Parallel Dynamic Scheduling in a Cluster of Computers

M. Damas, M. Salmerón, J. Ortega, G. Olivares, H. Pomares
Department of Computer Architecture and Computer Technology, University of Granada.
Facultad de Ciencias. Campus Fuentenueva s/n. E-18071, Granada, Spain
E-mail: {mdamas,moises,julio,gonzalo,hector}@atc.ugr.es

Abstract – The parallelization of complex planning and control problems arising in diverse application areas in the industrial, services, and commercial environments not only allows the determination of the control variables in the required times but also improves the performance of the control procedure as more processors are involved in the execution of the parallel program. In this paper we describe a scheduling application in a water supply network to demonstrate the benefits of parallel processing. The procedure we propose mixes dynamic programming with genetic algorithms, and time series prediction, in order to solve these aforementioned problems in which decisions are made in stages, and the states and control belong to a continuous space. Considering the computational complexity of these applications and the time constraints that are usually imposed, the procedure has been implemented by a parallel program in a cluster of computers, an inexpensive and wide extended platform that can make the parallelism a common alternative to tackle complex problems in many different environments.

Index Terms – Neuro-dynamic programming, evolutionary computation, parallel genetic algorithms, neural networks, clusters of computers, scheduling of water supply networks.

1 Introduction

The clusters of computers are relatively economical platforms that allow the execution of parallel applications. The advantages of such systems, based on standard hardware elements (general purpose microprocessors, personal computers or workstations, local area networks, etc.) to configure parallel architectures, become even clearer as the rate of performance improvement for these hardware components speeds up. Moreover, as the larger manufacturing volumes of the standard components allow manufacturers to amortize development costs over the higher number of units sold, it is possible to configure clusters with lower cost/performance ratios [1]. Thus, these systems are the object of a growing interest in the field of parallel processing because they confirm the parallel processing as a real choice for improving the performance of applications not only in the science and engineering fields but also in other wider industrial and commercial environments, where inexpensive platforms are required together with the availability of parallel applications, software, and tools [2].

Nevertheless, the use of clusters of personal computers or workstations as parallel platforms poses problems related to the communication latencies and the bandwidth for small packets [3] which may limit their use to coarse grain parallel applications, and need to be addressed in order to put the cluster performance nearer that provided by multicomputers or other specific supercomputing platforms. Moreover, it is also necessary to demonstrate the efficiency of the clusters of computers, not only with respect to their performances in new and real applications, but also decreasing the times required to develop and debug parallel applications by providing parallel software and tools similar to those available in the *usual* non-parallel environments.

Thus, in this paper we describe the way a cluster of computers has been applied to improve the performance in a realistic application dealing with the dynamic scheduling of a set of tanks in a water supply network. This problem arises in the global optimization of water supply systems, which presents different aspects [4] as for example, problems concerning the optimization of generation stations, the optimization of transport and distribution networks and the optimization of consumption. Of these problems, the application considered in the present paper is more related with the minimization of the exploitation cost (cost of electrical power to move the valves, the increase of the system mean-life, etc.), while meeting the physical constraints of the system and guaranteeing the required water consumption. Specifically, the problem consists of distributing the flow that the drinkable water treatment station (ETAP in spanish) provides to the different tanks of the distribution network system, such that demand is satisfied and that the tanks neither overflow nor fall bellow the established minimum water volume, while valve movements are minimized and the flow from the ETAP is kept constant. Figure 1 shows a system outline in the case of the water distribution network of the city of Granada (Spain).

Section 2 describes the hybrid procedure we propose to solve the kind of optimal control problems that include the scheduling problem here considered. Section 3 provides the details of the application of this procedure to our specific problem of scheduling in water supply networks. Section 4 describes the issues related with the parallel implementation of the procedure, while Section 5 gives the experimental results. Finally, the conclusions of the paper are provided in Section 6.

2 An hybrid procedure for dynamic scheduling.

In this section we present a technique that can be applied to problems of sequential decision making (in our application the volume of water that has to be sent to each tank at each time interval) with

uncertainty (in our application, the levels of water consumption in future time periods). The outcome of each decision is not fully predictable (the level in a tank at the end of each period depends on the volume of water that finally goes to the tank, and on the water consumption in that period) but can be estimated to some extent before making the next decision. Each decision has an *immediate cost* associated but it also affects the state in which future decisions are to be made thus having influence in the cost of future periods of time (*cost-to-go*). Thus, the obtention of a low cost solution at the present must be balanced against the undesirability of high costs in the future. This situation is, for example, similar to that of dynamic scheduling problems, where the scheduler has to be able to react to external events in order to find solutions to the problems resulting from the changes produced by those events.

The procedure proposed here can be applied when the possible states and control in each stage are to be selected from a continuous space. Moreover, the input to the system is also continuous and unknown, although it can be predicted to some extent. Figure 2 shows the situation of the module implementing the procedure with respect to the system and the corresponding variables. In this figure, the control vector in the time *i*t* is noted *Y(i)*; the vector of inputs to the system in time *i*t* is *W(i)*; and the system state vector corresponding to the previous time interval *(i-1)*t* is *X(i-1)*.

Dynamic programming [5,6] is a technique that can be applied to the kind of problems here considered, such as scheduling, inventory management, packaging and many other problems arising in fields such as operation research, control, communications, biology, etc. It provides the required formalization of the tradeoff between immediate and future costs, and considers a discrete-time dynamic system whose states change according to transition probabilities that depend on the control signal applied (decision made). Nevertheless, in some applications, the computational requirements to use dynamic programming are overwhelming because the number of states and possible controls are very large. Moreover, knowledge of the transition probabilities between states for each control applied is also required to compute the corresponding expected value at each state. This stochastic character of the problem derives from the uncertainty in the state transitions appearing, since the external input values and perturbations are not previously known and are uncontrollable.

In the last few years, the term *neuro-dynamic programming* (NDP) [7] has been coined to refer to a new methodology based on neural networks, dynamic programming, the theory of function approximation and the theory of iterative optimization. This methodology, also called *reinforcement learning* in the Artificial Intelligence literature, has been shown to be effective in coping with the two *curses* of dynamic programming and stochastic optimal control:

- *The curse of dimensionality*, which means the exponential computational complexity with the problem dimension.
- *The curse of modeling,* or the difficulty in determining an explicit model of the system to manage, the state-transition probabilities and the observed *immediate costs*.

Neuro-dynamic programming uses sub-optimal optimization methods based on the evaluation and approximation of the optimal cost-to-go function, J^*. The transition probabilities are not explicitly estimated, but instead the cost-to-go function of a given policy is progressively calculated by generating several sample system trajectories and the associated costs. Thus, Monte Carlo simulation of the system is the alternative used [7,8]. This allows the processing of systems for which no explicit models are available, because if no such model exists, it is not possible to estimate the state transition probabilities, and thus traditional dynamic programming cannot be applied.

In this paper we propose a different hybrid procedure whose modules are shown in Figure 3. As can be seen there is a Prediction module that provides the parameters of a model that allows the estimation of the system input in the next stage. It is also possible to use a Function Approximation module that will provide an approximated description of the system that makes possible an easier simulation of the system. The prediction model provided by the Prediction module, and the model of the system provided by the Function Approximation module are used by our Neuro-Dynamic Programming module to determine a set of feasible functioning points, and select the best one to be applied at the next stage.

Thus, at stage i, the controller has to determine the control vector $Y(i+1)$ for the next stage. To do this, the vector $W'(i+1)$, that estimates the vector $W(i+1)$ of inputs in the next stage, is obtained by using the model determined by the Prediction module, and the output $X(i)$ are used to define the cost function $Cost(i)$ whose minima correspond to the feasible control vector $Y(i+1)$ for the next stage. These feasible values for vector $Y(i+1)$ also llow the determination of the nexte state, $X(i+1)$ satisfying all the established restrictions. Once a set of feasible control vectors is obtained for a given stage $(i+1)$, $\{Y(i+1)_j, j=1,...,M_{i+1}\}$, it is necessary to select among them the best one to be used in the next control stage. The rank of $Y(i+1)_j$ $(j=1,...,M_{i+1})$ is determined by taking into account the immediate cost associated with its application, and the future evolution of the system.

The implementation of this part of the procedure can be done by simulation. This simulation uses the predictions of the input vectors for the following stages, $W'(i+2),...,W'(N-1),W'(N)$, from the

prediction model available at stage i. By using these predicted input vectors it is possible to estimate sets of feasible control values (by simulation and using the predicted values W') for the following stages $\{Y'(i+2)_j\ ,\ j=1,...,M_{i+2}\},...,\ \{Y'(N)_j\ ,\ j=1,...,M_N\}$, and to generate sample trajectories (as in Figure 4) in order to approximate the cost-to-go function at stage $i+1$.

