International Journal of Production Research
Vol. 00, No. 00, DD Month 2009, 1–17

A Neuroevolutionary Approach to Stochastic Inventory Control in Multi-Echelon Systems

S. D. Prestwich[1], S. A. Tarim[2], R. Rossi[3], and B. Hnich[4]
[1]Cork Constraint Computation Centre, Ireland
[2]Department of Management, Hacettepe University, Ankara, Turkey
[3]Logistics, Decision and Information Sciences Group, Wageningen UR, the Netherlands
[4]Department of Computer Engineering, Izmir University of Economics, Turkey
s.prestwich@cs.ucc.ie, armtar@yahoo.com.tr,
roberto.rossi@wur.nl, brahim.hnich@ieu.edu.tr

(Received 00 Month 200x; In final form 00 Month 200x)

Stochastic inventory control in multi-echelon systems poses hard problems in optimization under uncertainty. Stochastic programming can solve small instances optimally, and approximately solve larger instances via scenario reduction techniques, but it cannot handle arbitrary nonlinear constraints or other non-standard features. Simulation optimization is an alternative approach that has recently been applied to such problems, using policies that require only a few decision variables to be determined. However, to find optimal or near-optimal solutions we must consider exponentially large scenario trees with a corresponding number of decision variables. We propose instead a neuroevolutionary approach: using an artificial neural network to compactly represent the scenario tree, and training the network by a simulation-based evolutionary algorithm. We show experimentally that this method can quickly find high-quality plans using networks of a very simple form.

Keywords Inventory control, neural networks, evolutionary algorithms, neuroevolution, multi-echelon systems

1 Introduction

In the area of optimization under uncertainty one of the most mature fields is *inventory control*. A typical inventory control problem is as follows. Given a planning horizon of N periods and a demand for each period $t \in \{1, \ldots, N\}$, which is a random variable with a given probability density function. Demands occur instantaneously at the beginning of each time period and are *non-stationary* (can vary from period to period), and demands in different periods are independent. A fixed delivery cost a is incurred for each order, a linear holding cost h is incurred for each product unit carried in stock from one period to the next, and a linear stockout cost s is incurred for each period in which the net inventory is negative (it is not possible to sell back excess items to the vendor at the end of a period). The aim is to find a replenishment plan that minimizes the expected total cost over the planning horizon.

A cost-optimal solution can be expressed via a *scenario tree* which specifies an action to be taken in every possible *scenario* (instantiation of the random variables). However, under certain conditions much more compact policies are known to be optimal, or to have desirable properties such as planning stability. For example in (s, S) policies whenever a stock level falls below s it is replenished up to S, while in (R, S) policies the stock level is checked at times specified by R, and if it falls below S then it is replenished up to S. Whereas a scenario tree has exponential size in terms of the number of periods, these special policies have only a linear number of parameters to choose. This makes them popular even in situations in which they are not known to be optimal. For a discussion of inventory control policies see (Silver *et al.* 1998).

In *multi-echelon systems* there are multiple stocking points each with an inventory, linked together in a supply chain. Inventory control in multi-echelon systems is particularly difficult because no simple form of policy is known to be optimal, so in principle we must build a scenario tree. Faced with such problems we may resort to methods based on *simulation*. Simulation alone can only evaluate a plan, but when combined with an optimization algorithm it can be used to find near-optimal solutions (or plans). This approach is called *simulation optimization* (SO) and has a growing literature in many fields including production scheduling, network design, financial planning, hospital administration, manufacturing design, waste management and distribution. It is a practical approach to optimization that can handle problems containing features that make them difficult to model and solve by other methods: for example non-linear constraints and objective functions, and demands that are correlated or have unusual probability distributions. A survey of SO is given in (Fu 2002), and a tutorial and survey of the application of SO to inventory control is given in (Köchel 2007). Relatively little work has been done on applying SO to multi-echelon systems. A pragmatic approach is to use policies such as (s, S) and (R, S), and to apply an evolutionary algorithm (EA) by representing reorder points and replenishment levels as genes. But in general these policies may be highly suboptimal, and a cost-optimal plan for a multi-stage problem must specify an order quantity in every possible scenario. So the plan must be represented via a scenario tree, yet the number of scenarios might be very large, or infinite in the case of continuous probability distributions, making the use of SO problematic. Scenario reduction techniques may be applied to approximate the scenario tree, but it might not always be possible to find a small representative set of scenarios.

Another form of approximation for multi-stage optimization is the use of *linear decision rules*, which assume a simple form of policy based on affine functions of the stochastic decisions. This can lead to tractable problems that can be solved in polynomial time, though with approximate results. See for example (Chen *et al.* 2008) which also explores modified forms of these rules. An alternative form of approximation is to use an *artificial neural network* (ANN) to represent the policy. An ANN is simply a parameterized function whose inputs and outputs are vectors of values, but the interest of ANNs lies in their status as universal function approximators, their many machine learning applications, and their parallels to biological neural networks. In the case of multi-echelon inventory control the inputs to the ANN could be the current stock levels and the current time, and the outputs could be the recommended actions (whether or not to replenish each stocking point and by how much). We may use a general-purpose search procedure such as an EA to tune the parameters of a network so that it minimizes expected costs. This *neuroevolutionary* approach has been applied to control problems (Gomez *et al.* 2008, Hewahi 2005, Stanley and Miikkulainen 2002) and to playing strategies for games such as Backgammon (Pollack and Blair 1998) and Go (Lubberts and Miikkulainen 2001), but not extensively to inventory control, though several papers use EAs for inventory control (Arnold and Köchel 1996, Köchel and Nieländer 2005, Olsen 2003, Prestwich *et al.* 2008b). A related approach to neuroevolution is *genetic programming*, in which an EA is used to evolve an *algorithm* instead of an ANN. Genetic programming has also been applied to inventory control (Kleinau and Thonemann 2004).

