Java Call Centers Optimization Library

Installation and User's Guide

Package : umontreal.iro.lecuyer.ccoptim

Laboratoire de Simulation
Chaire de Recherche du Canada en Simulation et Optimisation Stochastiques
DIRO, Université de Montréal

November 20, 2006

Contents

1 Introduction

This document is a short user's guide for a staffing optimization program for multiskill call centers. It includes two differents optimization programs : a cutting plane algorithm based on simulation and linear programming [4] and a randomized search using approximations [1], both developed in the simulation laboratory of DIRO, under the Canada Research Chair in Stochastic Simulation and Optimization.

In a staffing problem, a multiskill call center is composed by multiple agent groups and different call types. A call requires the appropriate skill in order to be served. Agents of the same groups share an identical skills set and the cost an agent depends on his group (skills set).

The objective is to minimize the staffing cost while satisfying the service level (SL) targets. The service levels taken into account in our problem are defined as the proportion of calls having waited less than awt time, where awt is called the acceptable waiting time. Many variants of the service level formula exist, the user must assure than the desired formula is available in the call center evaluator library to be given to the optimization programs.

To formulate as a mathematical problem, we have : a set $\mathcal{M} = \{1, 2, \ldots, m\}$ of agent groups and a set $\mathcal{N} = \{1, 2, \ldots, n\}$ of call types, the cost vector $\mathbf{c} = (c_1, \ldots, c_m)$ and staffing vector $\mathbf{x} = (x_1, \ldots, x_m)$. The service levels are complex functions, denoted as $g(\mathbf{x})$ or $g_k(\mathbf{x})$ for call type k, that are approximated by simulations or analytically. The solution is constrained to satisfy the service level targets l for the global target and per call type target l_k for each $k \in \mathcal{N}$. We obtain the integer problem :

$$
\begin{aligned}
\min \quad & \mathbf{c}^T\mathbf{x} = \sum_{i=1}^{m} c_i x_i \\
\text{subject to :} \quad & \\
& g(\mathbf{x}) \geq l \\
& g_k(\mathbf{x}) \geq l_k, \qquad \forall k \in \mathcal{N} \\
& \mathbf{x} \geq 0 \text{ and integer}
\end{aligned}
$$

In the current version, the programs are developed on Sun Microsystems' Java 5. The programs use the ContactCenters : Java library for simulating contact centers [3], developed in the same laboratory by Eric Buist, the simulation library SSJ [7], the scientific library colt [5] and also implementation to support libraries for linear optimization li-

braries OR-Objects 1.2.4[8] and Cplex [6]. The linear optimization library OR-Objects 1.2.4[8] is freely available.

This guide contains informations for the installation under the Windows and Linux operating system and for the usage of the staffing optimization programs. It explains what needs to be installed, how to run the program, the description of each input parameter file.

It is recommended to consult the detailed guides describing the simulation programs and the XML parameters, written by Eric Buist. The details of the optimization algorithm are described in Ceşik and L'Ecuyer [4] and Avramidis et al. [2]. It explains the choice and reason of the values given to some parameters of the algorithm, principally for the cut generation.

2 Installation

The current version of the optimization programs are developed on Sun Microsystems' Java 5. A version of Java 5 is required for the execution of the programs. If one doesn't need to compile the programs, the Java Runtime Environment (JRE) is sufficient, otherwise installs the Jave Development Kit (JDK). For linear optimization library, the user must assure the proper installation of their library such as Cplex [6]. Note that for OR-Objects 1.2.4 no installation is required.

A copy of the ContactCenters, SSJ and colt are included in the distribution, and no executable installation is required.

3 Program Setup

The following Java archives (`jar`), included in the distribution, are required for the optimization programs to run: `ccoptim.jar`, `ssj.jar`, `contactcenters.jar`, `colt.jar`. The `jar` need to be declared in the system environment variable `CLASSPATH`.