The function $J'(X(i+1)_j)$, that estimates the cost-to-go function, can be obtained as

$$J'(X(i+1)_j) = (1/K)(d(X(i+1)_j,1)+d(X(i+1)_j,2) +...+ d(X(i+1)_j,K)) \quad (1)$$

where K is the number of simulated trajectories, for the m-th trajectory,

$$d(X(i+1)_{j1},m) = g(X(i+1)_{j1},Y'(i+2)_{j2},X'(i+2)_{j2}) + g(X'(i+2)_{j2},Y'(i+3)_{j3}X'(i+3)_{j3}) +...+ g(X'(N\text{-}1)_{j(N-i-1)},Y'(N)_{j(N-i)},X'(N)_{j(N-i)})$$

is, for the m-th trajectory, the cumulative cost up to reaching the final state at the last stage of this trajectory, and $g(a,u,b)$ is the immediate cost of a transition from state a to state b when control u is applied. It is assumed that different simulated trajectories are statistically independent. It is possible to determine $J'(X(i+1)_j)$ iteratively starting with $J'(X(i+1)_j)=0$, and by using [6]:

$$J'(X(i+1)_j) = J'(X(i+1)_j) + G_m * (d(X(i+1)_j,m) - J'(X(i+1)_j)) \quad (2)$$

with $G_m=1/m,\ m=1,2,..,\ K$. Once the values for $J'(X(i+1)_j)$ are computed for $j=1,2,..,\ M_{i+1}$, they are ranked, and the control $X(i+1)_j\ (j=1,..,\ M_{i+1})$ with the lowest value for the approximate cost-to-go function is selected for the next stage. The tasks implied in the estimation of the cost-to-go function and the way they interact are shown in Figure 5.

The most part of neuro-dynamic programming methods [23,26-28] start from an initial sequence of control actions that is used to compute an initial value for the cost-to-go function. From this initial situation, an improvement is tried by applying transformations in this initial sequence that are accepted if the obtained cost-to-go function improves the previous one. Some proposals [26-28] determine the cost-to-go function by functional approximation. These procedures usually require a lot of ('off-line') training time where the training set includes a (usually high) number of possible trajectories. The amount of computing time needed by these last procedures makes difficult their application to problems with real-time constraints. Other procedures [23], simulate some of the possible control trajectories and make some estimations about the expected values corresponding to the cost-to-go function. In the procedure here presented, we also use this strategy of simulating control trajectories, but we include the possibility of decreasing the space of alternative trajectories by using a prediction procedure that anticipates the

future values for the inputs to the system (the levels of water consumption). Moreover, an optimization procedure is applied to determine the feasible control points that define the possible trajectories.

3 Application on Water Supply Networks

In this section we illustrate the use of the previously described procedure in a scheduling problem that appears in a water supply system. The cost function that allows the determination of the feasible points for the time interval i, *Cost[i]*, is obtained by summing four terms:

$$Cost(i) = Cost1(i) + Cost2(i) + Cost3(i) + Cost4(i), \tag{3}$$

where the form and meaning of these are as follows:

Cost1: The sum of the water quantities entering each tank should be similar to the total water supply provided by the *ETAP*.

$$Cost1[i] = A \times \left| C - \sum_{j=1}^{n} y[i][j] \right|$$

Cost2: The water volume within each tank should be less than its maximum capacity (thus avoiding overflowing).

$$Cost2[i] = B \times \sum_{j=1}^{n} H(x[i-1][j] + y[i][j] - w[i][j] - V[j])$$

Cost3: The water volume within each tank should be greater than zero (thus it never empties completely).

$$Cost3[i] = D \times \sum_{j=1}^{n} H(-(x[i-1][j] + y[i][j] - w[i][j]))$$

Cost4: The solution search is guided such that within the tanks the water level remains close to preset levels to allow the system to react appropriately in cases of very high or very low demand.

$$Cost4[i] = E \times i^{k} \times \sum_{j=1}^{n} |x[i-1][j] + y[i][j] - w[i][j] - L[j]|$$

Where:

x[i][j]	Water volume in tank j (1...n) in the time interval i (1...N, where N is the horizon equal to 24 hours).
w[i][j]	Water volume expected to be consumed during time interval i from tank j.
y[i][j]	Water volume supplied to tank j during time interval i.
C	Water volume supplied by the *ETAP* during each time interval.
V[j]	Maximum volume within tank j.
L[j]	Required volume within tank j at the end of the horizon.

H(a) It is a function that H(a)=a if a>0 otherwise H(a)=0.

The continuity condition that exists between serial time intervals *(i-1)* and *i* is:

$$x[i][j] = x[i-1][j] + y[i][j] - w[i][j] \quad (4)$$

The minima (the feasible control vectors) of *Cost(i)* for a given interval *i*, are obtained with a genetic algorithm that includes a local search procedure applied after the mutation operator and it is based on that described in [10]. To determine the feasible control values (the vectors *Y(i)* that minimize the cost function *Cost[i]*), a procedure that mixes a genetic algorithm and a local optimization method is used here. The procedure is based in that described in [10] where genetic algorithms and hill-descending methods have been combined in order to take advantage of their characteristics while avoiding their drawbacks. In each generation of the genetic algorithm, the operators of crossover and mutation, and some iterations of the hill-descending procedure, are applied to the individuals of the population as a mechanism to speed up the convergence thanks to its exploitative properties. As the genetic algorithm works on the solutions found by the local optimization search, it can be considered that the hybrid algorithm has 'Lamarckian' characteristics [10].

Each individual of the population corresponds to a control vector, $Y(i)=(y_1(i),...,y_n(i))$, and each of its genes represents one of its components, $y_k(i)$, coded as a real number. The characteristics of the mutation and crossover, and more details about their implementation can be found in [10,19, 20].

After applying a genetic procedure to the function *Cost(1*), we obtain a way of distributing the water between the tanks that, together with the consumption registered during that period, determines the level of water in the tanks for the following interval. Applying the genetic algorithm again to the function *Cost(2*), a water distribution scheme for the second time interval can be determined. By these means, finding a solution for subsequent cost functions obtained, *Cost(3),..., Cost(24*), a feasible control trajectory is built, since within each interval the feasible water distribution between the tanks are defined, whilst physical restrictions and demand levels are met and the water levels required at the end of the horizon period are provided. However, this is only one of several possible trajectories and not necessarily the optimal one.

At the end of each time interval, the value of real demand, *W(i+1)*, along with *Y(i+1)* and *X(i)* determine the water level in the tanks to be considered for the subsequent time interval *i+1*. Thus the positions of the valves are determined such that the flows obtained are those required to enter each tank.

However, the possible imprecision in valve gauging and the interactions that take place between the different flows, due to the network operation regime, etc., mean that there exist flows into each tank that differ from those calculated. Therefore, the real water volumes that are finally introduced into each tank are the volumes taken to determine water tank levels during the following time interval.

In this way, the application of the general procedure described in Section 3 begins by determining the function *Cost(1)*. A set of optimal values (feasible control values) of the cost function *Cost(1)* is obtained ($M_1=R$ optimal values as maximum) and, starting from each of these solutions control trajectories are built to estimate the cost-to-go functions corresponding to each feasible value. To do that, the Prediction box of Figure 5 uses the prediction model received from the Prediction module (see Figure 3) and provides the predicted water consumption for the next stages, *W'(2),..., W'(24*). The aforementioned prediction model is obtained by using an hybrid technique based on RBF networks [11] and classical orthogonal transformations such as SVD and QR [12,13] that allows not only the computation of the parameters of the RBFs, as in the majority of the neural procedures, but also the automatic determination of the number of inputs and RBFs that define the structure of the network. A detailed description of this prediction procedure is out of the scope of the present paper and can be found in [14] and [15].

Thus, by using the predicted values *W'(2),..., W'(24*), it is possible to determine the *R* cost functions *Cost(2)* corresponding to each of the *R* feasible point obtained by optimizing *Cost(1)*, and *r* new feasible points are obtained by applying the genetic algorithm to each of the *R* functions *Cost(2)*. From each of the *R*r* ($M_2=R*r$) feasible points, *R*r* different cost functions *Cost(3)* can be determined, and applying again the genetic algorithm a given number of feasible points is obtained, and so on, until the last stage (*N=24*) is reached. In our case, only one feasible point is obtained from the *R*r* cost functions *Cost(3), Cost(4),..,Cost(24)*. Thus, during the first stage *K=R*r* trajectories can obtained starting from the *R* solutions of *Cost(1)* and the cost-to-go function for each of the *R* feasible control vectors is approximated, as indicated in Section 2, and by using a cost function, *g,* that evaluates the amount of change in the position of the valves: the values of $g(X(i)_{j0},Y(i+1)_{j1},X(i+1)_{j1})$ decrease with the distance between *Y(i)* (that determines *X(i)*) and *Y(i+1)*, because this implies smaller changes in the valves position, thus reducing the electrical consumption and the increasing the mean-life of the valves.

