EVOLUTIONARY OPTIMIZATION OF A FLOW LINE USED
ExtendSim BUILT-IN OPTIMIZER

Baiba Zvirgzdiņa, Jurijs Tolujevs

Transport and Telecommunication Institute
Lomonosova str. 1, LV-1019, Riga, Latvia
E-mail: baiba.zvirgzdina@inbox.lv

Keywords: flow line simulation, ExtendSim, evolutionary optimization

Introduction

ExtendSim is a simulation program for modelling discrete event, continuous, agent-based, and discrete rate processes [1]. There are four ExtendSim packages: CP for continuous processes; OR (operations research), which adds discrete event; AT (advanced technology), which adds discrete rate, a number of advanced modelling features, and Stat::Fit for statistical distribution fitting; and Suite, which adds 3D animation. ExtendSim is used for modelling manufacturing, logistics, supply chain, healthcare, communications, defence, environmental, agricultural, biological, energy, reliability, service, information flow, and recreational systems.

An open-source evolutionary optimizer is included in all versions of ExtendSim. ExtendSim facilitates optimization by making the optimization algorithm available within a block that can be added to any model to control all aspects of the optimization. Furthermore, having a block do the optimization increases flexibility and opens up the method and source code to users who might want to modify or create their own customized optimization blocks. Optimization, sometimes known as “goal seeking”, is a useful technique to automatically find the best answer to a problem. Like most optimization algorithms, the ExtendSim Optimizer solves models using an initial population of possible solutions. Each solution is explored by running the model several times using different values for some selected parameters, averaging the samples (for stochastic, random models), and sorting the solutions. The best solution sets of parameters are then used to derive slightly different but possibly better solutions. Each new derived solution is called a generation. This process continues for enough generations until the Optimizer determines that there are probably no better solutions in sight. The Optimizer then terminates the simulation runs and populates the model with the best solutions it has found.

Although the code applied in ExtendSim evolutionary optimizer is open, the principles of the internal organization of the optimizer is not described in the customer documentation of package ExtendSim and are offered to work with him as with a „black box“, for the settings that can use only two parameters: a) the maximum number of runs of the stochastic model with one combination of varying variables and b) the boundary value of the population convergence indicator. About ExtendSim optimizer block is only known that it uses an evolutionary algorithm, similar to genetic algorithms, but because it uses floating point arithmetic, it’s better suited to stochastic processes such as simulation. There are also indications that this algorithm is the most similar to the type of algorithm evolution strategies [2].

In writing this paper the authors had two aims: a) to summarize the theoretical basis of the algorithms types evolution strategies, and b) to study experimentally the ability to influence the work of the ExtendSim optimizer when used in conjunction with the „classic“ production line simulation model [3].

Evolution Strategies

In [2] is given a comprehensive introduction into one of the main branches of evolutionary computation – the evolution strategies (ES) the history of which dates back to the1960s in Germany. An evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based meta heuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem
play the role of individuals in a population, and the fitness function determines the quality of the solutions. Evolution of the population then takes place after the repeated application of the above operators. EA often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape.

An ES is an optimization technique based on ideas of adaptation and evolution. It belongs to the general class of evolutionary computation methodologies. ES use natural problem-dependent representations, and primarily mutation and selection, as search operators. In common with EA, the operators are applied in a loop. An iteration of the loop is called a generation. The sequence of generations is continued until a termination criterion is met.

Three of the nearly contemporaneous sources of the EA have been kept alive over three decades and experienced an amazing increase of interest during the last fifteen years. Two of them are lying in the United States of America, the source of evolutionary programming (EP) [4], the source of genetic algorithms (GA) [5]. ES is the third main variant of EA, were founded by students at the Technical University of Berlin [6].

There were two rules for designing and evaluating the consecutive experiments ES:
1. Change all variables at a time, mostly slightly and at random.
2. If the new set of variables does not diminish the goodness of the device, keep it, otherwise return to the old status.

Rule one seemed to resemble mutations in nature, and rule two modelled the “survival of the fittest”.

The performance of the ES largely depends on the adjustment of the internal parameters, prominently the mutation strength(s). Schwefel introduced two multi membered ES, i.e.,

- the $\mu + \lambda$ - ES, in which not only one offspring is created at a time or in a generation, but $\lambda \geq 1$ descendants, and, to keep the population size constant, the $\lambda$ worst out of all $\mu + \lambda$ individuals are discarded;
- the $\mu \lambda$ - ES, in which the selection takes place among the $\lambda$ offspring only, whereas their parents are “forgotten” no matter how good or bad their fitness was compared to that of the new generation. Obviously, this strategy relies on a birth surplus, i.e., $\lambda > \mu$.