Additionnal steps would be required to setup the linear optimization library. If using OR-Objects 1.2.4, `or124.jar` must also be included in the `CLASSPATH`. If using Cplex, `cplex.jar` must also be included in the `CLASSPATH`. We suggest the user to consult their user's manual for an appropriate setup of their libraries.

Note: To use the pre-implemented solver interfaces with Cplex 8, Cplex 9 or OR-Objects 1.2.4, the folder `lib` must be added to the `CLASSPATH`.

3.1 Windows 95/98/Me

Suppose, for example, that the folder `lib` has been copied to the directory `C:\ccoptim\lib`. Then the next step is to set the `CLASSPATH` variable. Open a text console and type *in a single line*:

```
set CLASSPATH=%CLASSPATH%;C:\ccoptim\lib\ccoptim.jar;
C:\ccoptim\lib\ssj.jar;C:\ccoptim\lib\colt.jar;
C:\ccoptim\lib\contactcenters.jar;C:\ccoptim\lib
```

It would be preferable to insert the line in the file `C:\autoexec.bat` if `C:` is the system drive.

3.2 Windows 2000/XP

Suppose, for example, that the folder `lib` has been copied to the directory `C:\ccoptim\lib`. Then the next step is to set the `CLASSPATH` variable.

1. Right-click on **My Computer** and select **properties**.
2. Select the tab **Advanced** and click on the button named **Environment Variables**.
3. If a variable named `CLASSPATH` already exists, select it and click on **Edit**, otherwise, click on **New** and type `CLASSPATH` as the **Variable name**.
4. In **Variable value**, add the location of each `jar` separete with a colon ";". Such as, `C:\ccoptim\lib\ccoptim.jar;C:\ccoptim\lib\ssj.jar; C:\ccoptim\lib\colt.jar; C:\ccoptim\lib\contactcenters.jar; C:\ccoptim\lib`

3.3 Linux/Unix

Suppose, for example, that the folder `lib` has been copied to the directory `/home/username/ccoptim/lib`.

The user need to define the `CLASSPATH` environment variable if not already defined and :

- Using C-shell, add to file `.cshrc` in one line :

```
setenv CLASSPATH /home/username/ccoptim/lib/ccoptim.jar:
/home/username/ccoptim/lib/contactcenters.jar:
/home/username/ccoptim/lib/colt.jar:
```

```
/home/username/ccoptim/lib/ssj.jar:
/home/username/ccoptim/lib:$CLASSPATH
```

- Using Bourne shell, add to file `.bashrc` in one line :

```
export CLASSPATH=${CLASSPATH}:
/home/username/ccoptim/lib/contactcenters.jar:
/home/username/ccoptim/lib/colt.jar:
/home/username/ccoptim/lib/ssj.jar:
/home/username/ccoptim/lib/ccoptim.jar:
/home/username/ccoptim/lib
```

4 Running the Optimization Programs

Since the `CLASSPATH` environment variable has been set, the programs can be executed in any directory.

4.1 Cutting Plane Optimizer

The cutting plane optimization program needs 3 parameter files which will be detailed along with examples in Section 5. It is important to setup the parameters accordingly to obtain the best performance.

To open a text console in Windows, click on **Start**, **Run** and type "`cmd`".

Command line to run the program :

```
java umontreal.iro.lecuyer.ccoptim.cp.CuttingPlaneOptimizer ccParams.xml
simParams.xml cpParams.xml
```

where,

- *ccParams.xml* is the call center's parameter file. It contains the descriptions of the call center which includes the arrival rates, service rates, routing scheme, global target service level and more.
- *simParams.xml* is the simulator's parameter file. This file is used only by the simulator.
- *cpParams.xml* is the cutting plane optimization parameter file. The user can change the cut generation rules, the type of cuts used and others options related to the optimization. See Section 5.3.