The solution of *Cost(1)* with the minimum value for the approximate cost-to-go function is applied to the system, and this determines the state for the following stage, that appears in the next hour. Starting from this operation point, and from the real state of the tanks after the first period, the function *Cost(2)* is

obtained and thus we obtain new *R*r* trajectories that allow us to select the control strategy for this second interval. The process is repeated in each interval until the end of the horizon is reached. This procedure is briefly described in Figure 6.

The maximum number of trajectories used, determined by *R* and *r*, is limited by the maximum computing time available, which is determined by the time required to apply the control action and to let the water levels gets their corresponding values. As can be well understood, as more trajectories are processed, the performance of the control procedure improves. This approach is similar to that described in [8], where Monte Carlo sampling is also applied to estimate the cost-to-go functions that are to be optimized.

In the application here considered, the system model used to determine, by simulation, the feasible scheduling trajectories in the estimation of the *cost-to-go function* is easy, as corresponds to equation (4). Thus, the Simulation box in the scheme of Figure 5 is very simple. In more complex cases it is possible to use techniques such as neural networks to derive a model of the system from its observed input/output behavior.

Once the main elements of the procedure are described, the next section presents the issues related with its parallel implementation and Section 5 provides experimental results to illustrate the performances of the method and demonstrate its correct operation.

4 Parallel implementation of the procedure

To get an efficient parallel implementation of a given application it is necessary to take into account the type (or types) of parallelism that this application presents and to compare it with the specific characteristics of the platform where the parallel procedure is to be executed. Together with the well-known *data parallelism* and *functional parallelism*, in [21], *object parallelism* and *metaproblem parallelism* are also defined. The *object parallelism* appears in problems solved with the aid of discrete event simulators and is similar to the data parallelism except that the objects are the units of parallelism, being their size larger than the fundamental units of parallelism in a data parallel problem.

With respect to the *metaproblem parallelism*, it can be considered as a special type of functional parallelism in which there are several interrelated units, each one corresponding to a complete problem itself. The number of concurrent components in a metaproblem is relatively small and they can be efficiently implemented in distributed systems as the requirements of communication latency and bandwidth between modules are less demanding.

The procedure here proposed to solve the water supply scheduling problem is outlined in Figures 3 and 5. It presents different modules that interact themselves by exchanging data and implement different types of techniques, i.e., neural networks for functional approximation and time series prediction, genetic algorithms to determine the feasible functioning points, Monte Carlo simulations to generate control trajectories in order to evaluate the cost-to-go function, etc. More specifically, in Figure 3 three modules are shown, the module to determine the parameters of the prediction model (a RBF network), the module to approximate the functional behaviour of the system (also a neural network), an the module that builds and simulates the control trajectories, evaluates their cost-to-go function, and determines the optimal control variables. This last module can be considered as the central element of the neuro-dynamic programming proposed procedure. Thus it is called neuro-dynamic programming module.

In this way, the technique here proposed can be shown as a metaproblem, where the procedures corresponding to each module require to exchange data. For example, the prediction module has to send the parameters of the prediction model used to make the predictions whenever a change on its parameters is determined. Nevertheless, the required communication bandwidth between the procedures of the different modules is not very high and, if necessary, each module can be executed in different machines that do not need to be connected by fast networks. Of course, the procedures corresponding to different modules can be executed in the same multiprocessor or cluster where different processors are allocated to each procedure, although each procedure presents different type of parallelism.

As it has been said, the different modules of the procedure can be implemented as parallel programs to allow the time constraints to be fulfilled (future consumption values should be predicted and feasible solutions should be obtained by the genetic algorithm before a given time limit) and/or the performance of the procedures to be improved (more feasible solutions and a more accurate cost-to-go function can be managed). Although in our present parallel implementation of the procedure we have used a parallel program for the prediction module, in this paper we will only describe the way the neuro-dynamic programming module has been parallelized, as it can be considered the kernel of the scheduling procedure.

Parallel processing can be used with relative efficiency to speed up the neuro-dynamic programming procedures and to improve their performance. Almost all neuro-dynamic programming procedures require to determine, usually by simulation of the system, several control trajectories. Thus, this part of the procedures can be parallelized by allocating a different subset of the trajectories to generate to a different processor. In this case, the processors requires to communicate in order to assure

that each processor generates a set of trajectories different to the ones generated by the other processors, and to detect trajectories having a low probability to be selected as optimal. So, if the number of trajectories generated by a processor can change or the computational cost associated to the analysis of a trajectory depends on the characteristics of the trajectory and cannot be estimated when the distribution of work is done, it is required a dynamic load balancing procedure that also requires communication between the processors. Some examples of parallel procedures for neuro-dynamic programming are presented in [22,23].

Figure 7 provides the considered possibilities for parallelizing the procedure described in Figure 6 corresponding to the Neuro-Dyamic Programming module shown in Figure 5. The lines in Figure 7 that are written in bold characters are those where parallel processing can be applied. Thus, *line (4)* corresponds to the execution of the genetic algorithm that search M_i=R vectors that minimize the previously determined cost function *Cost(i)* and represent feasible functioning points for the system. The parallelization of this *line (4)* can be done by using three main alternatives that are represented in Figure 8 and described in the following:

1) *Multiple runs of independent sequential genetic algorithms for unimodal optimization* (Figure 8.a)*:* By starting *R* executions of the genetic algorithm (considering that it may be very difficult that similar solutions were found by different executions of the algorithm).
2) *Multiple runs of independent parallel genetic algorithms for unimodal optimization* (Figure 8.b)*:* By repeating R times a parallel genetic algorithm that searches one of the feasible points in each execution.
3) *One run of a parallel genetic algorithm for multimodal optimization* (Figure 8.c): By implementing a parallel genetic algorithm that were able to find the *R* feasible points in only one parallel execution. In this option, the parallel genetic procedure should keep, in each processor, different subpopulations that explore different zones of the solution space, thus maintaining the diversity of the whole population in order to assure the convergence to different solutions.

From the point of view of parallel processing, the interest of the alternative (3) is related with the way to define the different search spaces assigned to each subpopulation, and the way these zones are dynamically modified in order to have a balanced workload distribution. Moreover, the speedup that can be obtained in this case is not limited to depend linearly with the number of processor because the

parallelization not only can reduce the number of iteration to get the solutions but also the computational cost of each iteration, as the subpopulations have less individuals than the whole subpopulation. Thus it would be possible to observe superlinear speedup grows with the number of processors.

The search of $M_{i+1}=r$ feasible points in the *line (11)* of Figure 7 can be parallelized in a similar way as the genetic algorithm, by using each of the possible three options previously described. Otherwise, the parallelization of the computation of a feasible point indicated in line (16) can be only done through the alternative number (2), but taking into account that r=1 ($M_{i+2}=1$, $M_{i+3}=1$,..).

As can be seen, the three possible alternatives to parallelize the search for R feasible points can be also mixed according to the value of R and the number of available processors in the parallel computer. For example, in a computer with P processors and $P < R$, it would be possible to repeat R/P times the execution of a parallel genetic algorithm (alternative (2)) that is able to find P different feasible points, one per processor (alternative (3)).

The other opportunity for parallelism in the procedure of Figure 7 corresponds to the processing of each simulated control trajectory in order to evaluate the cost-to-go function. In this way, the iterations indicated in *line (5)*, each corresponding to one of the R trajectories that starts from the corresponding feasible point obtained in *line (4)*, can be processed in parallel allocating each iteration (i.e. the execution of the procedure *Trajectory_generation()*) to a different processor. The same option can be used for the iterations indicated in *line (12)*, corresponding to each of the r trajectories simulated from each of the R initial feasible points. Figure 9 provides a scheme corresponding to the parallel execution of the procedure by a set of $P=6$ processors that process $K=6$ trajectories. The boxes in each layer of the figure correspond to the genetic algorithm used to determine the feasible control points that allow to build the trajectories. In the first layer a parallel genetic algorithm that is able to determine $R=3$ feasible control points is used. Then, for each of these points a parallel genetic algorithm that allows the obtention of the other $r=2$ feasible points is used in the second layer. Finally, the rest of each of the $K=R*r=6$ trajectories are processed by only one processor that apply a sequential parallel algorithm to determine the feasible point corresponding to each of the following layers.

In this way, there are several options to parallelize the procedure, taking into account the different options to parallelize the genetic algorithm that search the required feasible points, and the possibilities for distributing the generation and evaluation of the different trajectories. Moreover, the numbers R and r that determine the amount of trajectories simulated can be set according to the number of available

processors and the real time requirements. As many trajectories were processed, better is the approximation obtained for the cost-to-go function, and the performance of the method is also improved.

There is communication between processors in case of alternatives (2) and (3) for a parallel genetic algoritm, and in the parallel processing of the trajectories, in order to avoid that different processors generate the same trajectory thus doing the same work. In this case, one of the two processors proceeds with the simulation of the trajectory while the other can start the processing of one of the pending trajectories. Once the different parallelization alternatives have been outlined, the following section provides some representative experimental results of the procedure.