As far as real-valued search spaces are concerned, mutation is normally performed by adding a normally distributed random value to each vector component. The step size or mutation strength (i.e. the standard deviation of the normal distribution) is often governed by self-adaptation. Individual step sizes for each coordinate or correlations between coordinates are either governed by self-adaptation or by covariance matrix adaptation.

The usual goal of an ES is to optimize (some) given objective or quality function(s) $F$ with respect to a set of decision variables or control parameters $\mathbf{y}$ are often referred to as object parameters

$F(\mathbf{y}) \rightarrow \text{opt.}, \mathbf{y} \in Y.$

$Y$ can be any set of data structures of finite but not necessarily fixed length. Examples for $Y$ are the real-valued $N$-dimensional search space $\mathbb{R}^N$, the integer search space $\mathbb{Z}^N$, the binary search space $\mathbb{B}^N$, the space of permutations $\mathbb{P}_N$ as well as mixtures of different spaces and subspaces (due to constraints).

ES operate on populations $\beta$ of individuals $a$. An individual $a_k$ with index $k$ comprises not only the specific object parameter set (or vector) $\mathbf{y}_k$ and its objective function value $F_k \equiv F(\mathbf{y}_k)$, sometimes referred to as fitness, but usually also a set of strategy parameters $\theta_k$:

$a_k = (\mathbf{y}_k, \theta_k, F(\mathbf{y}_k)).$

The strategy parameters are a peculiarity of ES: they are used to control certain statistical properties of the genetic operators, especially those of the mutation operator. Strategy parameters can evolve during the evolution process and are needed in self-adaptive ES. Within one ES generation step, $\lambda$ offspring individuals $\Omega_\lambda$ are generated from the set of $\mu$ parent individuals $\Omega_\mu$. That is, the size $\lambda$ of the offspring population $\beta_\lambda$ is usually not equal to the size $\mu$ of the parent population $\beta_\mu$. The strategy-specific parameters $\mu$ and $\lambda$ as well as $\theta$ (the mixing number) are called “exogenous strategy parameters”
which are kept constant during the evolution run. The $\rho$ refers to the number of parents involved in the procreation of one offspring. For $\rho = 1$ (cloning), we have the special ES cases without recombination, usually denoted by $(\mu, 1)$ and $(\mu + 1, \lambda)$, respectively. All other cases $(\rho > 1)$ are strategies with recombination, called $\rho$-ES. This makes them less prone to get stuck in local optima. The “+” refers to the kind of selection used, i.e., “+” and “,”-selection, respectively.

**Selection**

Each EA needs a goal oriented selection operator in order to guide the search into promising regions of the object parameter space. It gives the evolution a direction. In ES selection only those individuals with promising properties, e.g., high fitness values (objective function values), get a chance of reproduction. That is, a new parental population at $g$ is obtained by a deterministic process guaranteeing that only the $\mu$ best individuals $a$ from the selection pool of generation $(g)$ are transferred into $\beta^g_{\mu + 2\lambda}$.

$$\beta^g_{\mu + 2\lambda} = \{a_{1, \mu}, \ldots, a_{\mu, \mu}\}.$$  

There are two versions of this selection technique, depending on whether or not the parental population at $(g)$ is included in this process, i.e., plus selection, denoted by $(\mu + 1, \lambda)$, and comma selection, denoted by $(\mu, \lambda)$, respectively. In the case of $(\mu, \lambda)$ selection, only the $\lambda$ newly generated offspring individuals, i.e. the $\beta^g_{\mu + \lambda}$ population, define the selection pool. In other words, the parents from generation $(g)$ are forgotten even when they are better than all offspring. The $(\mu + 1, \lambda)$ notation indicates that both the parents and the offspring are copied into the selection pool which is therefore of size $\mu + \lambda$. Plus selection guarantees the survival of the best individual found so far. Since it preserves the best individual such selection techniques are called elitist. Due to the elitism in plus strategies, parents can survive an infinitely long time-span. Both selection variants have their specific application areas. While the $(\mu + 1, \lambda)$ selection is recommended for unbounded search spaces $Y$, especially $Y = \mathbb{R}^n$, the $(\mu, \lambda)$ selection should be used in discrete finite size search spaces, e.g., in combinatorial optimization problems.