4.2 Randomized Search

The randomized search requires 4 parameter files which will be detailed along with examples in Section 5. It is important to setup the parameters accordingly to obtain the best performance.

To open a text console in Windows, goto **Start**, **Run** and type "`cmd`".

Command line to run the program :

```
java umontreal.iro.lecuyer.ccoptim.rs.RandomizedSearch ccParams.xml
ldParams.xml simParams.xml rsParams.xml
```

where,

- *ccParams.xml* is the call center's parameter file. It contains the descriptions of the call center which includes the arrival rates, service rates, routing scheme, global target service level and more.
- *ldParams.xml* is the loss-delay approximation parameter file.
- *simParams.xml* is the simulator's parameter file. This file is used only by the simulator.
- *rsParams.xml* is the randomized search parameter file. The user can set different multistarts position, a time limit to the execution and others options related to the optimization. See Section 5.4.

5 Parameter Files

For XML files, the parameter tags are all described in their respective Java class parser. Please look in the HTML API or PDF to find the complete list of parameters. A fully detailed description on how to use the XML configuration files is available the guides for the simulation and XML parameters. Some parameters are optional, if they are not specified, the default values will be used, while others are essential parameters like the arrival rates. If an essential parameter is missing, the program will terminate and print an error requiring the missing parameter before starting the algorithm. The type of value accepted by each parameter is the variable type described in the HTML API and PDF. It can be a boolean (true or false), an integer (no decimal) or a real (number with decimals). A parameter can also be an array of those 3 types.

To set a parameter, put the parameter name before and after the value.

Example : `<verbose> true </verbose>`
For an array, the values are separeted by a comma (","). Example : `<arrivalRates> 10, 9, 10 </arrivalRates>`

To put comments in the XML file, surround the comments like this : `<!-- comments -->`. Example to put the `changeHeuristicCut` parameter in comment :
`<!-- <changeHeuristicCut>true</changeHeuristicCut> -->`

5.1 Call Center Parameter File

Both the optimizers and the simulation use the call center parameter file. A fully detailed documentation on the available parameters can be found in the ContactCenters simulation library [3] and XML user's guide. However for the optimizer to work, not every parameter field is required to be filled. The documentation is available in the PDF file : `guidemsk.pdf` of the ContactCenters simulation library.

5.2 Simulator Parameter File

The Java call center simulator also needs a parameter file to configure the simulation. The parameters are described in the simulation and XML parameters user guide in the section `batchSimParams` and common simulation parameters. These informations are read only by the simulator and are never used by the optimizer. It would be important to understand the function of some variables. The quality of the approximation by simulation is dependent of the length of the time simulated. However, long simulation requires more time. The length of the optimization program can be greatly affected by the time used for simulations.

Again, here's a short description for a few parameters, other parameters could be used. For more details, please refer to the ContactCenters simulation user's guide PDF.

- `targetError` (real) : the target relative error to adjust the simulation length dynamically. Set to -1 to disable this. We suggest `-1` to control the simulation length manually.

- `level` (real) : the level of confidence intervals when computing relative errors or displaying statistical reports.

- `initNonEmpty` (boolean) : if `true`, then indicates that the system will be initialized with $n \times$ `targetInitOccupancy` agents busy and all queues empty, where n is the total number of agents. We suggest `true`.

- `batchSize` (real) : batch size in simulation time units.

- `minBatches` (integer) : minimal number of batches. When simulation length adjustment is not used (`targetError` < 0), this corresponds to the number of simulated batches. The total length of the simulation is `batchsize`×(`minbatches`+`warmupBatches`).

- `warmupBatches` (integer) : number of warmup batches to simulate before collecting statistical observations.