5 Experimental results

To predict the water demand from each tank, an RBF network learning procedure based on the LMS rule along with the QR-cp and SVD procedures was used, following [14,15]. The network was trained using a window of size equal to $D=8$; thus samples from the tank demand observed during the previous eight hours were used to predict the next hour's demand. The relative error, taken in absolute value, is at the 5% level and the NRMSE [14] for these 24 samples is approximately equal to 0.40. In Figure 10 the predicted demand values for one of the three tanks during a complete day are compared to their real counterparts; this gives some indication about the accuracy of the method.

The values of the parameters required by the operators of the hybrid genetic algorithm that determines the feasible control values have been set as follows: y_{max}=3000.0, y_{min}=0.0, s=5.0, T=100, $p1$=0.2, $p2$=0.2, and P=0.05.

Figure 11 shows the curves corresponding to the evolution of water levels in a given tank when the controller proposed here is used (controller) and when different manual controls (Man1, Man2, Man3) are used. It can be seen the solution obtained by the controller is feasible and presents a slower degree of variation, with no extreme maximum and minimum levels, than the curves corresponding to the manual control of the system.

The parallel procedure corresponding to the Neuro-Dynamic Programming module was implemented in a cluster of PCs (Pentium II, 333 MHz) by using. Figures 12.a and 12.b compare the changes in the volumes of the considered (three) tanks, $X(t)$, and the amount of water that enters each tank during one hour, $Y(t)$, when using only one of the processors (Figure 12.a) and when using eight processors (Figure 12.b) to search the feasible solutions in a given amount of time (each processor

processes a trajectory). The alternatives used to implement the parallel genetic algorithms requires in the procedure have been the alternatives (1) and (2).

The solution determined when using several processors presents a smaller change in the volumes of water entering the tanks. To give an idea of the computing times and the speedups obtained by the parallel implementation of the procedure, Figure 13 shows the results for different number of trajectories and processors. These results correspond to the mean values obtained from five executions of each case. As can be seen, taking into account that the control values are required each hour, the times obtained are low enough if four or more processors are used. If the procedure is executed by only one processor, the times obtained are satisfactory up to eight simulated trajectories. The evolution of the tanks satisfy the imposed constraints and, as is shown in Figure 12, the behaviour of the system is improved as more trajectories are processed.

6 Conclusions

The benefits provided by parallelism in complex planning, scheduling, and control applications has been demonstrated through an application of tank scheduling in a water supply system. A procedure based on Monte Carlo simulation, RBF networks and parallel genetic algorithms has been presented to solve the considered application. The procedure can be used when the sets of possible states, inputs and control variables are continuous, and applies the genetic algorithm to determine the feasible operation points at each stage, by using a predicted value of the inputs. The cost-to-go functions for each feasible state are approximated by Monte Carlo simulation, thus taking into account the influence of the future cost in the selection of the control for the next stage. The values considered in each stage are the real values of the system after the last stage.

The procedure has been implemented in parallel in a cluster of computers. As shown in the experimental results, parallel processing is beneficial because it allows us to calculate a greater number of trajectories at any given time, thus improving the effectiveness of the control procedure. Moreover, it reduces the time required to process a given number of trajectories.

The proposed procedure has a high amount of parallelism that can be used to reduce the time required to reach a control decision. The experimental results presented in Section 5 confirm this circumstance. It is possible to carry out multiple generation and simulation of trajectories in parallel and occasionally to communicate some results. Moreover, the genetic algorithm that determines the feasible

operation points can also be parallelized efficiently. In the present implementation of our genetic algorithm, several runs of the algorithm have to be done in order to determine sufficient feasible operation points. Although the number of minima in the cost function, *Cost()*, is usually high, and it is difficult to obtain repeated values in different runs, the genetic algorithm can be improved by including the ability for multimodal optimization with niching methods, such as crowding and fitness sharing [24,25]. These methods maintain population diversity and allow the genetic algorithm to converge to a set of the possible minima at the same time. Thus, the parallel implementation of a genetic algorithm including niching methods will improve our procedure and this will be the immediate focus of our future research.

Acknowledgements

This work has been financed as part of project CICYT TIC97-1149; we are also grateful to the water supply company of the city of Granada (Spain), EMASAGRA Corp.

Bibliography

[1] Anderson, T.E.; Culler, D.E.; Patterson, D.A. and the NOW team:"A Case for NOW (Networks of Workstations)". IEEE Micro, pp.54-64, February, 1995.

[2] Keane, J.A.:"Parallel Systems in Financial Information Processing". Concurrency: Practice and Experience, 8 (10), pp.757-768. December, 1996.

[3] Chien, A.A.; Hill, M.D.; Mukherjee, S.S.:"Design challenges for high-performance network interfaces". IEEE Computer, pp.42-44. November, 1998.

[4] Jowitt P.W., Germanopoulos G.: "Optimal Pump Scheduling in Water Supply Networks". *ASCE Journal of Water Resources Planning and Management*. Volume 118, No. 4, pp 406-422, July 1992.

[5] Bertsekas, D.P.: "Dynamic Programming and Optimal Control". *Athena Scientific*, Beltmont, Massachusetts, 1995.

[6] Bellmann, R.; Dreyfus, S.:"Applied Dynamic Programming". *Princeton University Press*, Princeton, N.J., 1962.

[7] Bertsekas, D.P.; Tsitsiklis, J.N.: "Neuro - Dynamic Programming". *Athena Scientific*, Belmont, Massachusetts, 1996.

[8] Tesauro, G.; Galperin, G.R.: "On-line policy improvement using Monte-Carlo search". In *Advances in Neural Information Processing*, 9, pp.1068-1074, MIT Press, 1996.

[9] Watkins, C.J.C.H.: "Learning from Delayed Rewards". PhD Thesis, Cambridge University, 1989.

[10] Renders, J.M.; Flasse, S.P.: "Hybrid Methods using Genetic Algorithms for Global Optimization". *IEEE Trans. on Systems, Man, and Cybernetics (Part B)*, Vol.26, No.2, pp.243-258. Abril, 1996.

[11] Moody, J., Darken, C.: "Fast Learning in Networks of Locally-Tuned Processing Units". *Neural Computation* 1 (2), pp. 281-294, 1989.

[12] Golub, G.H., Van Loan, C.F.: "Matrix Computations". *Johns Hopkins University Press*, Baltimore, MD, 1989.

[13] Kanjilal, P.P., Banerjee, D.N..: "On the Application of Orthogonal Transformation for the Design and Analysis of Feedforward Networks". *IEEE Transactions on Neural Networks*, 6 (5), pp. 1061-1070, 1995.

[14] Salmerón, M.; Ortega, J.; Puntonet, C.G.: "On-line optimization of Radial Basis Function Networks with Orthogonal Techniques". In *Foundations and Tools for Neural Modeling* (Mira, J.; Sánchez-Andrés, J.V. Ed.), Lecture Notes in Computer Science, 1606, pp.467-477, Springer-Verlag, Berlin, 1999.

[15] Salmerón, M.; Ortega, J.; Puntonet, C.G.; Prieto, A.:"Improved RAN sequential prediction using orthogonal techniques". Neurocomputing (in press).

[16] Booker, L.; Goldberg, D.; Holland, J.: "Classifier systems and genetic algorithms". *Artificial Intelligence*, 40 (1-3), pp.235-282, 1989.

[17] Grefenstette, J.: "Strategy acquisition with genetic algorithms". In L. Davis (Ed*.), Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[18] Whitley, D.; Dominic, S.; Das, R.; Anderson, C.: "Genetic Reinforcement Learning for Neurocontrol Problems". *Machine Learning*, 13, pp.259-284, 1993.

[19] Goldberg, D.E.:"Genetic Algorithms in search, Optimization, and Machine Learning". Addison-Wesley, NY, 1989.

[20] Janikov, C.Z.; Michalewicz, Z.: "An experimental comparison of binary and floating pointrepresentation in genetic algorithms". In *Proc. 4th Int. Conf. Genetic Algorithms*, Morgan Kaufman, San Francisco, pp.31-36, 1991.

[21] Fox, G.C.:”An application perspective on high-performance computing and communications”. Technical Report SCCS-757, Syracuse University, NPAC. April, 1996.

[22] Bertsekas, D.P.; Tsitsiklis, J.N.: “Parallel and distributed computation: Numerical methods”. Prentice Hall, 1989.

[23] Tesauro, G.J.; Galperin, G.R.:"On-line policy improvement using Monte-Carlo search". Advances in Neural Information Processing Systems, 9, MIT Press, pp.1069-1074, 1997.

[24] Sareni, B.; Krähenbühl, L.: "Fitness sharing and niching methods revisited". *IEEE Trans. on Evolutionary Computation*, Vol.2, No.3, pp.97-106. September, 1998.

[25] Goldberg, D.E.; Richardson, J.: "Genetic algorithms with sharing for multimodal function optimization". In *Proc. 2nd Int. Conf. Genetic Algorithms*, (J.J. Grefenstette, Ed.), pp. 41-49, 1987.

