Parallel Dynamic Water Supply 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 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 combines dynamic programming with genetic algorithms and time series prediction in order to solve problems in which decisions are made in stages, and the states and control belong to a continuous space. Taking into account 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 widely extended platform that can make parallelism a practical means of tackling 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

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.), in the configuration of parallel architectures, become even clearer as the rate of performance improvement for these hardware components speeds up. Moreover, as the larger manufacturing volumes of 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 growing interest in the field of parallel processing because they confirm it as a real possibility to improve the performance of applications not only in the fields of science and engineering but also in wider industrial and commercial environments, where inexpensive platforms, as well as parallel applications, software and tools, are required [2].

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

In this paper we describe how a cluster of computers has been applied to improve 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], such as 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 most closely related to minimizing the exploitation cost (cost of electrical power to move the valves, the increase in the mean life span of the system, etc.), while meeting physical constraints and guaranteeing the required water supply. 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 below 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 where the scheduling problem considered here can be included. Section 3 provides the details of the application of this procedure to our specific problem. Section 4 describes the issues related to 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 A 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 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, and thus influences the cost of future periods of time (*cost-to-go*). Therefore, the effect of a low cost solution at the present time must be balanced against the undesirability of high costs in the future. This situation is 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 actions 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 as $Y(i)$; 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 considered here, 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 in which the states change according to transition probabilities that depend on the control signal applied (decision made). In some applications, however, the computational requirements arising from the use of dynamic programming are overwhelming because the number of states and possible controls is very large. Moreover, knowledge of the transition probabilities between states for each control applied is also required in order to be able 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 recent years, the term *neuro-dynamic programming* (NDP) [7] has been used 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 be managed, 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 noted as 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: Figure 3 illustrates the modules for this procedure. 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 to provide an approximate description of the system and enable easier simulation of the system. The models provided by the Prediction module and by the Function Approximation module are used by our Neuro-Dynamic Programming module to determine a set of feasible functioning points, and to 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, we use a vector $W'(i+1)$, which estimates the vector $W(i+1)$ of inputs in the next stage, and is obtained by using the model determined by the Prediction module. The output $X(i)$ is 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 the vector $Y(i+1)$ also allow the determination of the next stage, $X(i+1)$, 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 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, using 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 means of 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,

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

is, for the m-th trajectory, the cumulative cost incurred on 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 $X(i+1)_j$ $(j=1,..,M_{i+1})$ 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.

Most 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 cost-to-go function thus obtained is an improvement on 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 and the training set normally includes a high number of possible trajectories. The amount of computing time needed by these latter procedures makes it difficult for them to be applied to problems with real-time constraints. Other procedures [23] simulate possible control trajectories and estimate the values corresponding to the cost-to-go function. The procedure presented here also uses this strategy of simulating control trajectories, but includes 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 feasible control points to define the possible trajectories.

3 Application to 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 [29,30]. The cost function that allows the determination of the feasible points for the time interval i, *Cost[i]*, is obtained by summing four terms:

$$Cost(i) = Cost1(i) + Cost2(i) + Cost3(i) + Cost4(i) \quad (3)$$

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 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:

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

To determine the feasible control values (the vectors *Y(i)* that minimize the cost function *Cost[i]*), a procedure that combines a genetic algorithm and a local optimization method is used here. The procedure is based on that described in [10] where genetic algorithms and hill-descending methods have been combined in order to take advantage of their positive characteristics while avoiding their drawbacks. In each generation of the genetic algorithm, the crossover and mutation operators, 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 (to speed up convergence, as a result of their exploitative properties). As the genetic algorithm works on the solutions found by the local optimization search, the hybrid algorithm is considered to have 'Lamarckian' characteristics [10].

Genetic algorithms can be considered as meta-heuristics or general purpose optimization algorithms that require very little knowledge concerning the problem at hand. In [31] the relation was explored between the effectiveness of a general purpose algorithm and the optimization problem to which it is applied, and some NFL (No Free Lunch) theorems are proved indicating that for any algorithm, an improved performance over one class of problems is offset by performance deterioration with respect to another class. Nevertheless, it is possible to improve the behaviour of a general purpose procedure in a problem where limitations are presented by modifications of the algorithm that include some problem-specific operators to accelerate the search with respect to a pure genetic algorithm. In the case of the water supply application presented here, the genetic algorithms perform relatively well. Nevertheless, as said above, we have used a real genetic algorithm based on that described in [10], which includes iterations of a gradient-descendent optimization algorithm to accelerate the convergence.

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 mutation and crossover characteristics, and further 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. By applying the genetic algorithm to the *Cost(2)* function, a water distribution scheme for the second time interval can be determined. By these means, finding a solution for subsequent cost functions, *Cost(3),..., Cost(24*), a feasible control trajectory is achieved, since within each interval a feasible water distribution pattern among the tanks is

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)*, 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 valves are positioned 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 ones taken to determine water tank levels during the following time interval.

The application of the general procedure described in Section 2 begins by determining the *Cost(1)* function. A set of optimal values (feasible control values) of the *Cost(1)* function is obtained ($M_1=R$ optimal values as maximum) and from each of these solutions control trajectories are constructed to estimate the cost-to-go functions corresponding to each feasible value. To do this, the Prediction box in Figure 5 uses the model received from the Prediction module (Figure 3) and provides the predicted water consumption for the next stages*, W'(2),..., W'(24*). This prediction model is obtained by using a 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 RBF parameters, as do most 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 beyond the scope of the present paper, but 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 points 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 by again applying 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 be obtained 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, 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 valve positions, thus reducing electrical consumption and 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. Starting from this operation point, and from the real state of the tanks after the first period, the *Cost(2)* function is obtained and thus new *R*r* trajectories that allow us to select a control strategy for the 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 depends on the time required to apply the control action and for water levels to attain their corresponding values. Evidently, 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 considered here, 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 illustrated in 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 behaviour.

After this description of the main elements of the procedure, the next section presents the issues related to its parallel implementation and Section 5 provides experimental results to illustrate the performance of the method and demonstrate its correct operation.

4 Parallel implementation of the procedure

For 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* [21], *object parallelism* and *metaproblem parallelism* are also defined. *Object parallelism* appears in problems solved with the aid of discrete event simulators and is similar to data parallelism except that the objects are the units of parallelism, being larger than the fundamental units of parallelism in a data parallelism problem.

With respect to *metaproblem parallelism*, this can be considered a special type of functional parallelism in which there are several interrelated units, each corresponding to a complete problem in

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 we propose to solve the water supply scheduling problem is outlined in Figures 3 and 5. This presents different modules that interact by exchanging data and implements different types of techniques, i.e., neural networks for functional approximation and time series prediction, genetic algorithms to determine the feasible functioning points, simulations to generate control trajectories in order to evaluate the cost-to-go function, etc. More specifically, Figure 3 shows three modules, one to determine the parameters of the prediction model (a RBF network), one to approximate the functional behaviour of the system (also a neural network), and one that constructs and simulates the control trajectories, evaluates their cost-to-go function, and determines the optimal control variables. This latter module can be considered the central element of the proposed neuro-dynamic programming procedure and is thus termed the neuro-dynamic programming module.

