search, there are many incomplete paths waiting for further investigation. The best one is expanded one level further, creating as many incomplete paths as there are children. These new paths are then considered together with the remaining old ones. Once more, the best one is expanded further. This repeats until at least one goal state has been reached somewhere in the tree while the best incomplete path is worse than any complete path. When this happens, the best path reaching a goal state is certain to be optimal, since the best path was always chosen for expansion.

As one can see, the procedure behaves like best-first search. Note however that, not to speak of the more elaborate termination criterion, there is a fundamental difference between branch-and-bound and best-first search. Indeed, in the former, the function used to order the available moves consists of true costs accumulated from the starting node along the different paths rather than being some heuristic measure of the remaining distance to a goal.

In system identification, nothing prevents us from using the difference between the prediction error variance of the models and the output signal variance as a measure of what has actually been gained but, as already explained, the problem with this function is that we are trying to maximize an ever increasing function instead of minimizing an ever increasing one, as in the case of the shortest-route problem.

Now, basic branch-and-bound can be improved further by using guesses about distances remaining to a goal state together with costs already accumulated along the paths. To do this, one generally introduces a new function $f(x)$ which is defined as the sum of $g(x)$, the true costs accumulated along the path from the starting node, plus $h(x)$, a heuristic estimator of the remaining distance to the goal state. It does certainly make sense to use the function $f(x)$ to rank the paths contending for further consideration since, in this way, the evaluation is based both on the work that has already been accomplished and on additional costs expected on the way that separates us from the final goal.

However, if a bad overestimate somewhere along the true optimal path may cause us to discard that path forever, underestimates cannot cause the right path to be ignored. It can be shown easily that the procedure always terminates with an optimal solution provided the estimator used is an admissible heuristic evaluation function, which simply means a function that is always less than or equal to the estimated quantity but never greater than that. In the shortest-route problem, for example, the straight line distance to the goal state considered as an estimator of the remaining distance has this nice property.

Needless to say, it would be very nice to possess such a pretty heuristic evaluation function when comparing model structures. One simple way to construct a heuristic is to proceed as in validation and compare some properties of the estimated model structures to check if they are close to those that are considered to characterize an hypothetical solution. In ESPION (Haest *et al.* 1988, 1990a), model structures are ordered according to a quality index that incorporates a number of validation criteria including statistical tests on the whiteness of the prediction errors and their independence from past sampled input signals. The quality index of a particular model is simply incremented by one each time one of the statistical tests is satisfied. Obviously, to make sure it is an admissible heuristic, properties that could be satisfied by other model structures than that of the data generating process should not enter the quality index. Then, the more tools will be used, the more informed the evaluation will be.

Unfortunately, it turns out that the quality index can hardly be used as a heuristic function. First of all, if we take the quality index of a particular model structure as $h(x)$, it cannot be consistently added to the variance of the prediction errors associated with the same model structure, taken as $g(x)$, to form the function $f(x)$. Note that this problem could be solved, at least theoretically, if the prediction error variances entering $g(x)$ were replaced by some kind of binary decision variable. Another problem is that good heuristic evaluation functions should be easily computable and the statistical tests that enters the quality index are precisely all but cheaply checked. Certainly, it does not make sense to spend as much time evaluating the quality index of a model structure as it would take to estimate 25 others, for example. Those are the reasons why the quality index as defined above has not been used in ESPION to guide the search process. It is used only to rank some of the best model structures obtained after completion of the search procedure.

So it seems that we are sentenced to be content with blind, basic branch-and-bound. Theoretically, this is not a problem since a zero guess is the less informed underestimate one can find on the market. Note however that the arguments developed above heavily depend on assumptions made about the facility with which model structures are estimated. Some kind of trade-off must exist somewhere for, if it is possible to estimate many ARX model structures at the same time one needs to compute the quality index of such a model, this certainly does not hold for ARARX or ARMAX model structures. The estimation of model structures with noise polynomials is so expensive that one should never rely only on an ordinary branch-and-bound search to identify such structures. Here, more knowledge is crucially wanted to direct the search competently and decrease the risk of making useless time consuming false movements.