**Mutation**

There is not an established design methodology up to now, but some rules have been proposed by analyzing successful ES implementations and theoretical considerations.

- **Reachability.** Given a parental state $(\beta^g_{\mu + 2\lambda})$, the first requirement ensures that any other (finite) state $(\beta^g, \beta^g)$ can be reached within a finite number of mutation steps or generations. This is also a necessary condition for proving global convergence.
- **Unbiasedness.** Selection exploits the fitness information in order to guide the search into promising search space regions, whereas variation explores the search space, i.e., it should not use any fitness information but the search space information from the parental population. Therefore, there is no preference of any of the selected individuals (parents) in ES.
- **Scalability.** The scalability requirement states that the mutation strength or the average length of a mutation step should be tunable in order to adapt to the properties of the fitness landscape. The goal of adaptation is to ensure the ‘evolvability’ of the “ES system”, i.e., of the ES algorithm in conjunction with the objective function. Term “evolvability” expresses the idea that the variations should be generated in such a way that improvement steps are likely, thus building a “smooth” evolutionary random path through the fitness landscape toward the optimum solution. The smoothness assumption is sometimes expressed in terms of the strong concept stating that small changes on the genetic level should result on average in small changes in the fitness values.

**Recombination**

While mutation performs search steps based on the information of only one parent and on its endogenous strategy parameters, recombination shares the information from up to $\rho$ parent individuals.
Unlike standard crossover in GA where two parents produce two offspring, the application of the Standard ES recombination operator to a parent family of size $P$ produces only one offspring.

There are two standard classes of recombination used in ES: “discrete recombination” sometimes referred to as “dominant recombination” and the “intermediate recombination”. Given a parental vector $\mathbf{a} = \{a_1, \ldots, a_P\}$ (object or strategy parameter vector), the dominant recombination produces a recombinant $\mathbf{r} = \{r_1, \ldots, r_P\}$ by coordinate-wise random selection from the corresponding coordinate values of the parent family $\mathbf{m}_k = \{m_{1k}, \ldots, m_{Pk}\}$. In other words, the $k$th component of the recombinant is determined exclusively by the $k$th component of the randomly (uniformly) chosen parent individual $m_k$ that can also be interpreted as “dominance”.

In contrast to discrete (dominant) recombination the intermediate recombination takes all $P$ parents equally into account. It simply calculates the centre of mass (centroid) of the $P$ parent vectors $\mathbf{a}_m$.

\[ \mathbf{v}_k = \frac{1}{P} \sum_{m=1}^{P} \mathbf{a}_m \mathbf{v}_k. \]

While this procedure is well defined for real-valued state spaces, the application in discrete spaces needs an additional procedure such as rounding or probabilistic rounding in order to map back onto the discrete domain.

**Three-Stage Buffer Allocation Problem**

We propose an optimization-via-simulation algorithm for use when the performance measure is estimated via a stochastic, discrete-event simulation, and the decision variables may be subject to deterministic linear integer constraints. We consider a three-stage flow line with finite buffer storage space in front of stations 2 and 3 and an infinite number of jobs in front of station 1. There is a single server at each station, and the service time at station $h$ is exponentially distributed with rate $\mu_h$, $h = 1, 2, 3$ (see Fig. 1). If the buffer of station $h$ is full, then station $h - 1$ is blocked and a finished job cannot be released from station $h - 1$. The total buffer space and the service rates are limited. The goal is to find a buffer allocation and service rates such that the throughput (average output of the flow line per unit time) is maximized.

Let $b_h$ be the number of buffer space at station $h$, $h = 2, 3$. The constraints are as follows:

\[
\begin{align*}
\mu_1 + \mu_2 + \mu_3 & \leq 20 \\
b_2 + b_3 & \leq 20 \\
- b_2 - b_3 & \leq -20 \\
1 & \leq \mu_h \leq 20, h = 1, 2, 3 \\
1 & \leq b_h \leq 20, k = 2, 3 \\
\mu_h, b_h & \in \mathbb{Z}. 
\end{align*}
\]

Such a three-stage flow line model has been selected in order to ensure an opportunity to compare the results of optimization with those obtained by other authors. In [3] is described exactly the same model of a three-stage flow line. In [7] and [8] optimizations of analogues flow lines models are viewed.