Example of the simulator parameter file :

```
<batchSimParams
   targetError="-1"
   level="0.95"
   targetInitOccupancy="0.9"
   initNonEmpty="true"
   batchSize="10"
   minBatches="10"
   warmupBatches="2">
</batchSimParams>
```

Listing 1: Example of the simulator parameter file

5.3 Cutting Plane Optimizer Parameter File

The cutting plane optimizer parameter file is only used by the optimization program and the documentation can be found in the class `CuttingPlaneParams`. Some parameters are very technical and depend on the call center's characteristics, they should be used by users who understand the optimization algorithm. See the paper Ceşik and L'Ecuyer [4] for details. The cost vector $\mathbf{c}$ can be set by defining the parameter `busyCost` for each agent group in the *simulator's parameter file* or by setting the `skillCost` parameter in this file. Important parameters are followed by a "*".

Here's the description of each parameter :

- `currentPeriod` (integer) : select the period to optimize. The period count starts with 0.

- `skillCost` (real) : If the parameter `busyCost` of an agent group in the *simulator's parameter file* is greater than zero (> 0), the cost per agent of that group is equal to `busyCost`. Otherwise, the cost per agent of that group is defined as : $1 +$ `skillCost` $\times$ (skill count $- 1$), where `skillCost` is this parameter.

- `minEachQosH` (real) : for the heuristic cut generation, if the QoS of a call type is below this value, then that call type is subject to generate a heuristic cut, not based on the subgradient. We suggest to set a low value (e.g.`0.1`).

- `testConvexity` (boolean) : if `true`, then the program will test the convexity of the QoS function around the current staffing vector solution by testing one dimension at a time with a length of `defaultSubStep + 1`. This adds more simulations and this test is not needed for the optimization. It is only to check the convexity of the problem. By default, set to `false`.

- `errorOnConvexity` (boolean) : if `testConvexity` is `false`, this option has no effect. If testConvexity is `true` and this is `true`, then whenever the program finds a convexity region, it will throw an error and terminate. If false, then it will only print a convexity warning message and continue the algorithm. By default, set to `false`.

- `cutPriority` (integer) : if equals to 1, then, between a heuristic and a subgradient cut, the priority goes to the subgradient cut. Otherwise, the priority goes to the heuristic cut. Default value is `0` (heuristic cut).

- `defaultSubStep`* (integer) : the default step length used to compute the subgradient around the current staffing vector. Default value is `1`, but should be set according to the call center. It is recommended to set to ≥ 2, if using short simulations and large call centers.

- `lowOverallSubStep`* (integer) : when the global QoS is lower than the `pivotOverallQos` point, the algorithm uses this value to compute the subgradient instead of the `defaultSubStep`. Default value is `2`, it should be higher than `defaultSubStep`.

- `pivotOverallQos` (real) : if the global QoS is lower than this point, then the program will use `lowOverallSubstep` instead of `defaultSubStep`. This parameter is used to reduce the noise of the simulation in convex region. The value should be around halfway between 0 and the global target service level.

- `pivotPerCallQos` (real) : if the global QoS is equal or higher than this value, then the algorithm can choose to generate the subgradient per call type instead of a subgradient for the global QoS when the QoS of call type doesn't satisfy its target QoS. This parameter is justified by the reason that improving a call type also improves the global service level. However, we don't want to start at the beginning since the noise is higher per call type. The value should be close to the global service level target.

- `loadCoefficient`* (array of real) : this is the coefficient that is multiplied to the arrival rate when solving the minimum load problem. Default value is `1.0`. These values should varie with the blocking and abandon rates. Set it sligthly lower when there is abandonment. This parameter is important to determine an initial starting point where the service levels shouldn't be too far in the convex region (very low service levels). If the array has more elements than the number of call types (let's say m) , only the first m elements are used.

- `changeHeuristicCut` (boolean) : if `true`, then the program will follow the improvement resulted from a heuristic cut. If the last heuristic cut added brought a big improvement to the QoS of the call type, it will change that heuristic cut to find the right cut constraint so that the QoS of that call type is at `minEachQosH`. By default, set to `true`.