[26] Tsitsiklis, J.N.; Van Roy, B.:"An analysis of temporal-difference learning with function approximation". IEEE Trans. on Automatic Control, Vol.42, No.5, pp.674-690. May, 1997.

[27] Crites, R.H.; Barto, A.G.:"Improving elevator performance using reinforcement learning". Advances in Neural Information Processing, 8, MIT Press, pp.1017-1023, 1996.

[28] Zhang, W.; Dietterich, T.G.:"High-performance job-shop scheduling with a time-delay TD(lambda) network". Advances in Neural Information Processing, 8, MIT Press, pp.1024-1030, 1996.

Figures

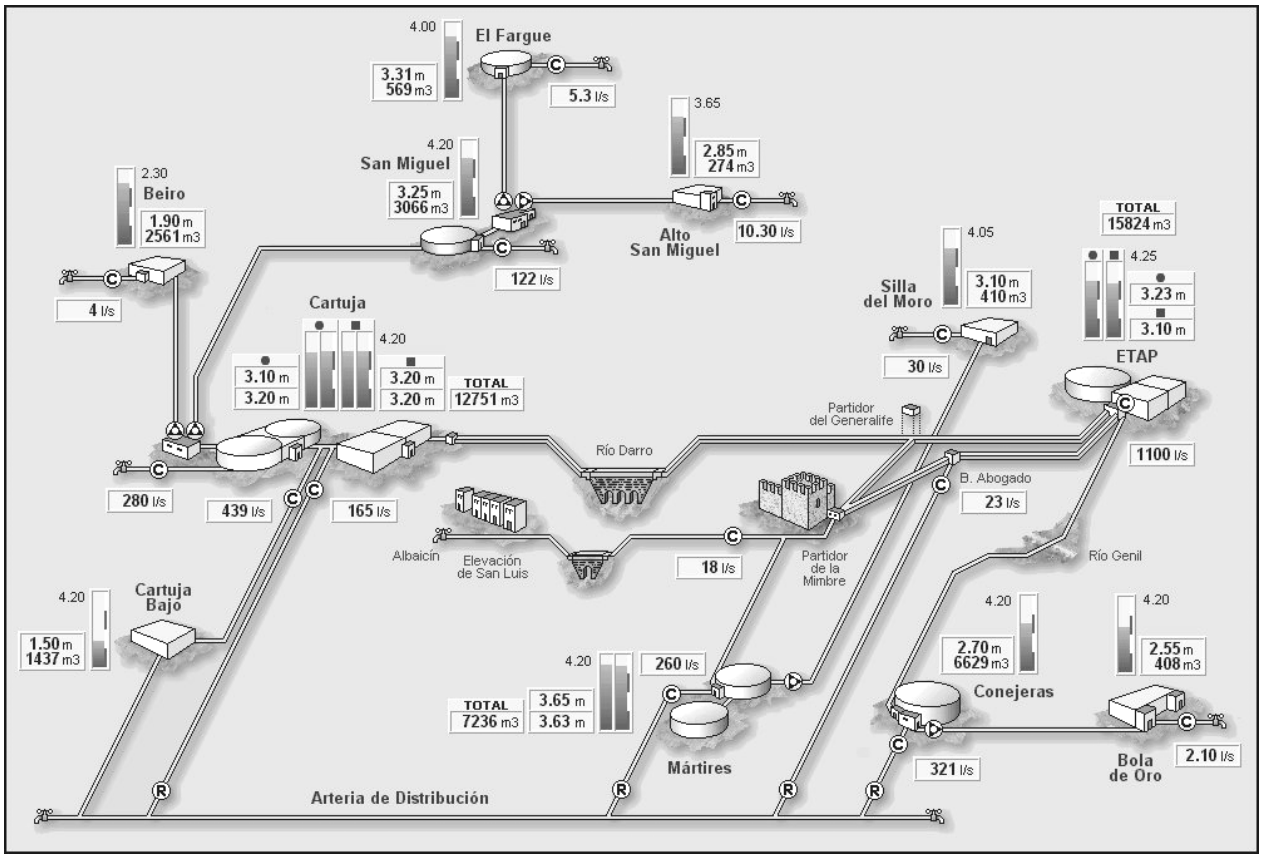

Fig.1. Scheme of the Water Distribution System of Granada (Spain)