Another interesting approach to multi-stage optimization is the field variously referred to as *neuro-dynamic programming*, *temporal difference learning* and *approximate dynamic programming*. This blend of dynamic programming and neural networks (and other forms of approximation) has been applied to many problems including inventory control: see for example (Chaharsooghi *et al.* 2008, Giannoccaro and Pontrandolfo 2002, Jiang and Shenga 2009, Van Roy *et al.* 1997). A drawback is that special techniques are needed to cope with the well-known “curse of dimensionality”: the vast number of states that result from a simple discretization of the continuum of states in these problems. This problem is even worse in multi-echelon systems, where we must discretize several stock levels and order quantities. In contrast, neuroevolution can directly handle a continuum of states.

The operations management community has recently addressed similar problems in inventory control. (Levi *et al.* 2007) address the correlated demand case for a single-item single-location inventory problem and provide computationally efficient policies with constant worst-case performance guarantees. (Graves and Willems 2008) consider the problem of where to place strategic safety stocks in a supply chain, in order to provide a high level of service to the final customer with minimum cost, and extend their model for stationary demand to the case of nonstationary demand. For an extensive coverage of stochastic multi-echelon systems the reader is referred to (de Kok and Graves 2003), which is concerned with the decision-making processes arising in multi-echelon production/inventory systems, and investigates how operations research can support decisions in the design, planning and operation of multi-echelon production/inventory systems.

In this paper we present the first application of neuroevolution to stochastic inventory control in multi-echelon systems. The paper is an extension of (Prestwich *et al.* 2009): we have added a new version of the method that scales better to problems with many time periods, added further experiments to test scalability, described our method more fully, and provided additional references. Section 2 presents our method, Section 3 evaluates the method experimentally, and Section 4 concludes the paper.

2 Neuroevolutionary inventory control

Recall that we wish to find a policy for stochastic inventory control in a multi-echelon system. This policy takes the form of an exponentially large scenario tree, which we will approximate via an ANN whose parameter values must be chosen. We call our method NEMUE (Neuro-Evolution for MUlti-Echelon systems).

2.1 *Inventory control by neural network*

Our ANN input is a vector containing the time period and current inventory levels, and its output is a vector of order quantities (one per stocking point, some or all of which might be zero). We use the ANN to choose order quantities at each stocking point and time period. However, before we can use the ANN we must choose values for its parameters, which are referred to as *weights*.

The process of tuning the weights is called *training*. ANNs come with a ready-made training algorithm: the well-known *backpropagation* algorithm. Given a set of training examples, backpropagation adjusts the ANN weights to minimize the error between the known desired output and the actual output of the ANN. This approach has been applied to inventory control (Gaafar and Choueiki 2000). However, to obtain training data we would first need to solve a set of instances, but the aim of this paper is to investigate a method for doing exactly that. In fact backpropagation is only useful for the class of machine learning problems known as *supervised learning* which is a form of regression, whereas we have a problem in *reinforcement learning*. In reinforcement learning problems we must choose parameters (in this case the ANN weights) in order to maximize a *reward* (in this case to minimize the expected cost).

To choose the weights we can use an EA whose genes are the weights and whose fitness function is the negation of the cost (fitness is conventionally maximized but we aim to minimize cost). In this neuroevolutionary process we evolve a population of chromosomes, each of whose genes specify an ANN. The smaller the cost incurred by using the ANN defined by a chromosome, the fitter the chromosome is considered to be. The hope is that the population will become fitter during evolution, until one or more chromosomes solves the problem by finding an optimal plan.

When using an ANN to solve a problem, an important aspect is the particular form of the ANN. An ANN is typically organised as layers of *units*, each representing a simple *transfer function* such as a sigmoid, limiter or a polynomial function. The ANN is then the composition of these simple functions, with the ANN weights controlling how they are composed. In our experiments we tried different network topologies and transfer functions, including an array of ANNs, one for each time period. Surprisingly, we obtained best results using an extremely simple network: a single layer of units each with the identity transfer function $f(x) = x$. This counter-intuitive result is explained by the fact that the ANN forms only part of the policy (see below). We will continue to refer to the function as an ANN, but it is not the form of ANN used by most researchers. We now describe the function in detail.

We use two alternative representations of the time period t: a *direct encoding* in which t is a single ANN input represented by an integer $t = 1 \dots P$, and a *unary encoding* in which we associate a binary variable with each period, and period t is represented by a binary vector $(0_1, \dots, 0_{t-1}, 1, 0_{t+1}, \dots, 0_P)$. The unary encoding uses more ANN inputs than the direct encoding, but is a technique often used to represent symbolic ANN inputs and sometimes gives better results when a numerical input can take only a few values. The ANN with unary time encoding represents a set of affine functions

$$O_{tj} = a_{tj} + \sum_{i=1}^{K} S_{ti} b_{ij} \quad (t = 1 \dots P,\ j = 1 \dots K)$$

where O_{tj} is the order quantity for stocking point j at time t, S_{ti} is the stock level for stocking point i at time t, K is the number of stocking points, and the a_{tj} and b_{ij} are parameters to be tuned. (An affine transformation is a linear transformation followed by a translation.) This ANN has only $K(P + K)$ parameters, so if we can successfully represent a scenario tree with it then we have achieved an exponential compression of the tree. The ANN with direct time encoding represents a different set of affine functions

$$O_{tj} = a_j + \sum_{i=1}^{K} S_{ti} b_{ij} + \sum_{k=1}^{C} T_k(t) c_{kj} \quad (t = 1 \dots P,\ j = 1 \dots K)$$

where the $T_k(t)$ are the Chebyshev polynomials of the first kind $T_1(t) = t$, $T_2(t) = 2t^2 - 1$, $T_3(t) = 4x^3 - 3x$, $T_4(t) = 8x^4 - 8x^2 + 1 \dots$ Chebyshev polynomials are a well-known family of functions used for function approximation, and therefore an interesting candidate for approximating policy functions. The a_{tj}, b_{ij} and c_{kj} are parameters to be tuned. There are now only $K(K + C + 1)$ parameters so the size of the ANN is independent of the number of time periods P. This might make NEMUE more scalable on problems with more time periods, but its policy will be capable of less complex behaviour so the plan quality might suffer. The hope is that a fairly small value of C will suffice to represent sufficiently complex policies, and we arbitrarily choose $C = 4$.