- `solverClass` (string) : input the name (and package name) of the linear solver interface class (without the suffix `.class`) to be used in the optimization. This class is used by the optimizer to interact with existing linear solver libraries. The currently implemented interfaces included as extra files are :

  - `umontreal.iro.lecuyer.ccoptim.cp.OR124Solver.class` (OR-Objects 1.2.4)
  - `umontreal.iro.lecuyer.ccoptim.cp.Cplex8Solver.class` (Cplex 8)
  - `umontreal.iro.lecuyer.ccoptim.cp.Cplex9Solver.class` (Cplex 9)

  Note: While OR-Object 1.2.4 is free but the implemented solver does not solve IP. The user must install his copy of Cplex in order to use the Cplex solvers.

- `solveIP` (boolean) : depends on `solverClass`. If the solver can solve IP instance, setting to `true` will enable to solve the IP instead of the LP. The setting also depends on the difficulties of the problem. Solving the IP could take a high amount of time in large size problems.

- `solveWithMaxFlowCut` (boolean) : If the service time distribution are independent of the agent groups, the user can choose to add heuristic cuts based on a maxflow subproblem. This parameter affects the heuristic used when the service levels are too poor to generate subgradients. If `false`, a constant number of network flow equations are incorporated at the beginning of the problem. Solving with a maxflow subproblem keeps the number of variables equal to the number of agents, but generating these heuristic cuts can be costly in large problems. When solving the the network flow equations, a limited number of non-negative real variables are added and the constraints are already known. By default, set to `false`.

- `verbose` (boolean) : if `false`, then the program will print only the simulation results, the staffing vector, the objective value and the QoS for the optimal solution. If `true`, then the program will also print a trace of every step of the algorithm, every contraint added, intermediate staffing solutions, quality of service and the objective value. By default, set to `false`.

```
<cuttingPlaneParams
   currentPeriod="0"
   skillCost="0.05"
   minEachQosH="0.1"
   testConvexity="false"
   errorOnConvexity="false"
   cutPriority="0"
   defaultSubStep="2"
   lowOverallSubStep="3"
   pivotOverallQos="0.65"
   pivotPerCallQos="0.70"
   changeHeuristicCut="true"
   solveIP="false"
   solverClass="umontreal.iro.lecuyer.ccoptim.cp.OR124Solver"
   solveWithMaxFlowCut="false"
   verbose="false" >

   <loadCoefficient>
      <row repeat="65">1</row>
   </loadCoefficient>
</cuttingPlaneParams>
```

Listing 2: Example of the cutting plane parameter file

5.4 Randomized Search Parameters

This parameter file is only read by the randomized search program. It contains the parameters that control the heuristic.

- `n1` (integer) : element is the minimum number of skill groups with at least q (randomly generated number) agents to activate the remove algorithm of size q. We suggest to set it to `1`.

- `n2` (integer) : element is the maximum test size desired. Which means that some test candidates may not be evaluated. A test candidate is a skill group that can reduce the staffing cost by changing its staffing. We suggest to set higher or equal to the number of agent groups. It is preferable to evaluate every candidate.

- `delta` (real) : element is the ratio of the number of test candidates over `n2` that is added to the serie of tests. If it is set to `1.0`, then it will test every candidates. This value must be between 0 and 1, inclusively.

- `findInitialStaffing` (boolean) : element, if set to `true`, will generate automatically the initial staffing (with the parameters `rootL` and `lowerCostRatio`). If set to `false`, then the program will use the staffing defined in the Call Center parameter file as the initial solution. We suggest to set to `true`.

- `opGlobalTargetServiceLevel` (boolean) : element is the global target service level used for the approximated values feasibility test if `useLDOTargetServiceLevel` is set to `true`. The global QoS of the optimal solution computed by approximation must be equal or greater than this parameter.
  *Note:*This value is not used with the correction by simulations step. The target service level used to test the feasibility of the simulations are in the Call Center parameter file.