Therefore, the proposed technique can be depicted as a metaproblem, where the procedures corresponding to each module are required to exchange data. For example, the prediction module has to send the parameters of the model used to make predictions whenever a change in its parameters occurs. Nevertheless, the communication bandwidth required between the procedures of the different modules is not very high and, if necessary, each module can be executed in different machines, which 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 a different type of parallelism.

As noted above, 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 obtained by the genetic algorithm within 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). Although the present parallel implementation of the procedure uses a parallel program for the prediction module, in this paper we only describe the way the neuro-dynamic programming module has been parallelized, as this can be considered the kernel of the scheduling procedure.

Parallel processing can be used with relative efficiency to speed up neuro-dynamic programming procedures and to improve their performance. Almost all neuro-dynamic programming procedures must determine several control trajectories, usually by simulation of the system. Thus, this part of the

procedure can be parallelized by allocating a different subset of the trajectories to be generated to a different processor. In the present case, the processors must intercommunicate in order to ensure that each one generates a set of trajectories that are different from those generated by the other processors, and to detect trajectories having a low probability of being selected as optimal. Thus, if the number of trajectories generated by a processor is liable to change or if the computational cost associated with trajectory analysis depends on the characteristics of the trajectory and cannot be estimated when the distribution of work is done, then we require a dynamic load balancing procedure: this also demands communication between the processors. Some examples of parallel procedures for neuro-dynamic programming are presented in [22,23].

Figure 7 illustrates the possibilities considered to parallelize the procedure described in Figure 6 corresponding to the Neuro-Dynamic Programming module shown in Figure 5. The lines in Figure 7 in bold characters are those where parallel processing can be applied. Thus, *line (4)* corresponds to the execution of the genetic algorithm that searches 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 achieved by using one of the following main alternatives (represented in Figure 8):

1) *Multiple independent runs sequential genetic algorithms for unimodal optimization* (USGA in Figure 8.a): By starting R executions of the genetic algorithm (assuming that similar solutions are very unlikely to be found by different executions of the algorithm).
2) *Multiple independent runs parallel genetic algorithms for unimodal optimization* (UPGA in Figure 8.b): By R repetitions of a parallel genetic algorithm that searches for one of the feasible points in each execution.
3) *One run of a parallel genetic algorithm for multimodal optimization* (MPGA in Figure 8.c): By implementing a parallel genetic algorithm that is capable of finding the R feasible points in just one parallel execution [24,25]. 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 and ensuring convergence to different solutions.

From the point of view of parallel processing, the main interest of alternative (3) is related to the definition of the different search spaces assigned to each subpopulation, and the way these zones are dynamically modified in order to achieve a balanced workload distribution. Moreover, the speedup that can be obtained in this case is not limited to a linear dependence on the number of processors because

parallelization not only reduces the number of iterations required to obtain a solution but also the computational cost of each iteration, as the subpopulations have fewer individuals than the whole subpopulation. Thus superlinear speedups can be observed [37].

The search for $M_{i+1}=r$ feasible points in *line (11)* of Figure 7 can be parallelized in a similar way to that of the genetic algorithm, by using any of the three options previously described. Otherwise, parallelization of the computation of a feasible point as indicated in line (16) can only be done by means of alternative (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 also be combined, depending on the value of R and the number of processors available 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 capable of finding P different feasible points, one per processor (alternative (3)).

The other opportunity for parallelism in the procedure shown in Figure 7 corresponds to the processing of each simulated control trajectory in order to evaluate the cost-to-go function. Thus, 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 us to construct the trajectories. In the first layer, a parallel genetic algorithm that is capable of determining $R=3$ 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 the $K=R*r=6$ trajectories are processed by just one processor that applies a sequential parallel algorithm to determine the feasible point corresponding to each of the following layers.

It is apparent that there are several options to parallelize the procedure, taking into account the alternative ways of parallelizing the genetic algorithm that searches for the required feasible points, and the ways of 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 real time requirements. The more trajectories are processed, the better is the approximation obtained for the cost-to-go function, and the better the performance of the method.

Communication is established between processors in the case of alternatives (2) and (3) for a parallel genetic algorithm, and in the parallel processing of the trajectories, in order to prevent different processors from generating the same trajectory and doing the same work. Thus, one of the two processors proceeds with the simulation of the trajectory while the other can start processing one of the pending trajectories. After this outline of the various parallelization procedures, the following section provides some representative experimental results.

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 size of $D=8$; 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 as the absolute value, is at the 5% level and the NRMSE [14] 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. As can be seen, the prediction procedure has a high degree of accuracy. Moreover, as the prediction module determines the parameters of the prediction model concurrently and asynchronously with the processing of the trajectories by the neuro-dynamic module, an on-line improvement of the prediction accuracy is possible.

The parameters for the operators of the hybrid genetic algorithm [10] that determines feasible control values were set to b=5.0, $p1$=0.2, $p2$=0.2, and T=100 (with the notation used in [10]). The components of the control vector are bounded between y_{max}=3000.0 and y_{min}=0.0.

Figure 11 shows the curves corresponding to the evolution of water levels in a given tank when the proposed controller was used (controller) and when different manual controls (Man1, Man2, Man3) were used. The solution obtained by the controller is feasible and presents a lower degree of variation (no extreme maximum or 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). Figures 12.a and 12.b compare the changes in the volumes of the three tanks considered, $X(t)$, and the amount of water entering each tank in one hour,

Y(t), using just one of the processors (Figure 12.a) and eight processors (Figure 12.b) to search for the feasible solutions in a given time (each processor processes one trajectory). The solution obtained by using several processors presents the smallest change in the volume of water entering the tanks.

The parallelization of the procedure implemented by the Neuro-Dynamic Programming module is based on two of the alternatives to take advantage of the parallel processing in genetic algorithms that are shown in Figure 8 (alternatives 1 and 2). More specifically, a parallel genetic algorithm is run on the P processors several times, in order to obtain a set of K feasible points where the trajectories start (alternative 2 in Figure 8). Once the beginning of K trajectories (the maximum number of trajectories to be processed) are obtained, the rest of each trajectory is determined by only one of the processors (alternative 1 in Figure 8). The distribution of the trajectories among the processors is done dynamically by allocating a trajectory to each processor when it finishes a previously allocated one.