**Realization of the Model and Optimizer Configuration in the ExtendSim Environment**

A simulation model realized by using ExtendSim package is shown on Figure 2. All five decision variables are set using blocks of type *constant*. Blocks of the type *equation* are used to calculate the mean
time values of processing time at workstations. In the first version of the model that fully corresponds [3], each value $\mu_h$ is interpreted as intensiveness of processing (rate), but mean time value is calculated by the formula:

$$\tau_h = \frac{1}{\mu_h}$$

This version of the model has disadvantage that optimization problem is obtained with not clearly expressed (strict) extreme, as due to the homogeneity of workstations set of values $(\mu_1, \mu_2, \mu_3) = (7, 7, 6)$ can be as optimal as $(\mu_1, \mu_2, \mu_3) = (6, 7, 7)$. For this reason, the second variant of the model was explored in which each workstation has its own average processing time on a every single machine $(t_1, t_2, t_3) = (0.333; 0.5; 0.2)$, and the value of mean time for the whole station is calculated by the formula

$$\tau_h = t_h / \mu_h,$$

where the parameter $\mu_h$ is interpreted as the number of parallel working machines [7].

Table 1 shows the values $\tau_h$ that correspond to optimization values $\mu_h$.

The following steps are needed to optimize the simulation model of a three-stage flow line [9]:
1) Add an Optimizer block (Value library) to a model.
2) Define the form of the objective function.
3) Determine which variables the equation needs and “clone-drop” them onto the Optimizer.
4) Set the limits for those variables in the Optimizer’s Variables table.
5) Derive the equations for the objective function.
6) If variables need to be constrained to certain values, add constraint equations.
7) Set the Optimizer’s Run Parameters for a random model, and then run the optimization.

Table 1. The values of mean time, calculated in the blocks equation

<table>
<thead>
<tr>
<th>Model Variant 1 [3]</th>
<th>rate $\mu_h$ [1/h]</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean time $\tau_h$ [h]</td>
<td>1</td>
<td>0.5</td>
<td>0.333</td>
<td>0.25</td>
<td>0.2</td>
<td>0.167</td>
<td>0.143</td>
<td>0.125</td>
<td>0.111</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Variant 2 [7]</th>
<th>number of machines</th>
<th>$\mu_h$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean time $\tau_1$ [h]</td>
<td>1</td>
<td>0.333</td>
<td>0.166</td>
<td>0.111</td>
<td>0.083</td>
<td>0.067</td>
<td>0.055</td>
<td>0.048</td>
<td>0.042</td>
<td>0.037</td>
<td>0.033</td>
<td></td>
</tr>
<tr>
<td>mean time $\tau_2$ [h]</td>
<td>0.5</td>
<td>0.25</td>
<td>0.167</td>
<td>0.125</td>
<td>0.1</td>
<td>0.083</td>
<td>0.071</td>
<td>0.063</td>
<td>0.056</td>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean time $\tau_3$ [h]</td>
<td>0.2</td>
<td>0.2</td>
<td>0.067</td>
<td>0.05</td>
<td>0.04</td>
<td>0.033</td>
<td>0.029</td>
<td>0.025</td>
<td>0.022</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. The three-stage flow line as an ExtendSim simulation model

Figure 3 shows the results of the Steps 1-5. For all decision values are defined boundary values: Minimum Unit = 1 and Maximum Unit = 20. Variable Var6 indicates the value of “average output of the flow line per unit time”, that is read by the connector TP of the information block. Expression MaxProfit = Var6 transforms variable Var6 into objective function, that is subject to maximization.
The results of the Step 6 are shown on Figure 4. In the global constraints form are written inequalities that are shown above in the mathematical formulation of the optimization problem.

Following Optimizer’s Run Parameters must be given in the Step 7:

- **Maximum Samples per Case**: This is the maximum number of runs averaged to get a member result. The Optimizer block starts the number of samples at 1 and increases them with each generation until it reaches the maximum. Sometimes it is useful to reduce the maximum number of samples (possibly to 5), to get a rough idea of the best solution without spending too much time running samples.

- **Member Population Size**: How many members are in the evolutionary population.

- **Check Convergence after “n” cases**: How many cases to wait before checking for convergence. This prevents premature false convergence by insuring that a minimum number of cases (generations) are run.

- **Terminate optimization after “n” cases**: Sets a maximum on the number of cases (generations) that the optimizer will run.

- **Terminate if best and worst within**: If this option is selected, you can specify how close to optimum you want the results to approach. This value is used to examine the current population of results to see if they are within a specified percentage of each other.

Concrete values of these parameters under which the optimization model was performed are shown in Table 2.