2.2 *Handling constraints*

The ANN forms only part of the policy. We also need a way of handling the constraints of the problem, which forbid (i) negative orders (corresponding to selling unused stock back to the supplier), and (ii) negative stock levels; both types of constraint might be violated by the O_{tj} values recommended by the ANN. We will train the ANN by an EA and there are several ways of handling constraints in EAs. We use a *decoder* which transforms the (possibly infeasible) ANN solution into one that violates no constraints. In EA terminology, a decoder is any method for finding a feasible solution from a chromosome representing a non-solution. Decoders are problem-specific and ours works as follows. Suppose at period t we have stock levels S_{ti} and the ANN suggests ordering quantities O_{tj}. We modify each quantity O_{tj} by

$$O_{tj} \leftarrow \max(O_{tj}, 0)$$

to avoid violating constraints of type (i). Then for any stocking point i that supplies a set of stocking points X_i we modify its order level O_{tj} by

$$O_{tj} \leftarrow \max\left(O_{tj}, \left(\sum_{k \in X_i} O_{tk}\right) - S_{ti}\right)$$

This ensures that each supplier orders sufficient stock to fulfil its deliveries, and avoids violating constraints of type (ii). The policy is now the composition of the ANN and the decoder, which transforms the affine function of the ANN into a continuous piecewise affine function.

Note that we must modify the order levels of the stocking points earlier in the supply chain first. This is always possible if the supply chain is in the form of a directed acyclic graph. If there are complications such as constraints on order sizes or storage capacities then the decoder must be modified, but we leave this issue for future work.

We used a decoder to handle the problem constraints, but there are other ways of handling constraints in EAs. The simplest is to use a *penalty function* which adds a large artificial cost for each violated constraint. In our problem this forces the ANN to learn to order sufficient stock in order to avoid stockout. We tried a penalty function but it gave inferior results to the decoder.

2.3 *The evolutionary algorithm*

To train the ANN we use an EA. There are many such algorithms in the literature and the choice is somewhat arbitrary. We decided to use a $(\mu + 1)$-Evolution

Strategy (ES) (Bäck *et al.* 1991) because of its simplicity, and because an efficient method for handling noise in the fitness function is known for similar algorithms (Prestwich *et al.* 2008a). However, we use a slightly more complicated *cellular* ES: see (Alba and Dorronsoro 2008) for example. Cellular algorithms mimic the evolution of cellular organisms that communicate only with their neighbors, and can reduce the likelihood of *premature convergence* in which an EA's chromosomes become trapped near a local optimum. In a cellular ES each chromosome is notionally placed in an artificial space and nearby chromosomes form its neighborhood. A common way of defining neighborhoods is to number the chromosomes $0 \ldots \mu - 1$, and for the neighbors of chromosome i choose chromosomes $(i \pm j)$ mod μ for $i = 1 \ldots n$ and some neighborhood size n (we use $n = 1$). In our ES the population size is μ, at each iteration a new chromosome c' is created by *mutating* a randomly selected chromosome c, and if c' is fitter than the least-fit chromosome c^* in the neighborhood of c then it replaces c^*, otherwise c' is discarded. Mutation is the random modification of gene values, analogous to noise in Simulated Annealing.

A common form of mutation adds normally distributed noise to each gene, but we use a method that gave better results in experiments. For each chromosome we generate two uniformly distributed random numbers, p in the range $(0, 1)$ and q in the range $(0, 0.5)$. Then for each allele (gene value) in the chromosome, with probability p we change it, otherwise with probability $1 - p$ we leave it unchanged; this is a form of *masking*. If we do change it then with probability q we set it to 0, otherwise with probability $1 - q$ we add to it a random number with Cauchy distribution. This is called *Cauchy mutation* and it has been shown to speed up EAs (Yao and Lin 1999). It can be computed as $s \tan(u)$ where u is a uniformly distributed random variable in the range $(-\pi, \pi)$ and s is a scale factor. For each chromosome we compute a random scale factor, itself with Cauchy distribution and fixed scale factor 100. Finally, if no allele was modified (which is possible for small p) then we modify one randomly selected allele as described. This rather complex mutation operator is designed to generate a variety of random moves, with different numbers of modified alleles and different scale factors. All chromosomes initially contain alleles generated randomly using the same Cauchy distribution.

2.4 *Handling uncertainty*

When demand is probabilistic the fitness function of the EA is noisy. In such cases we must average costs over a number of simulations. In some previous SO approaches to inventory control this problem was tackled by averaging costs over a small number of simulations, because the simulations were computationally expensive: for example (Köchel and Nieländer 2005) use 3 samples. The standard deviation of the sample mean of a random variable with standard deviation σ is $\sigma/\sqrt{n}$ where n is the number of samples, so a large number of samples may be needed for very noisy fitness functions. Here we use smaller problems than those in (Köchel and Nieländer 2005) so we can afford to use a much larger number of simulations and obtain reliable cost estimates. To do this for every chromosome would be expensive but there are more efficient methods.