To give an idea of the computing times and the speedups obtained by the parallel implementation of the procedure, Table 1 and Figure 13 shows the results for different number of trajectories and processors. We have obtained efficiencies between 0.7 and 0.9. These results correspond to the mean values obtained from five executions of each case. As Figure 13 shows, as the number of trajectories grows, the speedup obtained for the higher number of used processors is improved. This is a consequence of the way the parallel algorithm works: once the beginning of the simulated trajectories are determined by the parallel genetic algorithm UPGA (alternative 2 in Figure 8), the rest of each trajectory is determined by several sequential genetic algorithms that are executed practically without communication (USGA). Thus, as the number of trajectories grows, the rate of communication time (required basically in the UPGA) decreases with respect to the computing time required to build the whole trajectory.

Taking into account that control values are required each hour, the times obtained are adequate 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 water volume within the tanks satisfied the constraints imposed and, as shown in Figure 12, the response of the system improved as more trajectories were processed. The times presented in Table 1 can also be improved by implementing a new genetic algorithm able to converge to several optima in only one run, by including niching methods to maintain the diversity of population across the processors and in each processor (alternative 3 in Figure 8).

Table 1. Time for different number of generated trajectories and processors

PROCESSORS	TRAJECTORIES		
	8	**16**	**32**
1	42.0 min.	74.7 min.	145.3 min.
4	13.9 min.	25.6 min.	48.5 min.
8	7.8 min.	11.1 min.	20.1 min.

Finally, we consider that, as the number of tanks increases in the case of cities bigger than the one considered in these experiments, the number of bits that codify each individual in the population grows linearly with the number of tanks. Thus, the search space grows, and the time required by the genetic algorithm to find the solutions corresponding to the feasible points increases. This increase in the computing time comes from the increment in the time required to process each individual in the population, the possible increase in the population size, and in the number of generations required to find a solution. With respect to the population size, there exists some theoretical analysis of the optimal size [32,33], and algorithms that adjust this size according to the probability of selection error [34]. Moreover, some studies have attempted to find the adequate population size empirically and have recommended the use of populations with size ranges between some proposed values [35,36]. Thus, in order to get an insight into the effect of increasing the complexity of the water supply network, we fixed a population size across the different number of tanks (we selected a population that enabled us to find feasible solutions for different numbers of tanks with an error rate below 0.5%), and we performed some experiments that indicate the effect of the increase in the complexity of each individual of the population and the possible increase in the number of generations to obtain a feasible solution. Figure 14 represents the increase in the time required to execute a fixed number of iterations when the number of tanks grows (line *iter=175*). As can be seen in the Figure the time grows less than with a power of two, and slightly more than as a power of 1.9. Figure 14 shows the number of iterations executed to obtain a feasible solution with a 0.5% of error (line *err<0.5%*). In this case, the increase in the time required by the algorithm grows slowly. Thus, a reasonable increase in the number of tanks to be scheduled does not have an important effect on the practical applicability of our scheduling procedure.

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 in relation to

this application. The procedure can be used when the sets of possible states, inputs and control variables are continuous, and applies a genetic algorithm to determine the feasible operation points at each stage, by using a predicted input value. The cost-to-go functions for each feasible state are approximated by Monte Carlo simulation, thus taking into account the influence of 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 previous stage.

The procedure was 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 made 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 capability of 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 possible minima at the same time. Thus, the parallel implementation of a genetic algorithm including niching methods will improve our procedure.

Acknowledgements

The authors would like to thank the referees for their useful comments and suggestions. This work has been financed as part of project CICYT TIC2000-1348; 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 point representation 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.

[29] Damas, M.; Salmerón, M.; Diaz, A.; Ortega, J.; Olivares, G.; Prieto, A.: "Genetic Algorithms and Neuro-Dynamic Programming: Application to Water Supply Networks". 2000 Congress on Evolutionary Computation (CEC2000), pp.7-14, San Diego, July 2000.

[30] Damas, M.; Salmerón, M.; Ortega, J.: "ANNs and Gas for predictive controlling of water supply networks". The IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN2000), pp.365-368, Como (Italy), July 2000.

[31] Wolpert, D.H.; Macready, W.G.:"No Free Lunch Theorems for Optimization". IEEE Transactions on Evolutionary Computation, Vol. 1, No.1, pp.67-82. April, 1997.

[32] Smith, R.: "Population Size". In Handbook of Evolutionary Computation (T. Bäck, D. Fogel, Z. Michalewicz, Eds.), E1.1:1 – E1.1:5, Institute of Physics Publishing Ltd., Bristol and Oxford University Press, 1997.

[33] Eiben, A.E.; Hinterding, R.; Michelewicz, Z.:"Parameter Control in Evolutionary Algorithms". IEEE Trans. on Evolutionary Computation, Vol.3, No.2, pp.124-141. July, 1999.

[34] Smith, R.:"Adaptively resizing populations: an algorithms and analysis". Proc. 5th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, pp.653, 1993.

[35] Grefenstette, J.J.:"Optimization of control parameters for genetic algorithms". IEEE Trans. on Syst., Man, and Cybernet., Vol.16, No. 1, pp.122-128, 1986.

[36] Schaffer, J.; Caruana, R.; Eshelman, L.; Das, R.:"A study of control parameters affecting online performance of genetic algorithms for function optimization". Proc. 3rd Int. Conf. on Genetic Algorithms, Morgan Kaufmann, pp.51-60, 1989.

[37] Belding, T.C.: "The distributed genetic algorithm revisited". In Proceedings of the Sixth ICGA, Morgan Kaufmann, CA, pp.114-121, 1995.