This leads us to *pruning*, the ultimate refinement one can supply search procedures with. The principle, which finds many applications in methods that have been developed in the context of game theory such as *minimax search* and *alpha-beta pruning*, tells us that if we are sure to be beaten along one track, it does not make sense to waste time to find out in how many ways or how badly in the worst case (Winston 1984). So, if it can be recognized that a move is bad, no matter how the whole game continues, the tree should be pruned at this point once and for all, allowing us to save computational effort. The sooner the cut-off will appear, the more energy will be spared.

One way the principle could be applied when comparing model structures is the following. If a model structure is found a relative of which is significantly better, be it a child or a parent, then the tree should be definitely closed at this point. Moreover, all underparametrizations, which means all model structures that can be obtained by deleting parameters from the closed structure, should be removed for ever from the set of admissible structures contending for further consideration. Later on, each time one of these model structures will be encountered, we will be able to prune

the followed path immediately. It will never be needed to estimate these model structures. On the other hand, when relatives that do not differ significantly are encountered, those of higher dimension and their overparametrizations should be pruned also, unless there is some evidence that we are still far away from the prediction error variance of the data generating process.

6. A^*

The A^* *procedure* is a branch-and-bound search, with an estimate of the remaining distance to a goal state, combined with the application of any particular domain specific technique that allows to prune the tree judiciously. If the estimate of remaining distance is a lower bound on the actual distance, what is called an admissible heuristic function, then A^* produces optimal solutions.

Let us see how it works. Assume that the objective function is to be minimized and that an upper bound $f_u(x)$ on the best value of the objective function has already been computed. Usually, $f_u(x)$ is the value of the objective function associated with the best admissible solution identified so far. This solution is sometimes called the *incumbent solution.*

The set of all remaining admissible solutions is first divided into several subsets. Then, a lower bound $f_l(x)$ is obtained for the value of the objective function within each of these subsets. Those subsets whose lower bound exceeds the current upper bound are excluded from further consideration. Others are discarded either because they have no more admissible solutions or because their best solution is already known so that we do not have to worry about the rest of that subset. A subset that is excluded from the inquiry for any of these reasons is said to be *fathomed.* After the appropriate subsets have been pruned, one of the remaining subsets is chosen according to a branch rule to be divided further into several subsets. Their lower bounds are obtained in turn and used as before to prune some of them. Another subset is selected from the remaining ones to be split again, and so on. This process is repeated iteratively until an admissible solution the objective function of which is no greater than all the lower bounds of the remaining subsets is found. We are sure that this solution is optimal since none of the remaining subsets can contain a better solution. The procedure is summarized below (Hillier and Lieberman 1989):

> *Initialization step*: Set $f_u(x) = \infty$. Begin with the entire set of solutions as the only remaining subset and apply just the bound step, the fathoming step and the optimality test below.
>
> *Branch step*: Use some branch rule to select one of the remaining unfathomed subsets and split it into new subsets.
>
> *Bound step*: Obtain a lower bound $f_l(x)$ on the value of the objective function for each new subset.
>
> *Fathoming step*: Prune each new subset if
>
> - *Fathoming test 1*: $f_u(x) \leq f_l(x)$,
> - *Fathoming test 2*: The subset is found to contain no admissible solutions,
> - *Fathoming test 3*: The best admissible solution in the subset has been identified so that $f_l(x)$ corresponds to its objective function value. If this situation occurs and $f_l(x) < f_u(x)$, then reset $f_u(x) = f_l(x)$, store this solution as the new incumbent solution, and reapply the first fathoming test to all remaining subsets.
>
> *Optimality test*: Stop when there are no remaining unfathomed subsets. Otherwise, return to the branch step. If there is no incumbent solution, that is if $f_u(x)$ still equals ∞, then the problem possesses no admissible solutions. Otherwise, the current incumbent solution is optimal.