Several alternative techniques for handling fitness noise in EAs are surveyed in (Beyer 2000, Jin and Branke 2005). A popular approach is to use a *Noisy Genetic Algorithm* (NGA) which computes the fitness of each chromosome by averaging over a number of samples (Fitzpatrick and Grefenstette 1988, Gopalakrishnan *et al.* 2001, Miller 1997, Miller and Goldberg 1996). This wastes considerable time evaluating unpromising chromosomes, but it can be improved by linearly increasing the number of samples with search time, starting from a low value (Smalley *et al.* 2000, Wu *et al.* 2006). However, though NGAs have been used to solve real problems, they may not be the most efficient approach. An alternative technique is to *resample* chromosome fitness: that is, chromosome fitness estimates are periodically refined by taking additional samples (Arnold and Beyer 2002, Bui *et al.* 2005, Hughes 2001, Prestwich *et al.* 2008a, Stagge 1998, Stroud 2001, Then and Chong 1994).

We use the *greedy averaged sampling* resampling scheme of (Prestwich *et al.* 2008a). This requires two parameters to be tuned by the user: U and S. On generating a new chromosome c it takes S samples to estimate its fitness before placing it into the population. It then selects another chromosome c' (which may be c) for *resampling*: another S samples are taken for c' and used to refine its fitness estimate. c' is the chromosome with highest fitness among those with fewer than U samples, so the function of U is to prevent any chromosome from being sampled more times than necessary. If all chromosomes in the population have been sampled U times then no resampling is performed. The algorithm is summarized in Figure 1.

The aim of this resampling method is to obtain chromosomes with good fitness averaged over many samples, while expending a smaller number of samples on less-promising chromosomes. Using small S also has an effect beyond reducing the average number of samples per chromosome: it encourages exploration by preserving less-fit chromosomes for longer. We found this to be a very beneficial effect.

Some points are glossed over in Figure 1 for the sake of readability. Firstly, if S is not a divisor of U then fewer than S samples are needed in the final resampling of any chromosome to bring its total to U. Secondly, if no chromosome has U samples on termination then we must choose another chromosome to return. To avoid this, S should be assigned a sufficiently large value so that in experiments there is always a chromosome with U samples on termination. This value must be chosen by experimentation.

3 Experiments

Ultimately we are interested in solving large, realistic inventory problems with multiple stocking points, stochastic lead times, correlated demands and other features that make classical approaches impractical. Unfortunately there are no known methods for solving such problems to optimality, so there is no way of evaluating our method on problems of arbitrary form. Instead in Section 3.1 we consider more modest problems to test the ability of NEMUE to find good plans, and in Section 3.2 we test the method on larger problems with special forms and known solutions.

3.1 *Problems solvable by stochastic programming*

Our benchmark problems have two multi-echelon topologies: *arborescent* and *serial*. In the arborescent case we have three stocking points A, B and C, with C supplying A and B, while in the serial case C supplies B which supplies A. In both cases we have linear holding costs, linear penalty costs, fixed ordering costs, and stationary probabilistic demands. The closing inventory levels for period t are $I_t^A = I_{t-1}^A + Q_t^A - d_t^A$, $I_t^B = I_{t-1}^B + Q_t^B - d_t^B$ and $I_t^C = I_{t-1}^C + Q_t^C - Q_t^A - Q_t^B$ where Q_t is the order placed in period t and d_t is the demand in period t. If $I_t < 0$ then the incurred cost is $-I_t.\pi$, otherwise it is $I_t.h$, where π is the penalty cost and h the holding cost. Suppliers are not allowed to run out of stock. Lead times are assumed to be deterministic and, without loss of generality, equal to zero. We prepared 28 instances of both the arborescent and serial types, with various costs and numbers of time periods, giving a total of 56 instances with a range of characteristics. For space reasons we do not specify the demands in detail, but we used 10 demand patterns for arborescent instances and 4 patterns for serial instances. In each period we specify a deterministic demand which is then multiplied by either $\frac{2}{3}$ with probability 0.25, 1 with probability 0.5, or $\frac{4}{3}$ with probability 0.25. Thus the number of possible scenarios is 3^P, giving 59,049 scenarios for the largest problems ($P = 10$).

We solved these problems in two ways: using Stochastic Programming (SP) (Birge and Louveaux 1997) and NEMUE. SP is a field of Operations Research designed to solve optimization problems under uncertainty via scenario reduction techniques: a representative subset of all possible scenarios is selected and used to generate a deterministic equivalent optimization problem, which is then typically solved using integer linear programming. We use the SP results to evaluate the quality of plans found by NEMUE. The optimal replenishment plans are obtained using the following Stochastic Integer Programming model:

$$
\begin{array}{l}
\min \mathsf{E}[C] = \sum_{t=1}^{N} \sum_{p \in P} \left(a_p \delta_{pt} + h_p I_{pt}^{+} + \pi_p I_{pt}^{-} \right) \\
\text{s.t. } t = 1, \ldots, N \text{ and } p \in P \\
I_{pt} = I_{p,t-1} + Q_{pt} - Q_{P_p,t} - d_{pt} \\
I_{pt} = I_{p,t}^{+} - I_{p,t}^{-} \\
Q_{pt} \leq M \delta_{pt} \\
\delta_{pt} \in \{0, 1\} \quad Q_{pt} \geq 0
\end{array}
$$

where

C :	total holding and ordering/set-up cost of the system over N periods;
a :	fixed ordering/set-up cost;
h :	proportional inventory holding cost per period;
P :	the set of all stocking points;
P_p :	the set of stocking points supplied directly by the stocking point p;
d_{pt} :	random demand at stocking point p, in period t;
δ_{pt} :	a binary variable that takes the value of 1 if a replenishment occurs
:	at stocking point p in period t and 0 otherwise;
I_{pt} :	the inventory level at the end of period t at stocking point p;
Q_{pt} :	the order quantity at the beginning of period t at stocking point p;

and I^{+} and I^{-} denote positive and negative closing inventory levels. Except for the lowest echelon stocking points, I^{-} is zero. M is some large positive number. In this stochastic model a *here-and-now* policy is adapted: all decision variables are set before observing the realisation of the random variables. The certainty equivalent model is obtained using the compiler described in (Tarim *et al.* 2006) and solved with CPLEX 11.2.