Figures

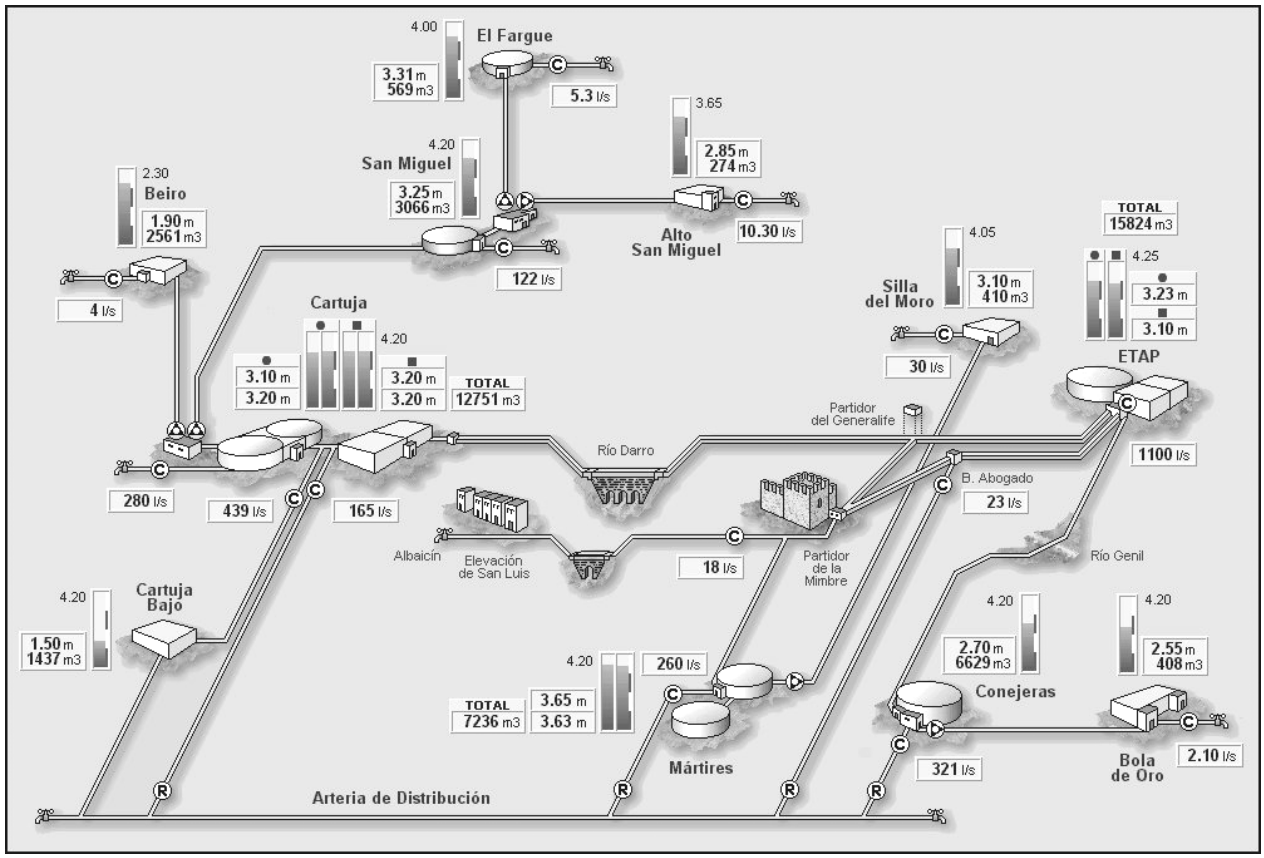

Fig.1. Scheme of the Water Distribution System of Granada (Spain)