### Experiments to Model Optimization

ExtendSim block Optimizer offers two standard set of values for the Run Parameters: Quicker Defaults and Better Defaults. Variant Quicker Defaults is used in experiments No. 1, 2, 3 and 4, but variant Better Defaults – in experiments No. 6, 5 and 6 (see Table 2). Experiment No. 0 was performed with Model Variant 1 only to verify the model. The optimization results are almost identical to results shown in [3]. Experiments No. 1, 2, 3, 4, 5 and 6 were performed with Model Variant 2.
The number of feasible solution is 21660. The optimal solution for the Model Variant 2 is \((6, 10, 4, 11, 9)\) with an expected throughput of 17,44 jobs per unit time. As it is seen from Table 2, this solution was obtained only once in Experiment No. 6. Fundamentally important relationship \(\mu_2 > \mu_1 > \mu_3\) is fulfilled in all experiments. This indicates a tendency to make throughput for all three stations approximately the same for a given condition \(t_2 > t_1 > t_3\). As Optimization Value on Figure 5 the output variables of the model MaxProfit (throughput) and Convergence are shown.

<table>
<thead>
<tr>
<th>Experiment No.</th>
<th>Maximum Samples per Case</th>
<th>Member Population Size</th>
<th>Check/Convergence after n cases</th>
<th>Terminate after optimization after n cases</th>
<th>MaxProfit (100)</th>
<th>Convergence [%]</th>
<th>Total cases (generations)</th>
<th>Total samples (simulation runs)</th>
<th>Values of decision variables (\mu_1, \mu_2, \mu_3, b_2, b_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>0.95</td>
<td>17,18</td>
<td>95,12</td>
<td>62</td>
<td>360</td>
<td>7, 9, 4, 7, 13</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>10</td>
<td>50</td>
<td>0.95</td>
<td>15,08</td>
<td>98,28</td>
<td>51</td>
<td>530</td>
<td>6, 8, 5, 15</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>10</td>
<td>50</td>
<td>0.99</td>
<td>17,14</td>
<td>99,00</td>
<td>255</td>
<td>1570</td>
<td>6, 9, 5, 10, 10</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>10</td>
<td>100</td>
<td>0.95</td>
<td>16,56</td>
<td>95,02</td>
<td>105</td>
<td>603</td>
<td>7, 9, 4, 3, 17</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>10</td>
<td>100</td>
<td>0.95</td>
<td>17,26</td>
<td>99,25</td>
<td>193</td>
<td>5056</td>
<td>6, 10, 4, 14, 6</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>20</td>
<td>100</td>
<td>0.95</td>
<td>17,44</td>
<td>98,45</td>
<td>201</td>
<td>2104</td>
<td>6, 10, 4, 11, 9</td>
</tr>
</tbody>
</table>

Each optimization procedure starts with decision variables values \((\mu_1, \mu_2, \mu_3, b_2, b_3) = (6, 6, 6, 10, 10)\). The number of feasible solution is 21660. The optimal solution for the Model Variant 2 is \((\mu_1, \mu_2, \mu_3, b_2, b_3) = (6, 10, 4, 11, 9)\) with an expected throughput of 17,44 jobs per unit time. As it is seen from Table 2, this solution was obtained only once in Experiment No. 6. Fundamentally important relationship \(\mu_2 > \mu_1 > \mu_3\) is fulfilled in all experiments. This indicates a tendency to make throughput for all three stations approximately the same for a given condition \(t_2 > t_1 > t_3\). As Optimization Value on Figure 5 the output variables of the model MaxProfit (throughput) and Convergence are shown.

![Figure 5. Optimization Value graphical charts for experiments with Model Variant 2](image-url)
Conclusions

The evolutionary algorithm used in the optimizer ExtendSim, showed satisfactory results in the optimization model of a three-stage flow line that belongs to a class stochastic, discrete-event simulation. Peculiarity of the optimization problem lay in the fact that all five decision variables were nonnegative integers and for them were formulated linear constraints.

In cases where the Optimizer’s Run Parameters received values according to Quicker Defaults strategy the algorithm led only to suboptimal solutions. Also according to strategy Better Defaults optimal solution could be obtained only after the increase of the parameter Member Population Size from 10 to 20. The optimal solution was not obtained even in Experiment No. 5, when the amount of Total samples (simulation runs) has reached the value 5056.

A wide variety of graphical charts, shown on Figure 5, is the result of high stochasticity level of both the model of a three-stage flow line as well as the evolutionary optimization algorithm. A rule, which recommends repeating the optimization procedure also for different initial values of the decision variables, remains in force.

References