The computational results are given in Table 1. All SP and NEMUE runs took one hour on a 2.8 GHz Pentium (R) 4 with 512 RAM, each NEMUE figure being the best of 12 five-minute runs. NEMUE with the unary encoding is denoted NEMUE^u and with the direct encoding NEMUE^d. The NEMUE parameters used were $S = 1$, $\mu = 50$ and $U = 10000$. The columns marked "%opt" denote the optimality gap: a reported cost c and gap g means that SP proved that the optimal solution cannot have cost lower than $c' = c(100 - g)/100$ (this does not imply the existence of a solution with cost c'). In several cases NEMUE found superior plans to those found by SP, showing that on larger instances SP fails to find optimal plans.

In a few cases NEMUE found plans that appeared to be slightly better than optimal. This is because we estimate the expected cost by sampling, and it may be an over- or under-estimate. An under-estimated cost for an optimal plan will of course appear to be better than optimal: for example serial instance 1 has optimum cost 995, but NEMUE^u found a plan with estimated cost 993 and NEMUE^d 981. With fewer samples the difference is greater, for example with $U = 1000$ NEMUE^u found cost

986 and NEMUE^d 976. This effect cannot be completely eradicated but it can be reduced by increasing U: with $U = 30000$ both NEMUE^u and NEMUE^d found estimated cost 994. In Table 1 we take a better-than-optimal cost estimate to indicate an optimal plan.

SP was unable to find provably optimal plans for all but the smallest instances. We believe that for the medium-sized instances SP finds optimal plans but does not prove optimality before timeout. For the largest instances SP ran out of memory, though we use a state-of-the-art CPLEX solver on a powerful machine. On the largest instances for which SP did not run out of memory, it was unable to prove optimality even within several days. Thus our benchmarks straddle the borderline of solvability by classical methods.

We found that for both serial and arborescent problems, under both the direct and unary encodings, the use of multiple short runs was very helpful. Especially for problems with more periods, the best solutions were found only in a minority of runs. Long runs appeared to be less useful and most improvements were found in the first few minutes. This may indicate premature convergence of the EA but not necessarily: the use of "random restarts" is common in EAs and other metaheuristics such as local search algorithms.

Despite the simplicity of its policies and the large number of scenarios (at least on the larger instances) the NEMUE results are remarkably good. The NEMUE^d results are worse than those of NEMUE^u in almost all cases, and sometimes much worse. However, on 13 of the 28 arborescent instances and 19 of the 28 serial instances, NEMUE^u found plans that were at least as good as those found by SP. On the three serial instances for which SP found provably optimal plans, NEMUE^u found equally good plans. On most of the largest instances NEMUE^u found better plans than SP. These results indicate that: (i) a relatively simple, continuous, piecewise affine function can closely approximate a large policy tree for multi-echelon systems; (ii) such a function can be effectively represented by an affine function followed by a decoder function; (iii) the affine function can be learned in a reasonable time by evolutionary search; (iv) that our approach is more scalable than SP; (v) the unary time encoding can express better policies than the direct encoding (at least those policies that can easily be found by EA).

3.2 *Larger problems*

We now consider what happens when the number of periods increases further. Clearly NEMUE will find feasible policies (because of its decoder all its policies are feasible) while SP will run out of memory, but how well does NEMUE^u scale up, and does it still beat NEMUE^d? We constructed two larger classes of problem to investigate these questions, one serial and one arborescent. Both have 3 stocking points as above, 4 periods, and optimal plans that repeat every 2 periods. The latter fact means that we can construct arbitrarily large problems with $P = 4N$ periods by repeating the demand patterns, and in each case the optimum cost will be N times that of the original 4-period problem. NEMUE does not exploit this knowledge and we can examine how it scales up as N increases. The serial problem has an optimal plan with cost 1287 while the arborescent problem has an optimal plan with cost 3654.

We tested NEMUE^u and NEMUE^d on these two problems with $N \in \{5, 10, 15, 20, 25\}$ ($P \in \{20, 40, 60, 80, 100\}$) and recorded the cost after 3, 10, 30, 100 and 300 seconds. The results are shown in Table 2 and an interesting pattern emerges: NEMUE^d scales much better than NEMUE^u when runtime is limited. The greater the number of time periods the more pronounced the effect, and more so on the arborescent problems. On the serial problems NEMUE^d is always worse than NEMUE^u after 300 seconds but the difference decreases as P increases, and after shorter runtimes NEMUE^d still beats NEMUE^u. The most likely explanation for the superior scalability of NEMUE^d is that it has a constant number of parameters to learn, whereas NEMUE^u has more parameters as P increases; this makes learning more difficult so the EA takes longer. Given sufficient time NEMUE^u should surpass NEMUE^d, for example after 1 hour NEMUE^u found a plan for the arborescent problem with 20 periods with cost 19860 which is 8.7% optimal, while NEMUE^d never progresses beyond 11.5%.

In conclusion, which version of NEMUE is the best depends on the application. NEMUE^u finds better policies and is recommended if the number of periods is moderate, or if we have a great deal of time in which to find a plan. However, if we require a good plan in a limited time, or if the number of periods is large, then NEMUE^d is better.