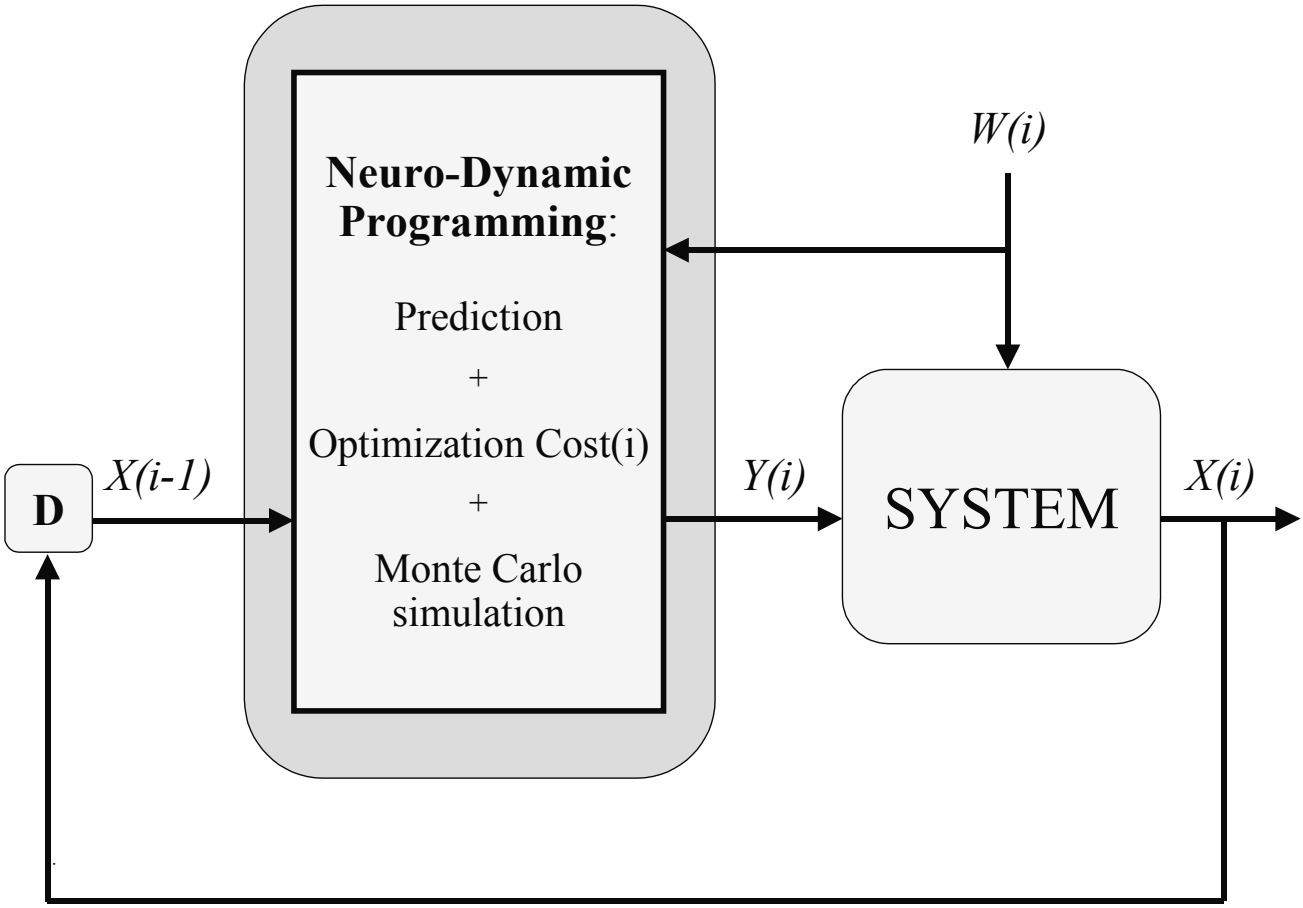

Fig.2. Outline of the system controller

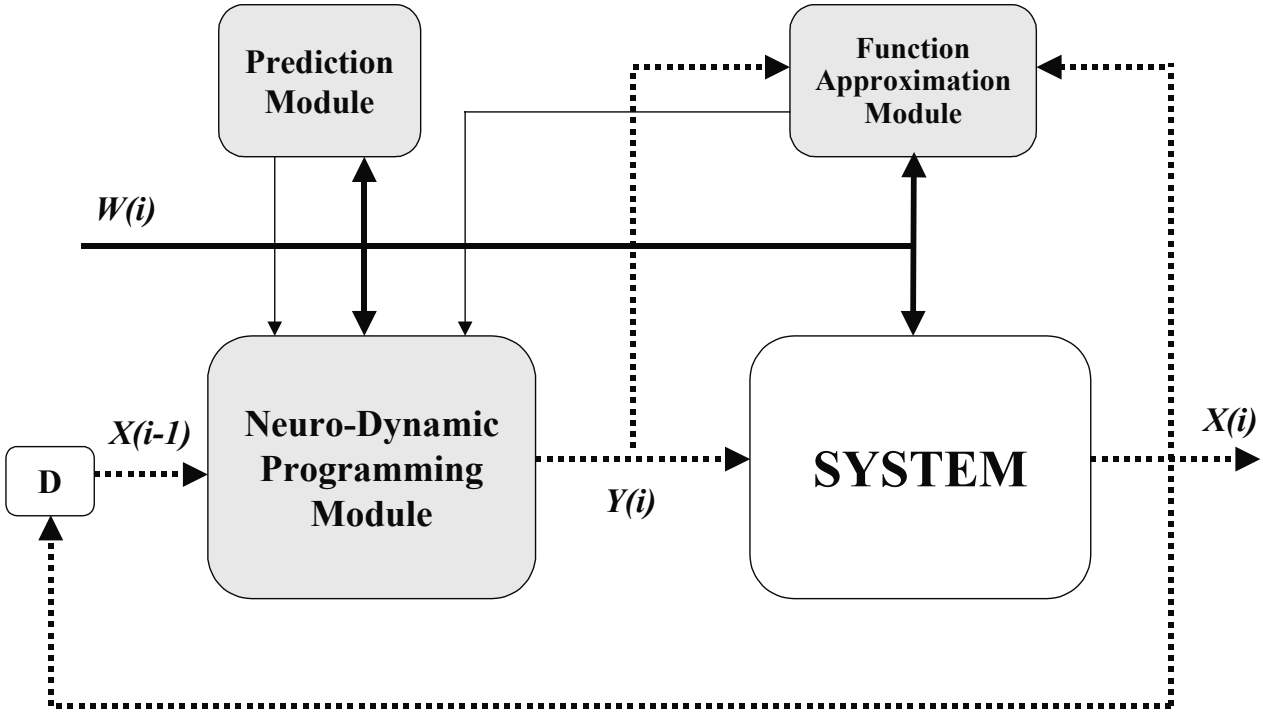

Fig.3. Modules of the proposed hybrid procedure

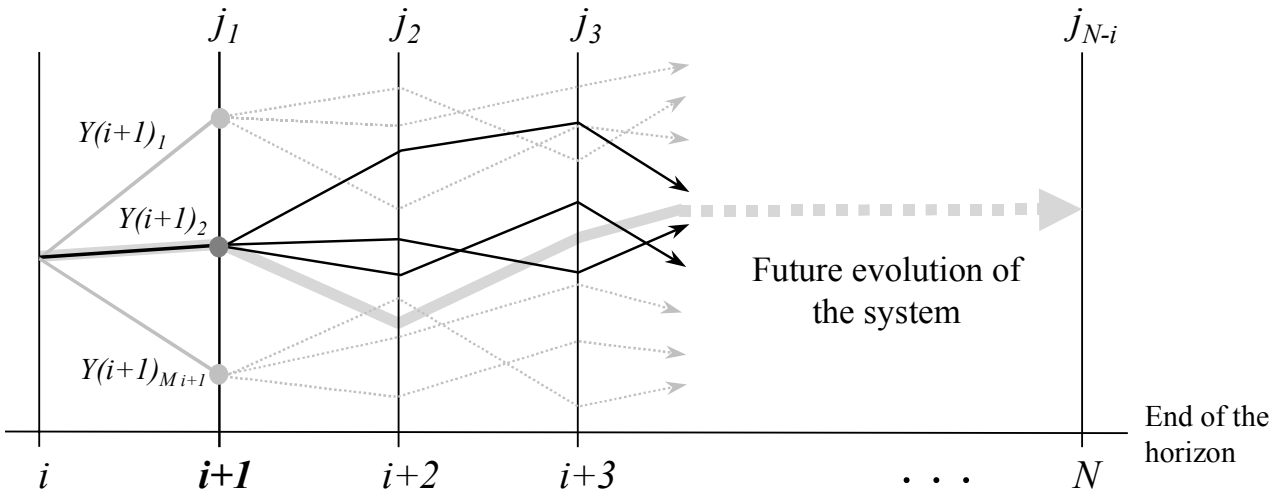