Note that the branch and bound steps, which give us considerable freedom when implementing domain specific procedures, have important implications on the overall performances of the method. As it is naturally expected, the two most popular branch rules used to select the subset that will be split further are the best bound rule and the newest bound rule. When the best bound rule is used, the subset having the most favorable bound is selected, while the most recently created unfathomed subset is retained when the newest bound rule is used. If several subsets have been created at the same time, the one with the best bound is preferred.

7. APPLICATION

We now investigate how the techniques that have been described in the previous Sections can be used during an identification exercise. The basic assumption made here is that the "good enough" model structures we are looking for actually exist somewhere in our working space. Roughly speaking, this is equivalent to postulate the existence, in the set of admissible model structures, of a good approximation of an hypothetical data generating process which constitutes, together with all its overparametrizations, the set of models we could be content with.

It is also assumed that we are provided with statistical tests allowing us to decide whether or not a given model structure is significantly better than another. Remember that we do not have to theorize too much about the appropriateness of those tools and which confidence level should be used. We just need some heuristic in order to help discovery and common sense rules of thumb could be used as well.

The problem is then to extract an optimal solution from the set of "good enough" model structures just defined. In this context, the model structure with the best prediction error variance all the underparametrizations of which are significantly worse, while none of its overparametrizations is significantly better, will be considered as the optimal solution. With this definition, one way to use branch-and-bound would be to start in the state space with the model structure of highest dimension. Doing this, we are sure to start with an overparametrization of the model structure we are looking for. Now, starting from this structure, we are also certain to be able to generate all admissible structures in the state space only by deleting parameters from the root.

Then, the state space can be partitioned simply by generating the children of the root. Clearly, those children are of lower dimension than the root node. Therefore, their prediction error variances are all greater than the one associated to their parent node. Moreover, those prediction error variances certainly constitute lower bounds on the best value of the objective function that can be achieved in their respective subsets.

Children which are significantly worse than their parents can be definitely excluded from further investigation together with all their underparametrizations. They can be removed safely from the set of unexplored admissible structures while the other children, those that are not significantly worse that their parents, are kept in the arena. At the next step, the unexpanded model structure with the best prediction error variance is selected to be partitioned further and the whole process continues.

An overestimate of the objective function is obtained the first time a dead-end is encountered along one path. This occurs when a node is found all the children of which are either significantly worse or already excluded from the set of admissible structures. Such a node represents an incumbent solution, a candidate for the optimal solution. The process continues from the unexpanded model structure with the best prediction error variance but, in the sequel, each time a better solution is encountered, it replaces the current incumbent solution and the upper bound on the objective function is modified accordingly.

The procedure can be stopped as soon as at least one incumbent solution has been found and all the remaining path extremities have been pruned or have greater prediction error variances than the one associated to the incumbent solution.

Obviously, the major drawback with this basic implementation is that we are running the risk of estimating many overparametrizations of the optimal model structure before it can be discovered. So, the computational work spared at lower dimensions by pruning has been replaced by a loss of energy at higher dimensions. We thus need a mechanism to speed up the whole process.

One cure would be to start from lower dimensions. However, if the root model structure is chosen in lower dimensions, we will be in danger of missing the optimal solution for, if the starting structure is not an overparametrization of the optimal solution, we will not be able to generate it. A more interesting possibility is to first map the landscape at "equidistant" nodes by adding or deleting many more than one parameter at a time. Such a simple mapping is illustrated in Figure 5 where bold lines have been used to indicate model structures that do not differ significantly from

each other. If the state space given there is compared to the one given in Figure 1, it can be seen that they both contain the same number of nodes but that the former represents many more admissible model structures than the latter. A quick search in such a "reduced" state space together with the use of statistical tests can give valuable information on subsets in which the "best" model structure should be looked for.

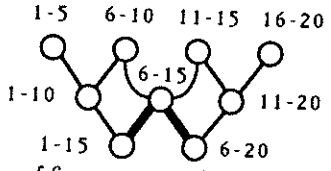

Figure 5: A map of five-parameter "equidistant" autoregressive model structures.