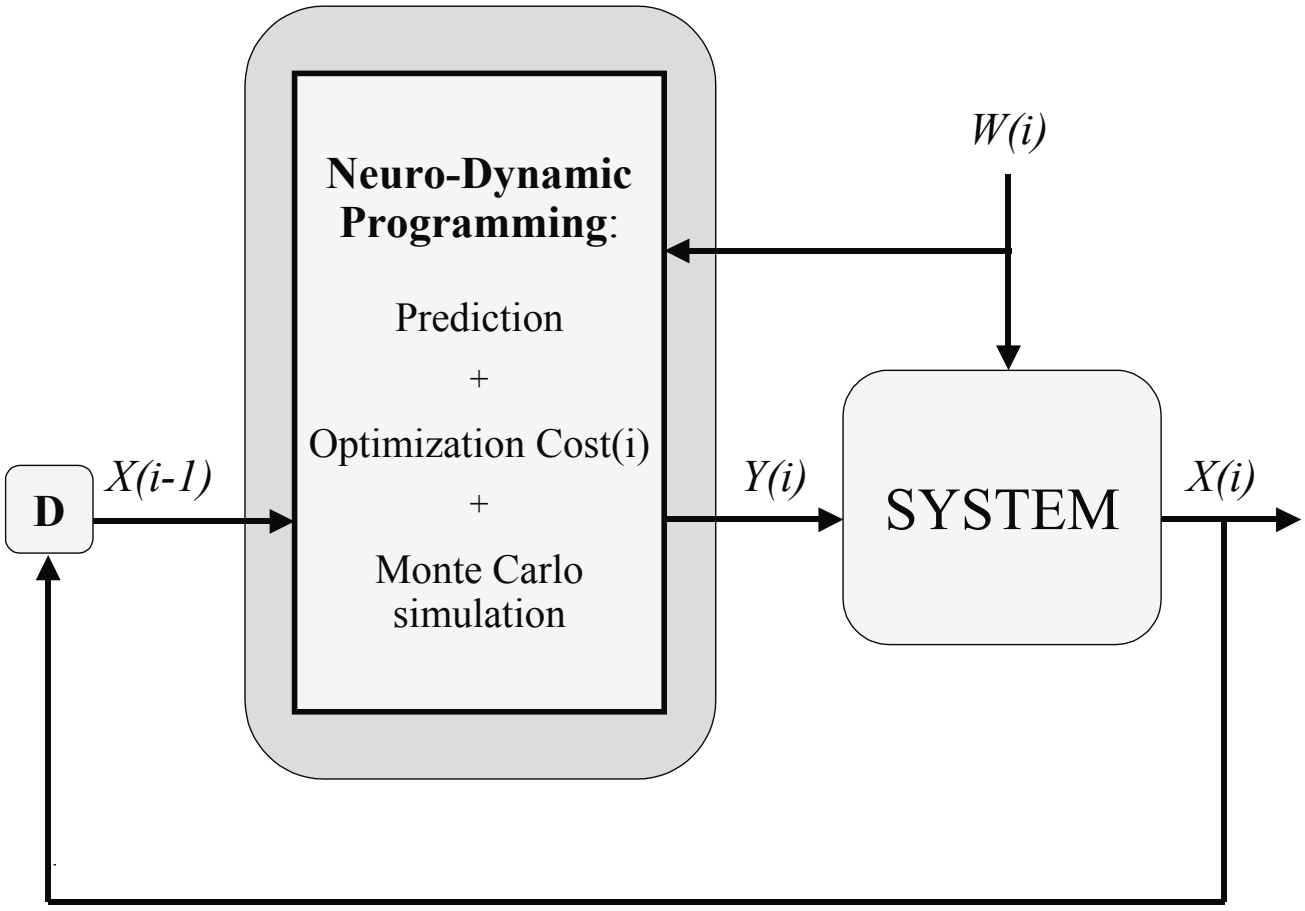

Fig.2. Outline of the system controller

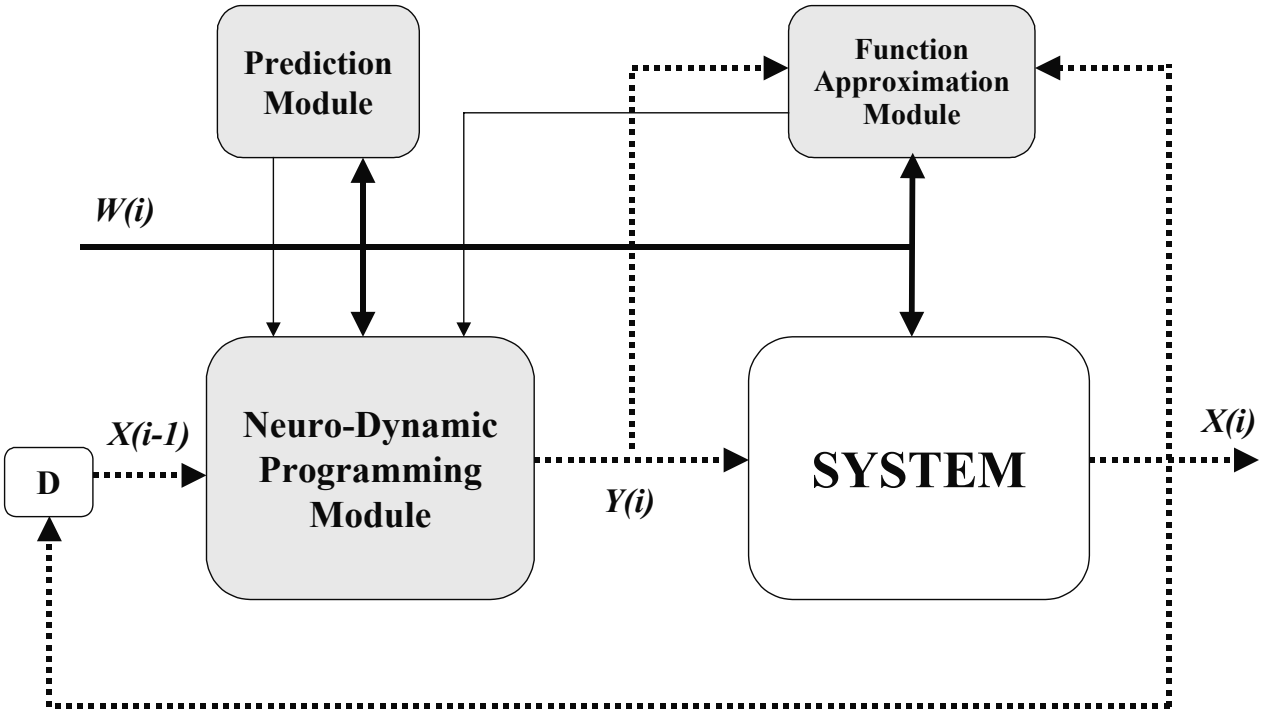

Fig.3. Modules of the proposed hybrid procedure

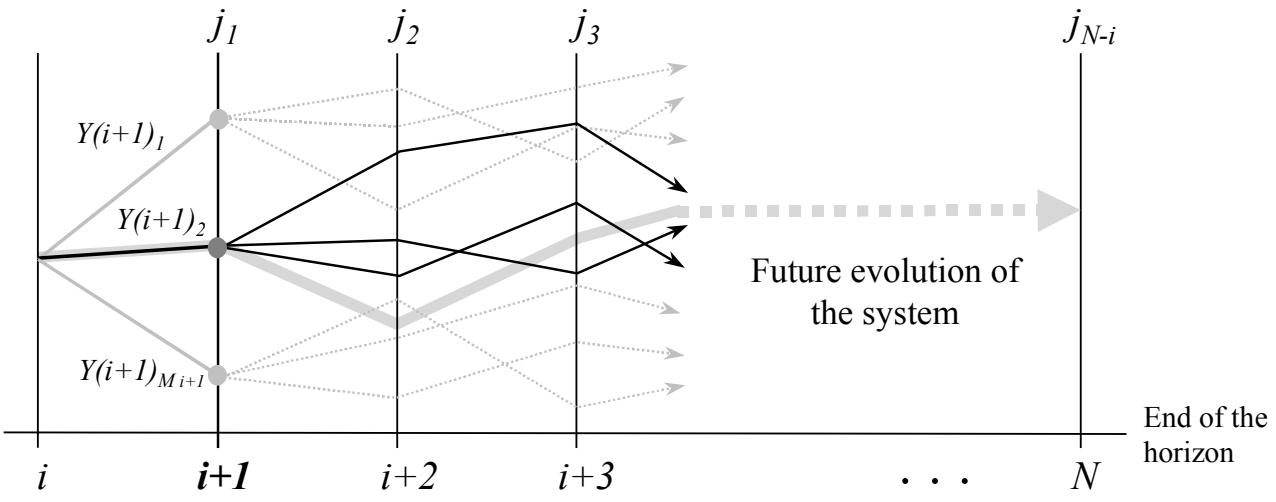

Fig.4. Generation of sample trajectories

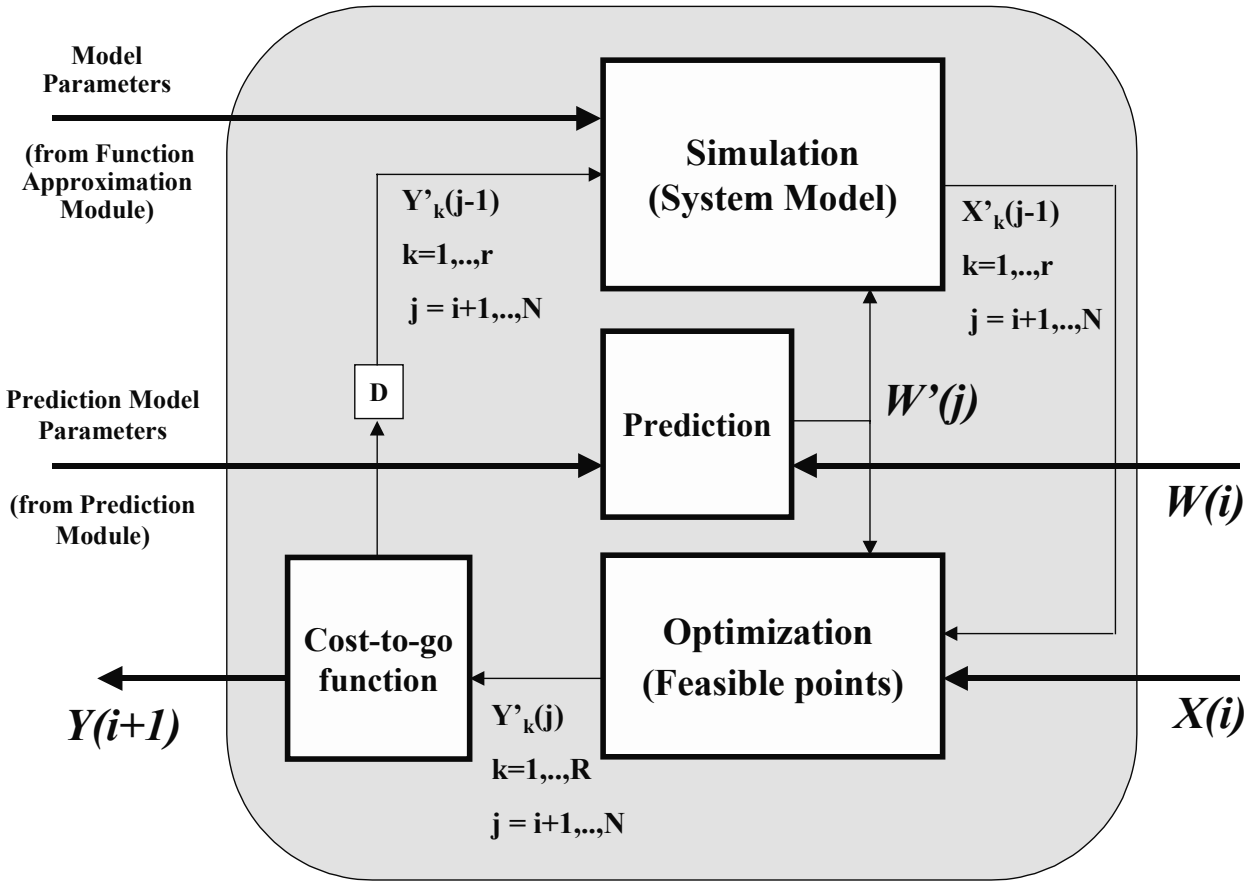

Fig.5. Outline of procedure implemented by the Neuro-Dynamic Programming Module

```
(1)  For each i (i = 1,2,..,N)  {
(2)       Obtain W'[i][j] (j=1,2,.., n);          /* Using the Model (in box Prediction)          */
                                                  /* computed by the Prediction  Module           */
(3)       Determine Cost[i];                      /* Using W'[i][j] and X[i-1][j]  according to (7) */
(4)       Compute R minima of Cost[i];            /* R feasible points obtained by a Genetic Algorithm */
(5)       For each k (k=1,..,R){                  /* For each feasible point a set of trajectories are evaluated */
(6)         Trajectory_generation(i,k,r);         /* Build r control trajectories for feasible point k and evaluate */
                                                  /* the cost-to-go function                      */
          }
(7)       Select S such that J'(S) = min {J'(1),...,J'(R)}
          /* The vector Y corresponding to the point S is the solution for stage i, Y[i][j] (j=1,..,n)  */
     }

(8)  Trajectory_generation(i,k,r) {
(9)               Obtain W'[i+1][j] (j=1,2,...,n)
(10)              Generate Cost[i+1];          /* Using W'[q][j] and X[q-1][j] corresponding to point k */
(11)              Determine r minima of Cost[i+1];
(12)              For each p (p=1,..,r) {
(13)                    For each s (s=i+2,..,N) {
(14)                          Obtain W'[s][j] (j=1,2,..,n)
(15)                          Generate Cost[s];
(16)                          Determine a minima of Cost[s];
                        }
                  }
(17)              Evaluate J'(k);              /* Using the r trajectories simulated and expressions (5) and (6) */
     }
```