4 Conclusion

We have proposed the first neuroevolutionary method for approximating optimal plans in multi-echelon stochastic inventory control problems. Large or infinite scenario trees are approximated by a neural network, which is trained by an evolutionary algorithm with resampling, while problem constraints are handled by a decoder. Because the method is simulation-based and uses general-purpose techniques such as evolutionary algorithms and neural networks, it does not rely on special properties of the problem and can be applied to inventory problems with non-standard features. We showed experimentally that the method can find near-optimal solutions. In future work we will extend the method to handle problem features such as capacity constraints, and automatically evolve neural networks via methods such as that of (Stanley and Miikkulainen 2002).

Acknowledgements This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 05/IN/I886. B. Hnich and S. A. Tarim are supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under the Support Programme-1001. Roberto Rossi has received funding from the European Community's Seventh Framework Programme (FP7) under grant agreement no. 244994 (project VEG-i-TRADE).

REFERENCES

[Alba and Dorronsoro 2008] E. Alba, B. Dorronsoro. Cellular Genetic Algorithms. *Operations Research Computer Science Interfaces Series*, Vol. 42, Springer, 2008.

[Arnold and Beyer 2002] D. V. Arnold, H.-G. Beyer. Local Performance of the (1+1)-ES in a Noisy Environment. *IEEE Trans. Evolutionary Computation* 6(1):30–41, 2002.

[Arnold and Köchel 1996] J. Arnold, P. Köchel. Evolutionary Optimisation of a Multi-Location Inventory Model With Lateral Transshipments. *9th International Working Seminar on Production Economics*, Igls 1996, Preprints 2, pp. 401–412.

[Bäck *et al.* 1991] T. Bäck, F. Hoffmeister, H.-P. Schwefel. A Survey of Evolution Strategies. *4th International Conference on Genetic Algorithms*, 1991.

[Beyer 2000] H.-G. Beyer. Evolutionary Algorithms in Noisy Environments: Theoretical Issues and Guidelines for Practice. *Computer Methods in Applied Mechanics and Engineering* 186(2–4):239–267, 2000.

[Birge and Louveaux 1997] J. R. Birge, F. Louveaux. Introduction to Stochastic Programming. Springer, New York, 1997.

[Bui *et al.* 2005] L. T. Bui, H. A. Abbass, D. Essam. Fitness Inheritance for Noisy Evolutionary Multi-Objective Optimization. *Genetic and Evolutionary Computation Conference*, Washington DC, USA, ACM Press, 2005.

[Chaharsooghi *et al.* 2008] S. K. Chaharsooghi, J. Heydari, S. H. Zegordi. A Reinforcement Learning Model for Supply Chain Ordering Management: an Application to the Beer Game. *Decision Support Systems* 45(4):949–959, 2008.

[Chen *et al.* 2008] X. Chen, M. Sim, P. Sun, J. Zhang. A Linear Decision-Based Approximation Approach to Stochastic Programming. *Operations Research* 56(2):344–357, 2008.

[de Kok and Graves 2003] A. G. de Kok, S. C. Graves (eds). Handbook in Operations Research and Management Science, Volume 11: Supply Chain Management: Design, Coordination and Operation. Elsevier, Amsterdam, 2003.

[Fitzpatrick and Grefenstette 1988] J. M. Fitzpatrick, J. J. Grefenstette. Genetic Algorithms in Noisy Environments. *Machine Learning* 3:101–120, 1988.

[Fu 2002] M. C. Fu. Optimization for Simulation: Theory vs Practice. *INFORMS*

Journal of Computing 14:192–215, 2002.

[Gaafar and Choueiki 2000] L. K. Gaafar, M. H. Choueiki. A Neural Network Model for Solving the Lot-Sizing Problem. *Omega* 28(2):175–184, 2000.

[Giannoccaro and Pontrandolfo 2002] I. Giannoccaro, P. Pontrandolfo. Inventory Management in Supply Chains: a Reinforcement Learning Approach. *International Journal of Production Economics* 78(2):153–161, 2002.

[Gomez *et al.* 2008] F. Gomez, J. Schmidhuber, R. Miikkulainen. Efficient Non-Linear Control Through Neuroevolution. *Journal of Machine Learning Research* 9:937–965, 2008.

[Gopalakrishnan *et al.* 2001] G. Gopalakrishnan, B. S. Minsker, D. Goldberg. Optimal Sampling in a Noisy Genetic Algorithm for Risk-Based Remediation Design. *World Water and Environmental Resources Congress*, ASCE, 2001.

[Graves and Willems 2008] S. C. Graves, S. Willems. Strategic Inventory Placement in Supply Chains: Non-Stationary Demand. *Manufacturing and Service Operations Management* 10(2):278–287, 2008.

[Hewahi 2005] N. M. Hewahi. Engineering Industry Controllers Using Neuroevolution. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 19(1):49–57, 2005.

[Hughes 2001] E. J. Hughes. Evolutionary Multi-objective Ranking with Uncertainty and Noise. *First International Conference on Evolutionary Multi-Criterion Optimization, Lecture Notes In Computer Science* vol. 1993, Springer-Verlag, 2001, pp. 329–343.

[Jiang and Shenga 2009] C. Jiang, Z. Shenga. Case-Based Reinforcement Learning for Dynamic Inventory Control in a Multi-Agent Supply-Chain System. *Expert Systems with Applications* 36(3 part 2):6520–6526, 2009.

[Jin and Branke 2005] Y. Jin, J. Branke. Evolutionary Optimization in Uncertain Environments — a Survey. *IEEE Transactions on Evolutionary Computation* 9(3):303–317, 2005.