For example, in our case, one can reasonably suspect 6-15 to be an overparametrization of the optimal structure since its prediction error variance does not decrease significantly when it is filled up with more parameters. Therefore, the procedure described above can be started from this node, by deleting one parameter at a time, and the subsets that could be generated from 1-10, 11-20, 1-15 and 6-20 can be discarded forever. Note that this technique can be interpreted as a way to delimit the frontier between the subset containing overparametrizations of the structure we are interested in on the one hand and the rest of the state space on the other hand.

8. SIMULATIONS

The branch-and-bound search with pruning described in the previous Section has been tested on both simulated and industrial data sets. Our purpose in this Section is to illustrate some typical behaviours on two examples. The F-test has been used to check whether the prediction error variance σ^2 increases significantly when the number of parameters is decreased (Söderström and Stoica 1989, Haest *et al.* 1990a). The notation $n_a(\tau_1 - n_{b1})(\tau_2 - n_{b2})$ has been used to characterize two input - one output model structures of the form

$$(1 + \sum_{i=1}^{n_a} a_i\, z^{-i})\, y(t) = (\sum_{i=\tau_1}^{n_{b1}} b_{1_i}\, z^{-i})\, u_1(t) + (\sum_{i=\tau_2}^{n_{b2}} b_{2_i}\, z^{-i})\, u_2(t) + e(t)$$

while only one parenthesis has been used for one input - one output model structures.

8.1 - Simulated data: The branch-and-bound search was run on 1000 sampled data obtained from the following seven parameter, two input - one output system

$$(1 + z^{-1} + 0.5z^{-2})\, y(t) = z^{-2}(1 - z^{-1} + 0.5z^{-2})\, u_1(t) + z^{-7}(1 + 0.2z^{-1})\, u_2(t) + e(t)$$

where pseudo random binary signals with amplitude 1 (variance 1) were used for u_1 and u_2 and a Gaussian white noise with zero mean and variance 0.25 was taken for e, leading to an output signal variance of around 14.

The starting model structure was chosen to be 10 (1-10)(1-10) and its prediction error variance was computed to be 0.2494. The model structures expanded during the first step by stripping the starting model structure of five parameters at a time are shown in Table 1 together with their prediction error variances.

models	σ^2	comments
5(1-10)(1-10)	0.2520	-
10(6-10)(1-10)	1.5166	pruned
10(1-5)(1-10)	0.2507	-
10(1-10)(6-10)	0.2504	-
10(1-10)(1-5)	1.2075	pruned

Table 1: Step 1 from 10 (1-10)(1-10).

Since three of these underparametrizations do not differ significantly from the starting one, it is believed that the latter is an overparametrization of the true data generating process. The structures that differ significantly from the starting one, 10 (6-10)(1-10) and 10 (1-10)(1-5), are pruned forever and the procedure is restarted from 10 (1-10)(6-10), the child with the best prediction error variance obtained so far. The four model structures that should be expanded during the second step are listed in Table 2.

models	σ^2	comments
5(1-10)(6-10)	0.2531	-
10(6-10)(6-10)	→	already excluded
10(1-5)(6-10)	0.2517	-
10(1-10)(0-0)	→	already excluded

Table 2: Step 2 from 10 (1-10)(6-10).

In fact, only two of them need to be estimated since the others have already been excluded from further consideration. For example, it makes no sense to compute the prediction error variance associated with 10 (6-10)(6-10) since this structure is an underparametrization of 10 (6-10)(1-10), which has been pruned after the first step. Now, the best structure remaining unexplored is 10 (1-5)(1-10). It is expanded during the third step which is shown in Table 3.

models	σ^2	comments
5(1-5)(1-10)	0.2531	-
10(0-0)(1-10)	→	already excluded
10(1-5)(6-10)	→	already computed
10(1-5)(1-5)	→	already excluded

Table 3: Step 3 from 10 (1-5)(1-10).

As can be seen, the third step actually requires the estimation of only one new model structure since the others have already been computed or excluded for further consideration.