Figure 6. Procedure implemented by the Neuro-Dynamic Programming Module

```
(1)  For each i (i = 1,2,..,N)  {
(2)       Obtain W'[i][j] (j=1,2,.., n);          /* Using the Model (in box Prediction)                    */
                                                  /* computed by the Prediction  Module                     */
(3)       Determine Cost[i];                      /* Using W'[i][j] and X[i-1][j]  according to (7)         */
(4)       Compute R minima of Cost[i];            /* R feasible points obtained by a Genetic Algorithm      */
(5)       Par For each k (k=1,..,R){              /* Evaluation of each of the R trajectories  done in parallel */
(6)         Trajectory_generation(i,k,r);         /* Build r control trajectories for feasible point k and evaluate */
                                                  /* the cost-to-go function                                */
          }
(7)       Select S such that J'(S) = min {J'(1),...,J'(R)}
          /* The vector Y corresponding to the point S is the solution for stage i, Y[i][j] (j=1,..,n)      */
     }

(8)  Trajectory_generation(i,k,r) {
(9)            Obtain W'[i+1][j] (j=1,2,...,n)
(10)           Generate Cost[i+1];              /* Using W'[i+1][j] and X[i][j] corresponding to point k  */
(11)           Determine r minima of Cost[i+1];
(12)           Par For each p (p=1,..,r) {
(13)                 For each s (s=i+2,..,N) {
(14)                       Obtain W'[s][j] (j=1,2,..,n)
(15)                       Generate Cost[s];
(16)                       Compute a minima of Cost[s];
                     }
               }
(17)           Evaluate J'(k);                  /* Using the r trajectories simulated and expressions (5) and (6) */
     }
```

Figure 7. Parallelization of the procedure implemented by the Neuro-Dynamic Programming Module

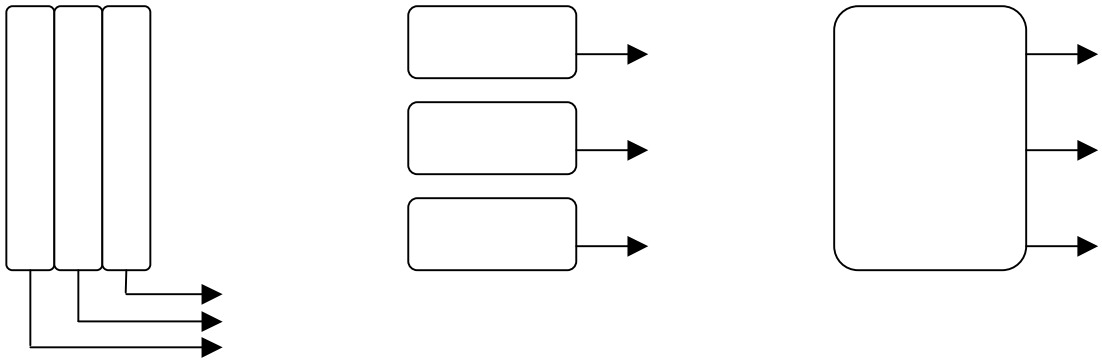

Fig.8. Alternative for parallelizinf the genetic algorithm: (a) alternative 1; (b) alternative 2; and (c) alternative 3

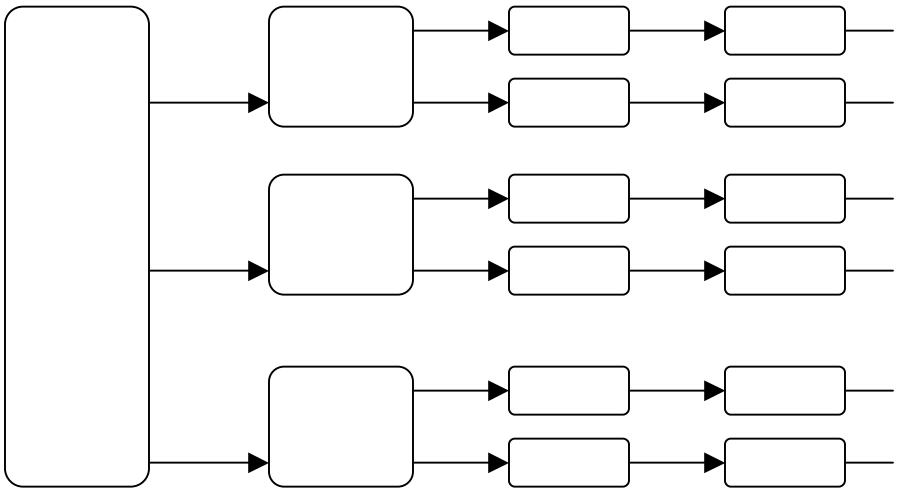

Figure 9. Parallel processing of the procedure with 6 trajectories and 6 processors