[Kleinau and Thonemann 2004] P. Kleinau, U. W. Thonemann. Deriving Inventory-Control Policies With Genetic Programming. *OR Spectrum* 26(4):521–546, 2004.

[Köchel 2007] P. Köchel. Simulation (Optimisation) in Inventory Theory. Tutorial, *8th ISIR Summer School on New and Classical Streams in Inventory Management: Advances in Research and Opening Frontiers*, 2007.

[Köchel and Nieländer 2005] P. Köchel, U. Nieländer. Simulation-Based Optimisation of Multi-Echelon Inventory Systems. *International Journal of Production Economics* 1:503–513, 2005.

[Levi *et al.* 2007] R. Levi, M. Pál, R. Roundy, D. Shmoys. Approximation Algorithms for Stochastic Inventory Control Models. *Mathematics of Operations Research* 32(2):284–302, 2007.

[Lubberts and Miikkulainen 2001] A. Lubberts, R. Miikkulainen. Co-Evolving a Go-Playing Neural Network. *Genetic and Evolutionary Computation Confer-*

ence, Kaufmann, 2001, pp. 14–19.

[Miller 1997] B. L. Miller. Noise, Sampling, and Efficient Genetic Algorithms. PhD thesis, University of Illinois, Urbana-Champaign, 1997.

[Miller and Goldberg 1996] B. L. Miller, D. E. Goldberg. Optimal Sampling for Genetic Algorithms. *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 6, ASME Press, 1996, pp. 291–298.

[Olsen 2003] A. L. Olsen. An Evolutionary Algorithm for the Joint Replenishment of Inventory with Interdependent Ordering Costs. *Genetic and Evolutionary Computation Conference*, *Lecture Notes in Computer Science* vol. 2724, Springer, 2003, pp. 2416–2417.

[Pollack and Blair 1998] J. B. Pollack, A. D. Blair. Co-Evolution in the Successful Learning of Backgammon Strategy. *Machine Learning* 32(3):225–240, 1998.

[Prestwich *et al.* 2008a] S. D. Prestwich, S. A. Tarim, R. Rossi, B. Hnich. A Steady-State Genetic Algorithm With Resampling for Noisy Inventory Control. *10th International Conference on Parallel Problem Solving From Nature, Lecture Notes in Computer Science* vol. 5199, Springer, 2008, pp. 559–568.

[Prestwich *et al.* 2008b] S. D. Prestwich, S. A. Tarim, R. Rossi, B. Hnich. A Cultural Algorithm for POMDPs from Stochastic Inventory Control. *5th International Workshop on Hybrid Metaheuristics, Lecture Notes in Computer Science* vol. 5296, Springer, 2008, pp. 16–28.

[Prestwich *et al.* 2009] S. D. Prestwich, S. A. Tarim, R. Rossi, B. Hnich. Neuroevolutionary Inventory Control in Multi-Echelon Systems. *1st International Conference on Algorithmic Decision Theory, Lecture Notes in Artificial Intelligence* vol. 5783, Springer, 2009, pp. 402–413.

[Schaffer *et al.* 1992] J. Schaffer, D. Whitley, L. Eshelman. Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art. *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, pp. 1–37.

[Silver *et al.* 1998] E. A. Silver, D. F. Pyke, R. Peterson. Inventory Management and Production Planning and Scheduling. John-Wiley and Sons, New York, 1998.

[Smalley *et al.* 2000] J. B. Smalley, B. Minsker, D. E. Goldberg. Risk-Based In Situ Bioremediation Design Using a Noisy Genetic Algorithm. *Water Resour. Res.* 36(10):3043–3052, 2000.

[Stagge 1998] P. Stagge. Averaging Efficiently in the Presence of Noise. *5th International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science* vol. 1498, Springer-Verlag, 1998, pp. 188–197.

[Stanley and Miikkulainen 2002] K. O. Stanley, R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation* 10(2):99–127, 2002.

[Stroud 2001] P. D. Stroud. Kalman-Extended Genetic Algorithm for Search in Nonstationary Environments with Noisy Fitness Functions. *IEEE Transac-*

tions on Evolutionary Computation 5(1):66–77, 2001.

[Tarim *et al.* 2006] S. A. Tarim, S. Manandhar, T. Walsh. Stochastic Constraint Programming: A Scenario-Based Approach. *Constraints* 11:53–80, 2006.

[Then and Chong 1994] T. W. Then, E. K. P. Chong. Genetic Algorithms in Noisy Environments. *9th IEEE International Symposium on Intelligent Control*, Columbus, Ohio, USA, 1994, pp. 225–230.

[Van Roy *et al.* 1997] B. Van Roy, D. P. Bertsekas, Y. Lee, J. N. Tsitsiklis. A Neuro-Dynamic Programming Approach to Retailer Inventory Management. *Proceedings of the IEEE Conference on Decision and Control*, 1997.

[Wu *et al.* 2006] J. Wu, C. Zheng, C. C. Chien, L. Zheng. A Comparative Study of Monte Carlo Simple Genetic Algorithm and Noisy Genetic Algorithm for Cost-Effective Sampling Network Design Under Uncertainty. *Advances in Water Resources* 29:899–911, 2006.