The overall behaviour of the branch-and-bound search is summarized in Table 4 where only the model structures that have been estimated are numbered (n). The number of parameters that are deleted when attempting to expand the tree at a particular node has been called the stripping factor (s_f). This number is given for each step (s), together with the parent model structure (f) and all the children that have actually been computed (models). Investigated model structures are further characterized by their prediction error variances (σ^2) and their dimensions (d). Pruned structures are indicated by † signs.

Only two new model structures appear during steps 4 to 8. After completion of the eighth step, we are unable to go further by stripping five parameters at a time without increasing significantly the variance of the prediction errors. At this scale, the best approximation of the data generating process is 5 (1-5)(6-10) since this is the only structure all the underparametrizations of which are significantly worse while none of its overparametrizations is significantly better.

During step 9, we now try to strip four parameters at a time from this latter structure. Since all chidren have to be pruned, we can jump to step 10 immediately where it is tried to strip three parameters at a time. Two good model structures are encountered there: the others have to be pruned. Then, only one uninteresting model structure is estimated during steps 11 and 12. At this stage, we are unable to go further by stripping three parameters at a time. Both 2 (1-5)(6-10) and 5 (1-5)(6-7) have all their underparametrizations significantly worse, but the former is preferred because it has the least prediction error variance. So, the best approximation of the data generating process becomes 2 (1-5)(6-10).

In step 13, we now try to strip two parameters at a time. Starting from 2 (1-5)(6-10), one promising model structure is encountered, but no new model structure appears during step 14. So, we enter step 15 where it is tried to strip one parameter at a time, and so on. After completion of the whole process, the only struc-

s	f	s_f	n	models	σ^2	d
			1	10(1-10)(1-10)	0.2494	30
1	1	5	2	5(1-10)(1-10)	0.2520	25
			3	10(6-10)(1-10)	1.5166†	25
			4	10(1-5)(1-10)	0.2507	25
			5	10(1-10)(6-10)	0.2504	25
			6	10(1-10)(1-5)	1.2075†	25
2	5	5	7	5(1-10)(6-10)	0.2531	20
			8	10(1-5)(6-10)	0.2517	20
3	4	5	9	5(1-5)(1-10)	0.2531	20
4	8	5	10	5(1-5)(6-10)	0.2541	15
5	2	5	11	0(1-10)(1-10)	0.7290†	20
6	7	5				
7	9	5				
8	10	5				
9	10	4	12	1(1-5)(6-10)	1.1298†	11
			13	5(5-5)(6-10)	1.5264†	11
			14	5(1-1)(6-10)	1.5595†	11
			15	5(1-5)(10-10)	1.2228†	11
			16	5(1-5)(6-6)	1.2240†	11
10	10	3	17	2(1-5)(6-10)	0.2555	12
			18	5(4-5)(6-10)	1.5129†	12
			19	5(1-2)(6-10)	0.5107†	12
			20	5(1-5)(9-10)	1.2208†	12
			21	5(1-5)(6-7)	0.2638	12
11	17	3	22	2(1-5)(6-7)	0.2865†	9
12	21	3				
13	17	2	23	2(3-5)(6-10)	1.2715†	10
			24	2(1-3)(6-10)	0.3844†	10
			25	2(1-5)(8-10)	1.2277†	10
			26	2(1-5)(6-8)	0.2570	10
14	26	2				
15	26	1	27	2(2-5)(6-8)	0.2572	9
			28	2(1-4)(6-8)	0.2570	9
			29	2(1-5)(7-8)	0.2574	9
16	28	1	30	2(2-4)(6-8)	0.2572	8
			31	2(1-4)(7-8)	0.2575	8
17	27	1	32	2(2-5)(7-8)	0.2576	8
18	30	1	33	2(2-4)(7-8)	0.2577	7
19	29	1				
20	31	1				
21	32	1				
22	33	1				

Table 4: The overall behaviour of the branch-and-bound search on simulated data.

ture all the underparametrizations of which are significantly worse, while none of its overparametrizations is significantly better, is 2 (2-4)(7-8), the structure of the true data generating process.