- `opTargetServiceLevel` (array of real) :array of elements that are the target service level per call type used for the approximated values feasibility test. The array contains a target service level for each call type. This is the corresponding parameter of `opGlobalTargetServiceLevel` to the call types. The program will use these values only if `useLDOTargetServiceLevel` is set to `true`. *ATTENTION:*This value is not used with the correction by simulations step. The target service level used to test the feasibility of the simulations are in the Call Center parameter file.

- `useBusyCost` (boolean) : element represents the cost type for the agent groups. If set to `false`, then the agent group cost is computed with the number of skills, which is : 1.0 + (# of skills - 1) * `skillPremiumCost`. If set to `true`, then the program will use the busy cost defined for each agent group in the Call Center parameter file.

- `skillPremiumCost` (real) : element is the premium cost of an agent group for each additional skill over 1.

- `searchMode` (integer) : enables the user to select the type of search : `0` for mix (mix of remove and switch) or `1` for sequential (remove then switch). We suggest to set it to `0`.

- `useSimRemove` (boolean) : if set to `true` will execute the `SimRemove` procedure in the correction by simulation phase. We suggest to set it to `true`.

- `useSimSwitch` (boolean) : if set to `true` will execute the `SimSwitch` procedure in the correction by simulation phase. Because this procedure requires a huge amount of time and the cost saved might not always be worth the extra time, we suggest to set it to `false`. Set to `true` if time is not a constraint.

- `maxRSCPUsec` (int) : is the maximum number of seconds spent in search mode, this does not include finding an initial staffing. Since the program doesn't take hours to finish, this parameter should be omitted. We suggest to set a high value.

- `useRealTime` (boolean) : if set to `true` will compute the execution time as the real (world) time. If set to `false`, the program will compute the execution time as the amount of time spent in the CPU (the CPU time). This value is used to truncate the randomized search if the execution time goes above `maxRSCPUsec`.

- `rootL` (array of doubles) : is used to generate the initial staffing. It uses a root-solver to find the number of agents for each skill group to satisfy the inflows with a quality of service of `rootL`. The AWT is the same as declared in the Call Center parameter file. Setting multiple values triggers a multi-start. We suggest a value around the global service level target.

- `lowestCostRatio` (array of doubles) : represents the proportion of the inflow calls that goes to the cheapest agent group that can serve call type. The rest are divided equally to the other agent groups that can answer to that call type. If there is only one agent group that can serve, then a ratio of 1.0 is automatically used.

This parameter seems to have a bigger impact on the quality of the initial solution. Setting multiple values triggers a multi-start. We suggest a value of 0.8.

- `randomSeeds` (array of integer) : contains the initial seed of the random number generator used by the randomized search program, but not the simulation. This program use the MRG32k3a generator which needs 6 integer numbers. If this parameter is omitted, then it will use the default values.

- `verbose` (boolean) : if set to `true`, prints the status of each iteration : cost, global QoS, the staffing move, number of agents. If set to `false`, then only the final (after simulation) staffing solution, the staffing cost and the QoS are printed.

Here's an example of the parameter file for the optimizer :

```
<randomizedSearchParams
   n1="1"
   n2="10"
   delta="1.0"

   verbose="true"
   findInitialStaffing="true"
   opGlobalTargetServiceLevel="0.8"
   simPriority="1"

   useBusyCost="false"
   skillPremiumCost="0.05"

   searchMode="0"
   useSimRemove="true"
   useSimSwitch="false"
   maxRSCPUsec="300000000"
   useRealTime="false"

   useLDOTargetServiceLevel="false">

   <opTargetServiceLevel>
      0.8, 0.8, 0.8, 0.75, 0.6 , 0.6, 0.6
   </opTargetServiceLevel>

   <rootL> 0.8</rootL>
   <lowestCostRatio> 0.5</lowestCostRatio>
   <randomSeeds> 19992,6210,1414,595,56558,2146 </randomSeeds>
</randomizedSearchParams>
```

Listing 3: Randomized Search parameters

5.5 Loss Delay Approximation Parameters

This file contains all the configuration variables for the Loss Delay approximation program. It approximates the performance levels of a multi-skill call center of a staffing vector for a single period. If the user only wants to approximate the performance of a staffing vector, this vector must be defined in the Call Center parameter file, see Section 5.1.