[Yao and Lin 1999] X. Yao, Y. Liu, G. Lin. Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation* 3(2):82–102, 1999.

```
train($\mu$,$S$,$U$)
  create ANN population of size $\mu$
  evaluate population using $S$ samples
  while not(timeout)
    select a parent
    breed an offspring O by mutation
    evaluate O using $S$ samples
    if O fitter than locally least-fit chromosome L
      replace L by O
    select globally fittest chromosome F with #samples$<U$
    if F exists
      re-evaluate F using $S$ more samples
  return best chromosome found with #samples$\geq U$
```

Figure 1. Cellular evolution strategy with resampling

CAPTIONS:

Figure 1. Cellular evolution strategy with resampling

Table 1. Experimental results on small problems

Table 2. Experimental results on large problems

		arborescent						serial					
		SP		NEMUE^u		NEMUE^d		SP		NEMUE^u		NEMUE^d	
#	periods	cost	%opt	cost	%opt	cost	%opt	#	%opt	cost	%opt	cost	%opt
1	4	2507	0.0	2573	2.6	2600	3.6	995	0.0	993	0.0	981	0.0
2	5	3124	1.4	3180	3.1	3251	5.3	1269	0.7	1298	2.9	1286	2.0
3	6	3657	2.7	3775	5.7	3846	7.5	1493	1.8	1491	1.7	1604	8.6
4	7	4214	5.6	4250	6.4	4356	8.7	1794	7.4	1797	7.6	1888	12.0
5	8	4654	8.2	4722	9.5	4934	13.4	2087	12.0	1987	7.6	2167	15.2
6	9	5472	16.9	5162	11.9	5443	16.5	2741	25.7	2295	11.3	2460	17.2
7	10	—	?	5590	?	6046	?	—	?	2498	?	2735	?
8	4	2100	0.0	2169	3.2	2182	3.8	1311	0.2	1306	0.0	1327	1.4
9	5	2626	0.6	2722	4.1	2738	4.7	1598	2.2	1594	2.0	1664	6.1
10	6	3311	1.8	3409	4.6	3586	9.3	1833	4.3	1832	4.2	1935	9.3
11	7	4065	2.5	4153	4.6	4714	15.9	2024	6.7	2024	6.7	2140	11.8
12	8	4454	3.4	4542	5.3	5863	26.6	2160	9.3	2142	8.5	2285	14.3
13	9	5158	10.3	5115	9.5	6144	24.7	2678	25.1	2264	11.4	2414	16.9
14	10	—	?	5432	?	6756	?	—	?	2407	?	2596	?
15	4	1342	0.2	1340	0.1	1350	0.8	1104	0.0	1104	0.0	1105	0.1
16	5	1657	1.8	1671	2.6	1673	2.7	1417	2.1	1423	2.5	1446	4.1
17	6	1930	2.2	1938	2.6	1994	5.3	1759	4.1	1763	4.3	1790	5.8
18	7	2180	4.5	2192	5.0	2289	9.0	2057	5.4	2055	5.3	2143	9.2
19	8	2428	6.1	2393	4.7	2480	8.1	2266	6.6	2258	6.3	2363	10.4
20	9	2853	13.9	2617	6.1	2778	11.6	2706	17.7	2479	10.2	2671	16.6
21	10	—	?	2851	?	3108	?	—	?	2627	?	2871	?
22	4	1086	0.0	1096	0.9	1128	3.7	828	0.0	828	0.0	830	0.2
23	5	1334	0.2	1330	0.0	1392	4.3	931	0.0	934	0.3	944	1.4
24	6	1680	0.6	1677	0.4	1886	11.5	1259	1.3	1265	1.8	1423	12.7
25	7	2055	0.7	2051	0.5	2326	12.3	1633	2.4	1639	2.8	1817	12.3
26	8	2219	1.1	2219	1.1	2639	16.9	1757	2.7	1766	3.2	1971	13.3
27	9	2479	2.0	2531	4.0	3127	22.3	1983	3.9	2000	4.7	2340	18.6
28	10	—	?	2665	?	3223	?	—	?	2150	?	2464	?

Table 1. Experimental results on small problems

		arborescent				serial			
		NEMUE^u		NEMUE^d		NEMUE^u		NEMUE^d	
periods	time	cost	%opt	cost	%opt	cost	%opt	cost	%opt
20	3	24521	34.2	20530	12.4	6798	5.6	6680	3.8
20	10	22125	21.1	20432	11.8	6530	1.5	6680	3.8
20	30	21550	18.0	20409	11.7	6502	1.0	6680	3.8
20	100	21274	16.4	20394	11.6	6466	0.5	6680	3.8
20	300	20911	14.5	20365	11.5	6461	0.4	6680	3.8
40	3	79882	118.6	41090	12.5	17471	35.7	13366	3.9
40	10	65979	80.6	40973	12.1	13814	7.3	13343	3.7
40	30	47395	29.7	40922	12.0	13131	2.0	13343	3.7
40	100	44871	22.8	40882	11.9	13032	1.3	13343	3.7
40	300	43150	18.1	40842	11.8	12959	0.7	13343	3.7
60	3	122742	123.9	61578	12.3	28582	48.1	20182	4.5
60	10	117888	115.1	61365	12.0	27658	43.3	20065	3.9
60	30	84147	53.5	61357	11.9	22399	16.0	20065	3.9
60	100	67718	23.6	61277	11.8	19864	2.9	20065	3.9
60	300	65029	18.6	61277	11.8	19614		20052	3.9
80	3	163555	123.8	82935	13.5	38172	48.3	26864	4.4
80	10	151493	107.3	82234	12.5	37216	44.6	26840	4.3
80	30	139543	90.9	82114	12.4	33334	29.5	26834	4.3
80	100	122334	67.4	81830	12.0	26535	3.1	26779	4.0
80	300	117580	60.9	81713	11.8	26311	2.2	26750	3.9
100	3	182033	99.3	106664	16.8	48546	50.9	34022	5.7
100	10	177525	94.3	104586	14.5	47691	48.2	33673	4.7
100	30	173158	89.6	103943	13.8	46468	44.4	33507	4.1
100	100	169331	85.4	102876	12.6	37037	15.1	33486	4.1
100	300	162582	78.0	102645	12.4	33376	3.7	33459	4.0

Table 2. Experimental results on large problems