Fig.4. Generation of sample trajectories

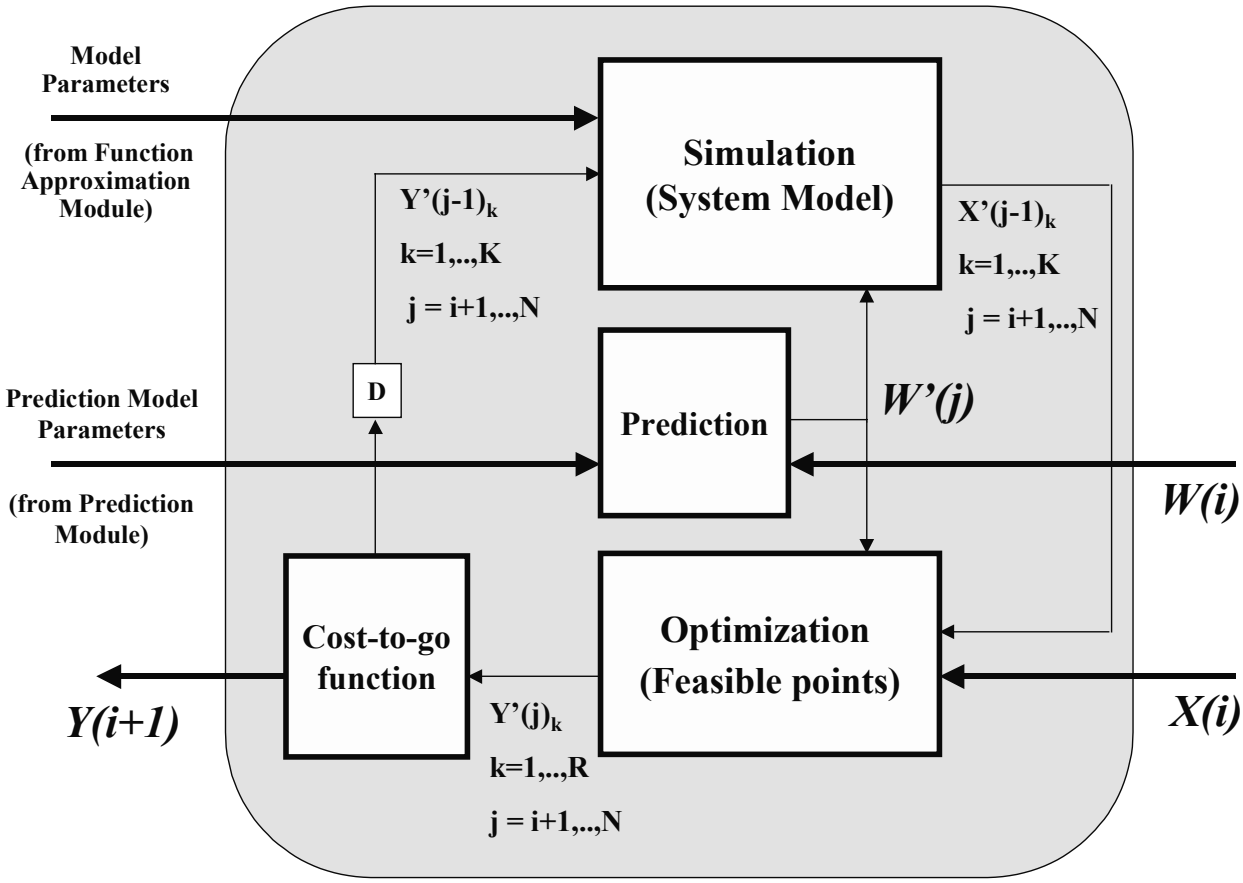

Fig.5. Outline of procedure implemented by the Neuro-Dynamic Programming Module

```
( 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) */
      }
```