- `currentPeriod` (integer) : specifies the period to approximate. It is important to select the right period. Period `0` corresponds to the first period of the Call Center parameter file.

- `maxIteration` (integer) : sets the maximum number of iterations to execute while waiting for convergence of the approximation. If the values don't converge, the program returns "non-convergent", which means the convergence failed. This parameter affects directly the speed of the program. We suggest a value of 400.

- `absErrorTolerance` (double) : represents the convergence test value. It is the minimum accepted difference for a performance measure between two iterations. The program iterates until every performance measure (choosen with the parameter `accuracyType`) satisfies this tolerance. A big value (ie: 0.01) would diminish the quality of the approximation. We suggest a value of `1e-4` (or 10^{-4}).

- `accuracyType` (integer) : represents the type of convergence to test. If set to `1`, then the convergence is tested on the blocking probability of each skill groups. Else if set to `2`, the convergence is tested on the inflows of each skill groups. We suggest to set it to `1`.

- `queueAbandon` (boolean) : should be set according to the possibility of abandonment. Set to `false` if there are no abandonment. This parameter decides on the CTMC to be used to estimate the blocking and queueing probabilities.

- `queueCapacity` (integer) : set the queuing length at each agent group. This parameter is only needed when `queueAbandon` is `true`. Setting to < 0 for automatic selection rule which is : $\delta\sqrt{m}$, where m is the number of agents in the group and δ is the defined by the parameter `delta`. We suggest to set to `-1`.

- `delta` (double) : is used compute the queue length when `queueCapacity` is `true`, otherwise it is ignored. See the parameter `queueCapacity`. We suggest a value of 2.

- `debug` (boolean) : if set to `true`, prints out the unstable skill group (when the queues are unstable). We suggest to set it to `false`.

- `verbose` (boolean) : if set to `true`, prints out at each iteration. Set if to `false`, the program doesn't print while in the randomized search.

Here's the parameter file for the Loss Delay approximation :

```
<lossDelayParams
   currentPeriod="0"
   maxIteration="400"
   absErrorTolerance="1e-4"
   accuracyType="1"
   queueAbandon="true"
   queueCapacity="-1"
   delta="2"
   debug="false"
   verbose="false"/>
```

Listing 4: Loss Delay approximation parameter file

References

[1] A. N. Avramidis, W. Chan, and P. L'Ecuyer. Staffing multi-skill call centers using a performance approximation and search methods. Manuscript, 2006.

[2] A. N. Avramidis, W. Chan, and P. L'Ecuyer. Staffing multi-skill call centers via search methods and a performance approximation. Manuscript, 2006.

[3] E. Buist and P. L'Ecuyer. *ContactCenters: A Java Library for Simulating Contact Centers*, 2005. Software user's guide, forthcoming.

[4] M. T. Cezik and P. L'Ecuyer. Staffing multiskill call centers via linear programming and simulation. *Management Science*, 2006. To appear.

[5] Wolfgang Hoschek. *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. CERN, Geneva, 2004. Available at `http://dsd.lbl.gov/~hoschek/colt/`.

[6] ILOG inc. Ilog cplex, 2006. Optimization library. See `http://www.ilog.com/products/cplex/`.

[7] P. L'Ecuyer. *SSJ: A Java Library for Stochastic Simulation*, 2004. Software user's guide, Available at `http://www.iro.umontreal.ca/~lecuyer`.

[8] DRA Systems. Or-objects 1.2.4, 2000. Java mathematic library. Available at `http://opsresearch.com/`.