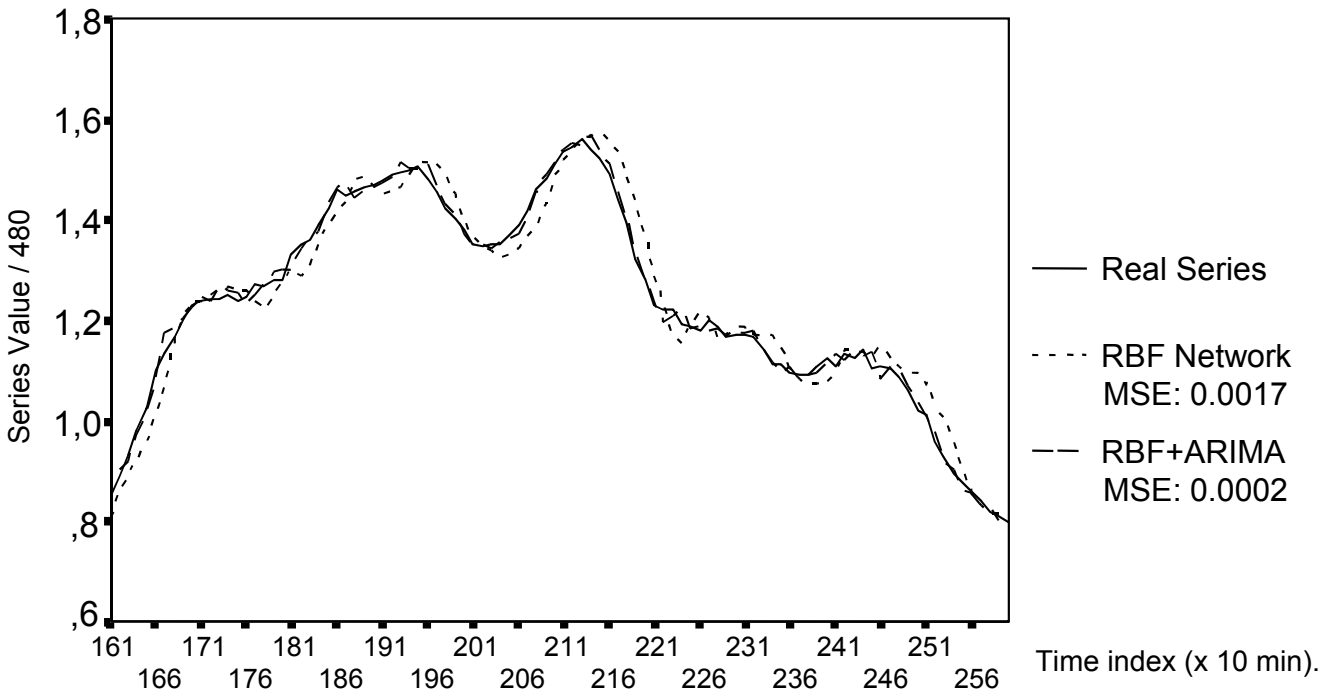

Fig.10. Comparison of the real and the used predicted demands

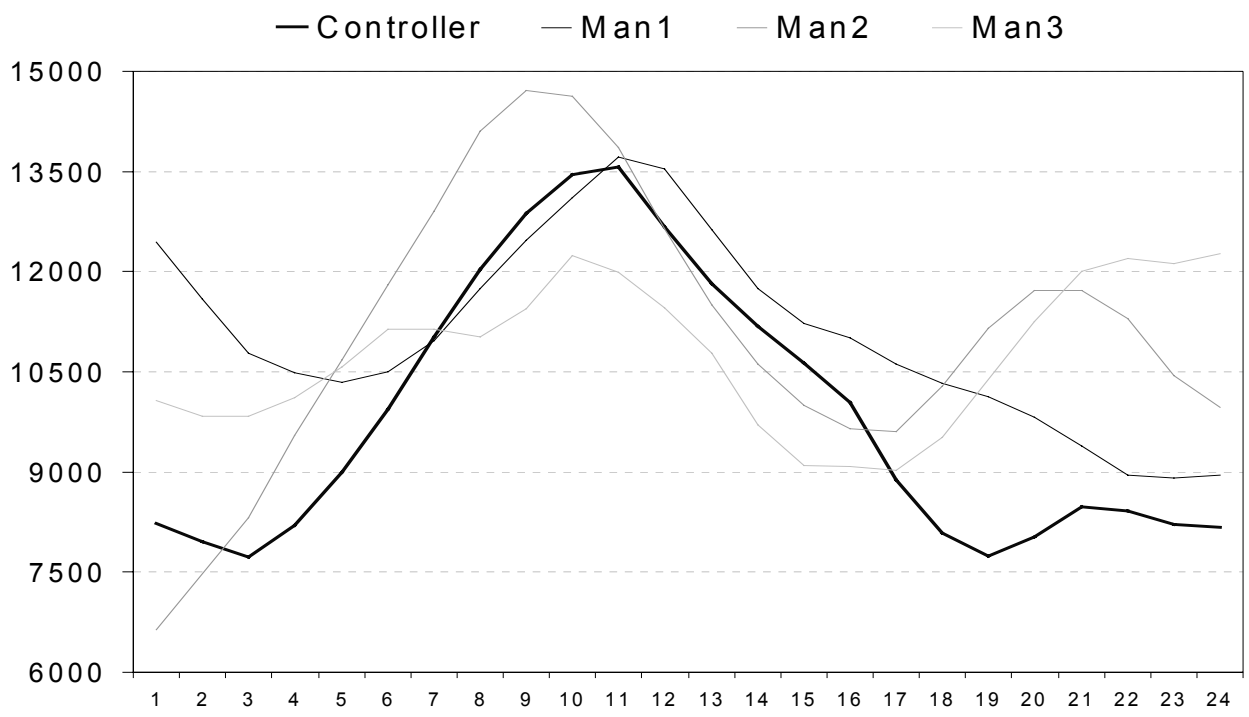

Fig.11. Comparison between different manual control (Man1,Man2,Man3) and the controller obtained with the procedure described

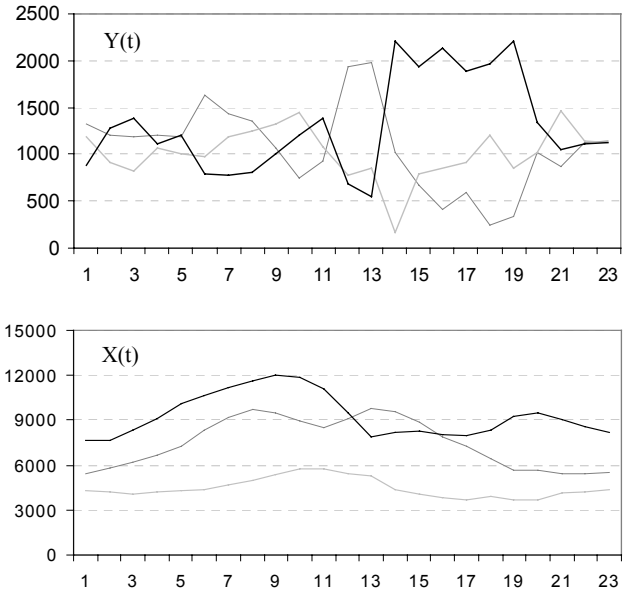

Fig.12.a. Results for 1 trajectory

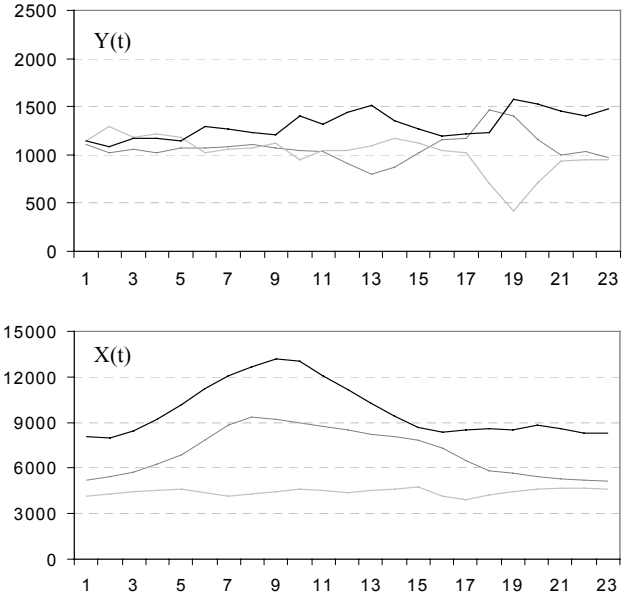

Fig.12.b. Results for 8 trajectories

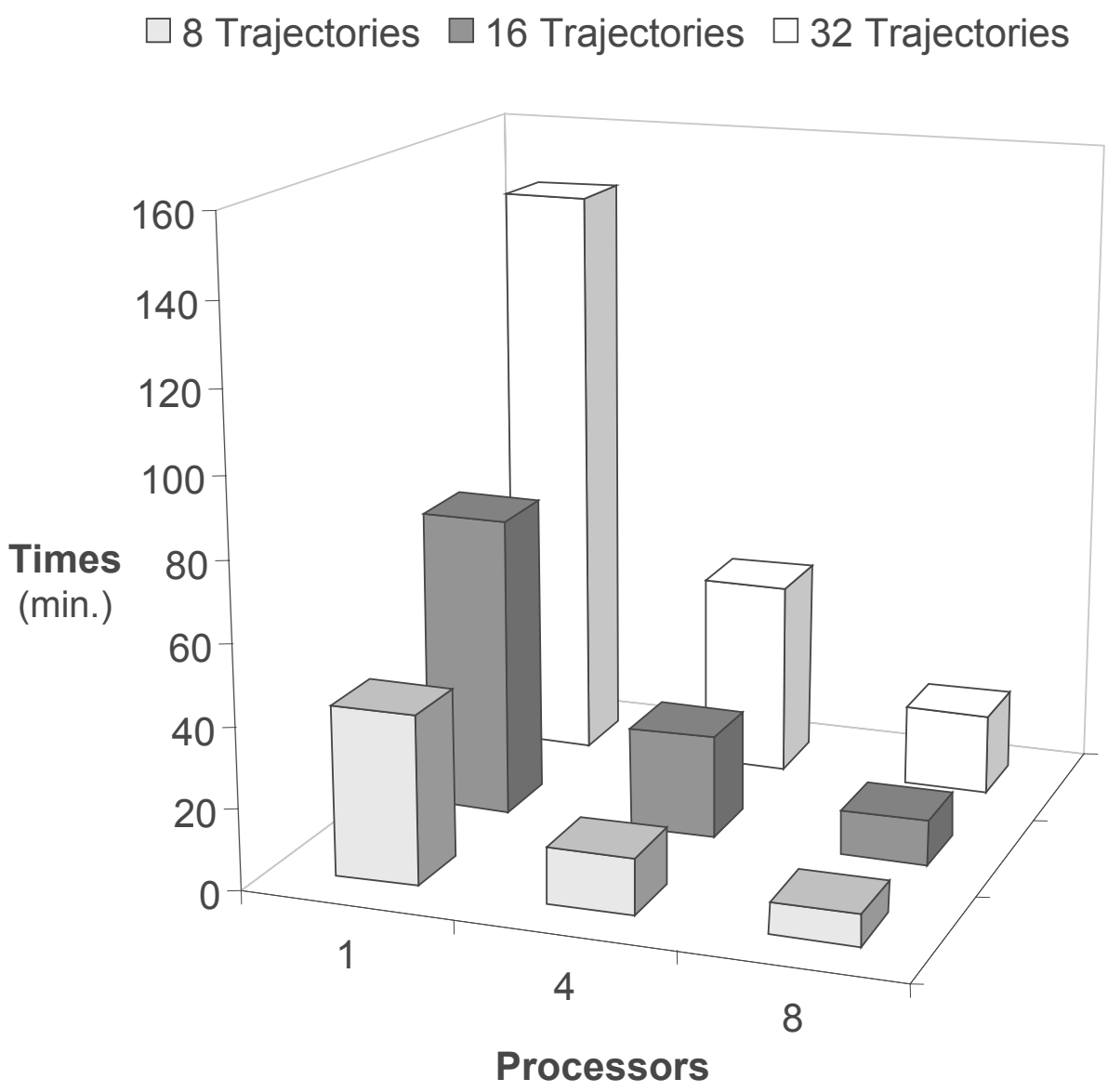

Fig.13. Speedups for different number of generated trajectories and processors