Fig.6. Pseudocode for the procedure of 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) */
      }
```

Fig.7. Simplified description of the parallel implementation of the Neuro-Dynamic Programming Module

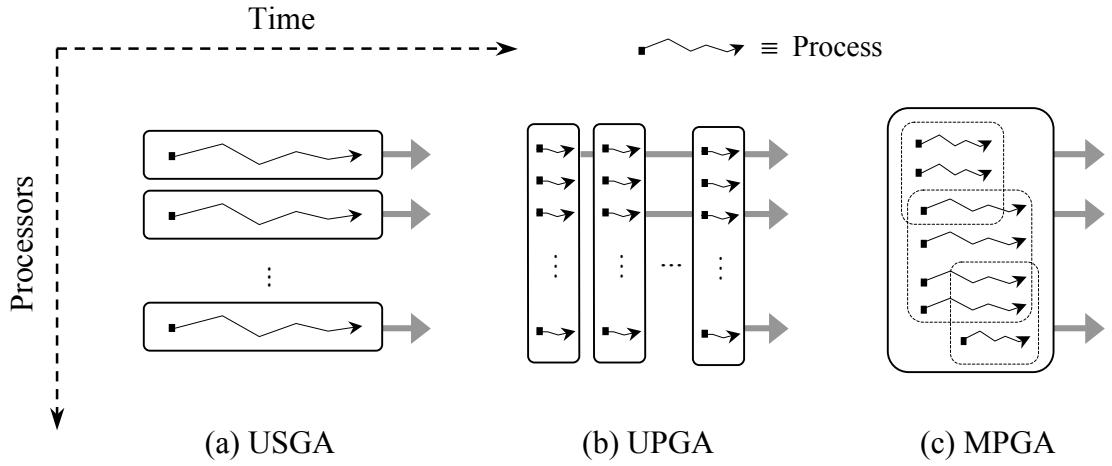

Fig.8. Alternatives for parallelizin the genetic algorithm:
(a) alternative 1, USGA; (b) alternative 2, UPGA; and (c) alternative 3, MPGA

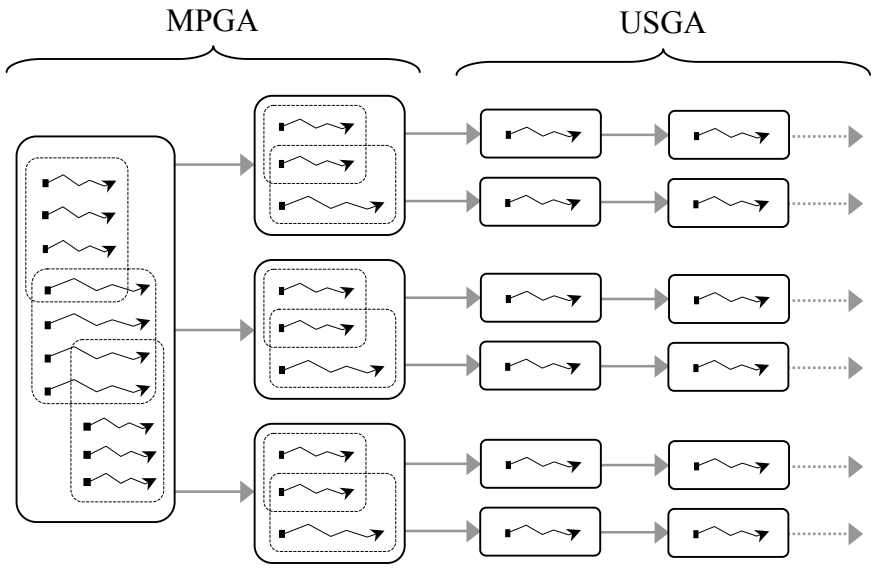

Fig.9. Parallel processing of the procedure with 6 trajectories and 6 processors

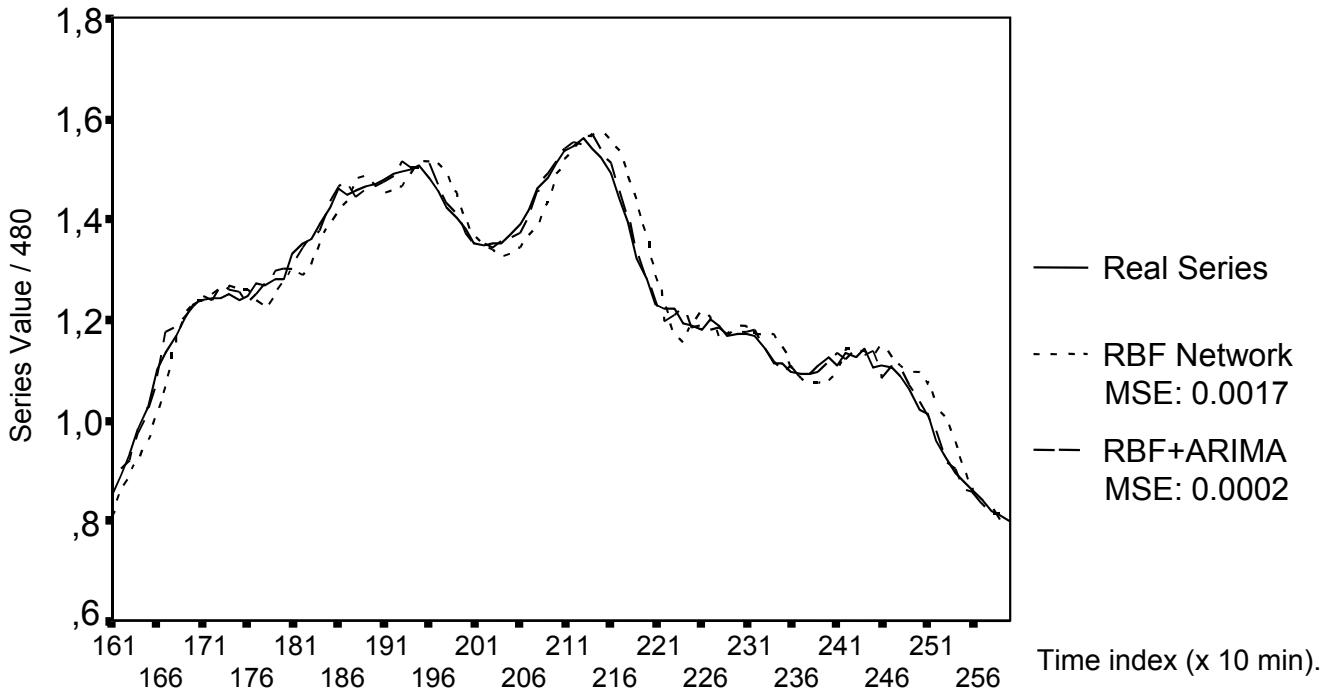

Fig.10. Comparison of the real and the used predicted demands

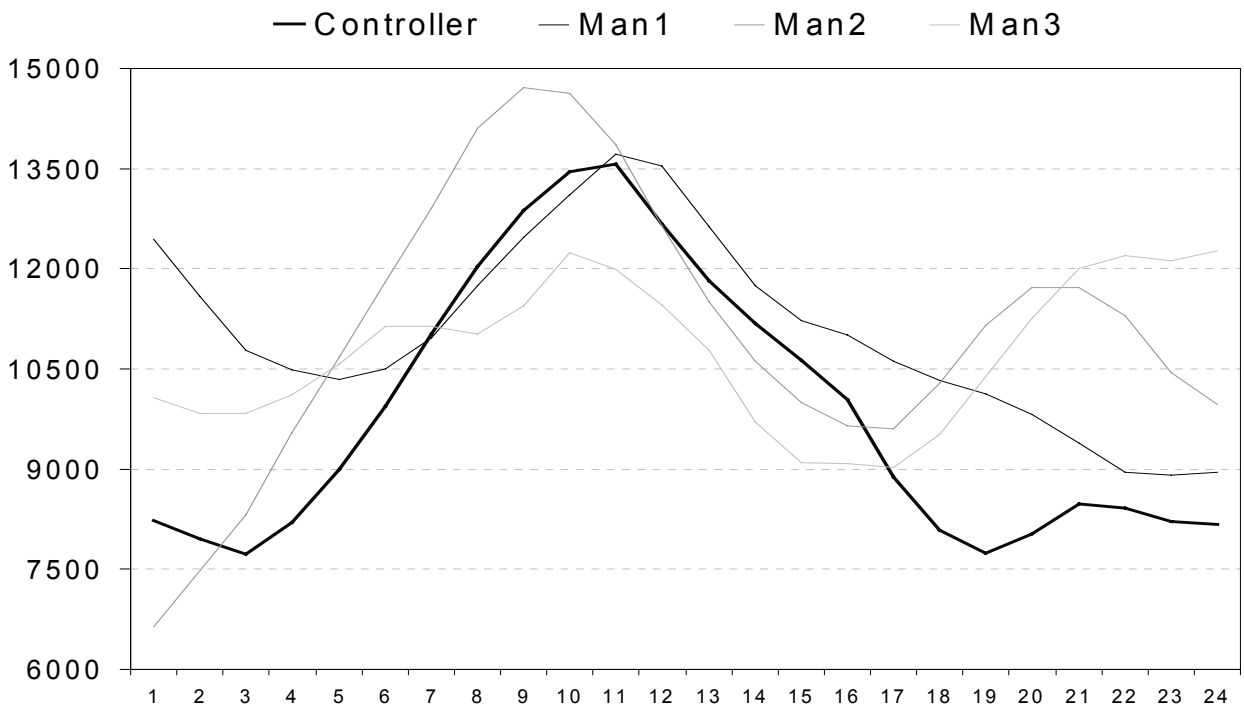

Fig.11. Comparison between different manual control (Man1,Man2,Man3) and the controller obtained with the procedure described

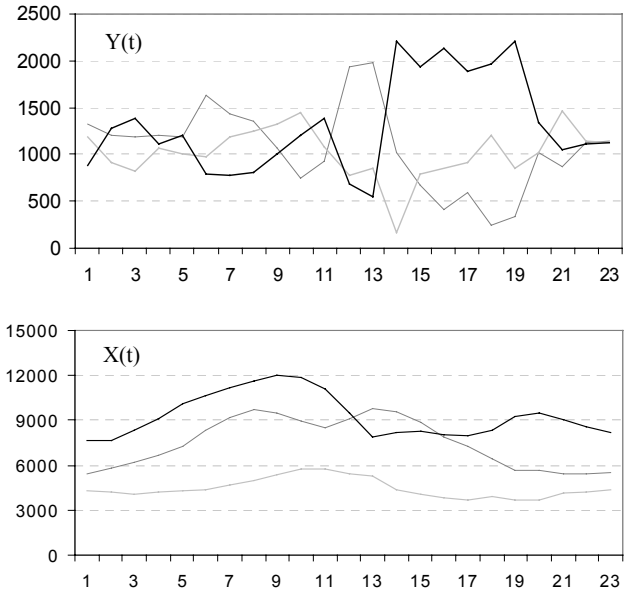

Fig.12.a. Results for 1 trajectory

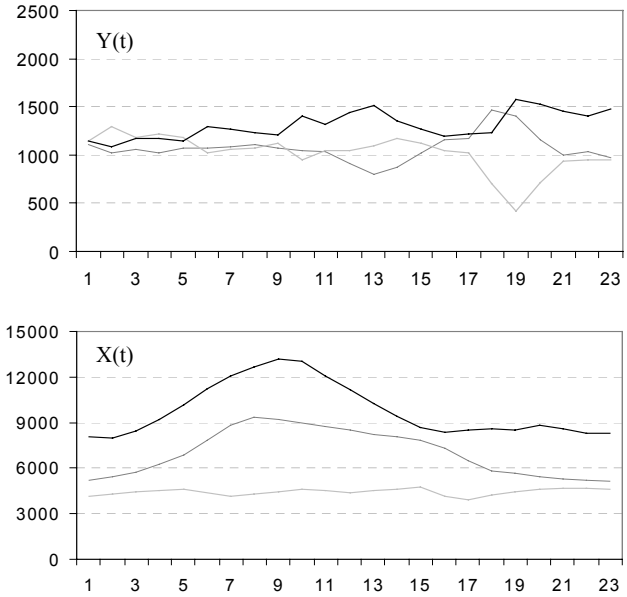

Fig.12.b. Results for 8 trajectories

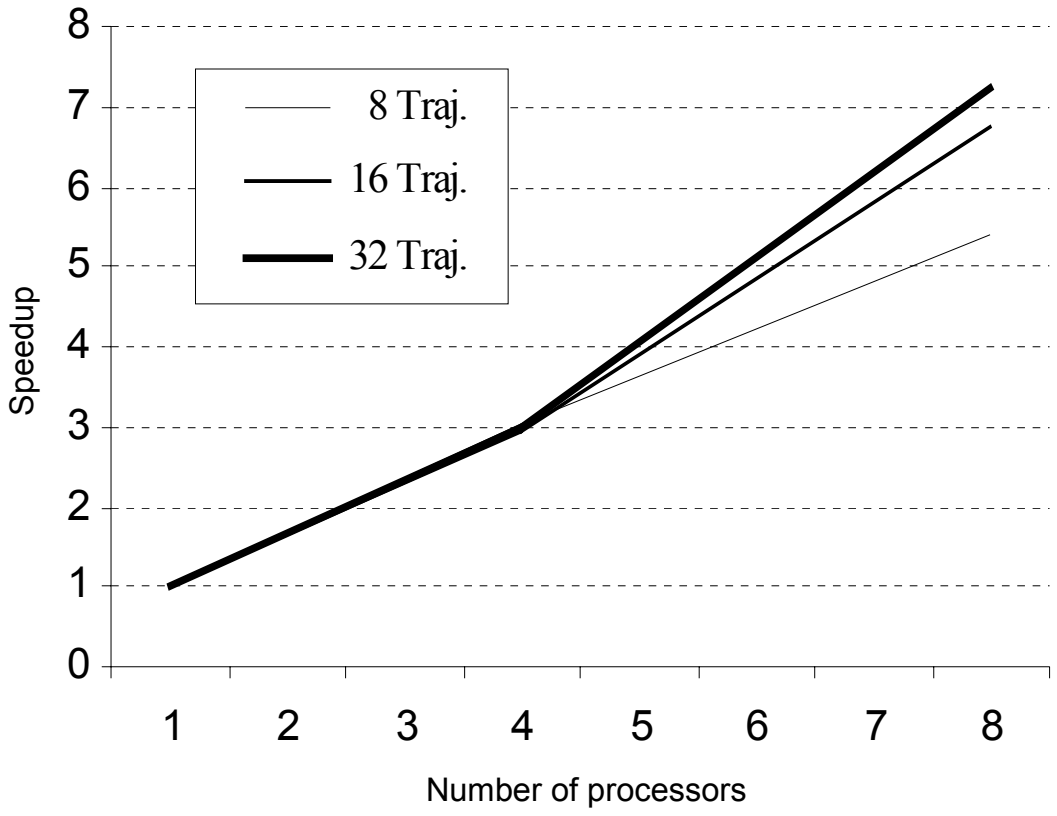

Fig.13. Speedups for different number of generated trajectories and processors

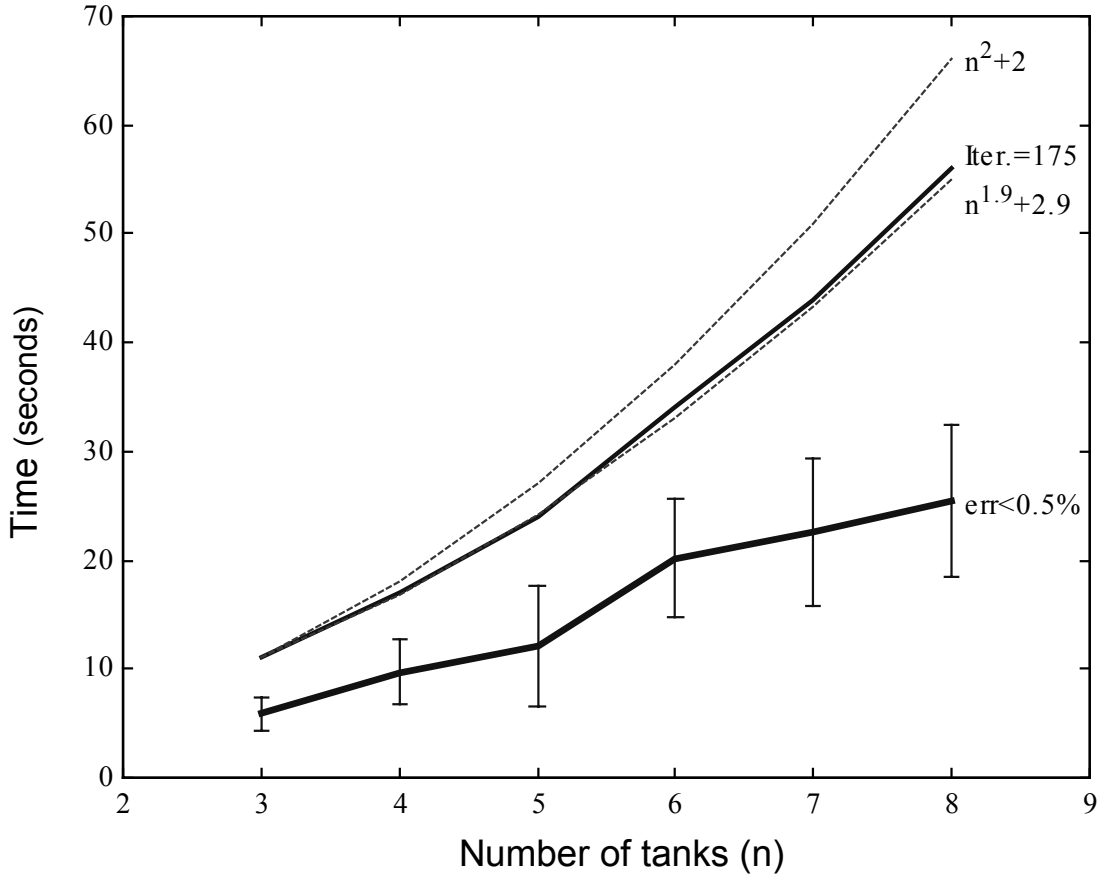

Fig.14. Convergence time vs. number of tanks