Note that only 33 model structures have been estimated. However, a few more would have been investigated if another confidence level had been chosen when applying the F-test, but what does really matters is that many more could have been generated if another strategy were used to adapt the stripping factor.

Now, since it first explores model structures in low dimensions before increasing the number of parameters, ESPION is unable to exploit the pruning principle in its current state. Run on the same data, it estimated 149 model structures before discovering the one with which the data were generated. Note however that an exhaustive search over all underparametrizations of the starting model structure would have required the estimation of 30,250 models!

8.2 - A glass tube drawing bench: The branch-and-bound procedure was also run on industrial data from a glass tube drawing bench on which human experts had already worked (Wertz *et al.* 1987). Eighteen model structures were estimated in order to identify a one input - one output ARX model between drawing speed and glass tube diameter from a 2700 sample file. After completion of the whole process, the best structure all the underparametrizations of which are significantly worse, while none of its overparametrizations seems significantly better, is 2 (4-7) with prediction error variance 1.6169: not very far from 2 (4-6), the structure chosen by the specialists with prediction error variance 1.6806. As expected, the results are generally not as clear on industrial data than on simulated ones and other validation tools could be used to order seemingly equivalent model structures. Run on the same data, ESPION needed to estimate 54 model structures to end up with the same final structure as branch-and-bound, while an exhaustive search would have required the estimation of 550 model structures.

9. CONCLUSIONS

In this paper, our aim has been to investigate ways in which advanced search methodologies from artificial intelligence and integer programming can contribute to solve the identification problem.

After having defined what is meant by search and introduced the necessary terminology and basic concepts, several search methodologies have been presented. Then it has been shown that identification exercises can effectively be based on variants of the branch-and-bound procedure and that only an easily computable admissible heuristic evaluation function is missing for us to have a true A^* procedure.

The major innovation introduced here is that a quick exploration of the state space can be made by mapping it at "equidistant" model structures by adding or deleting many more than one parameter at a time. This method, combined with the use of statistical tests, gives valuable information on subsets in which the "best" model structure should be searched for and allows to save considerable computer time. Once a smaller subset that most probably contains the "best" model has been identified, it can be mapped in turn at a lower scale and so on until a solution has been found.

Finally, the typical performances of our branch-and-bound procedure have been illustrated on both simulated and industrial data. The results are quite encouraging since, when limited to ARX model structures, the proposed search procedure already allows us to reduce significantly the number of trial structures expanded by our expert system.

REFERENCES

Chabris, C. F. (1989). *Artificial Intelligence & Turbo C.* Dow Jones-Irwin, Homewood, Illinois.

Charniak, E. and D. McDermott (1985). *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts.

Haest, M., G. Bastin, M. Gevers and V. Wertz (1988). An Expert System for System Identification. *Proc. 1st IFAC Workshop on Artificial Intelligence in Real-time Control,* Swansea, U.K., pp. 101-106.

Haest, M., G. Bastin, M. Gevers and V. Wertz (1990a). ESPION: an Expert System for System Identification. *Automatica,* Vol. 26, No. 1, pp. 85-95.

Haest, M., G. Bastin, M. Gevers and V. Wertz (1990b). On the use of Search Methodologies in System Identification. Tech. Report AP90.04. Depart. of Automatic Control, University of Louvain.

Hillier, F. S. and G. J. Lieberman (1989). *Introduction to Operations Research.* McGraw-Hill, New York.

Ljung, L. (1987). *System Identification. Theory for the user.* Prentice Hall, Englewood Cliffs, New Jersey.

Pearl, J. (1984). *Heuristics. Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, Massachusetts.

Rich, E. (1983). *Artificial Intelligence.* McGraw-Hill, New-York.

Söderström, T. and P. Stoica (1989). *System Identification.* Prentice Hall, Englewood Cliffs, New Jersey.

Wertz, V., G. Bastin and M. Haest (1987). Identification of a glass tube drawing bench. *Prepr. 10th IFAC World Congress on Automatic Control,* Munchën, F.R.G., Vol. 10, pp. 334-339.

Winston, P. H. (1984). *Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts.