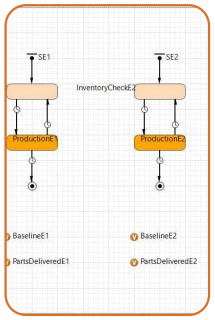
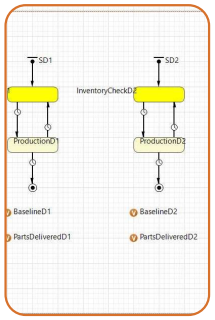
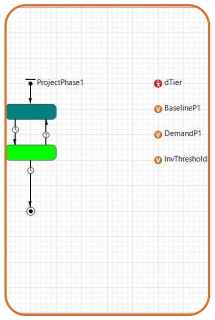
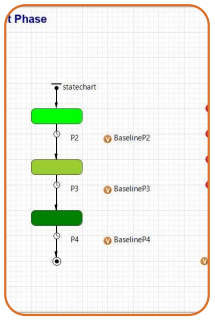
Building Project Simulation Model in AnyLogic









E-Tier Agent:

2nd supply tier, 2 suppliers

- Baseline set for each supply
- Each supplier has its own state chart to simulate order delivery
  - Calculates order cost
  - Tracks time to delivery
  - Time to delivery is set as probability distribution to simulate uncertainty
- Tracks when demand complete for this tier as inventory available for next tier

D-Tier Agent:

1st supply tier, 2 suppliers

- Baseline set for each supply
- Each supplier has its own state chart to simulate order delivery
  - Calculates order cost
  - Tracks time to delivery
  - Count does not start until previous tier demand is met
  - Loop checks to see if inventory from previous tier is complete
  - Time to delivery is set as probability distribution to simulate uncertainty
- Tracks when demand complete for this tier as inventory available for next tier

Project Phase 1 (P1) Agent

- Sets baseline for project phase 1
- State chart for project phase
  - Tracks time to completion
  - Count does not start until previous tier inventory threshold is met
  - Loop check to see if inventory threshold is met

Project Phases 2-4 Agent

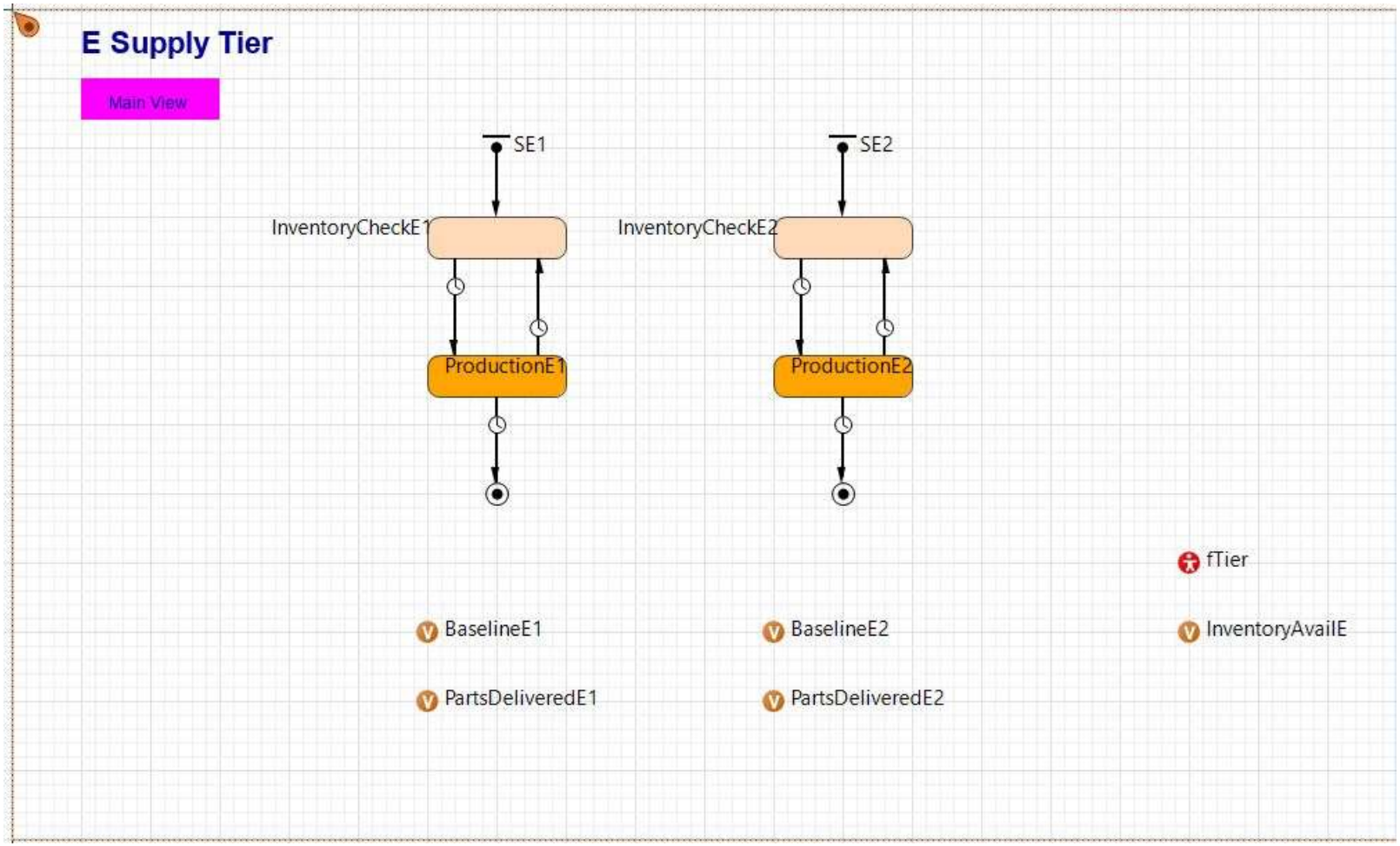
- Baseline set for project phases 2-4
- Coding calculates:
  - Baseline Project Costs Section 6.1.5
  - Project Costs Section 6.1.6
  - Project Schedule Baseline and Deviations Section 6.1.7
  - Forecasting Project Schedule Section 6.1.8

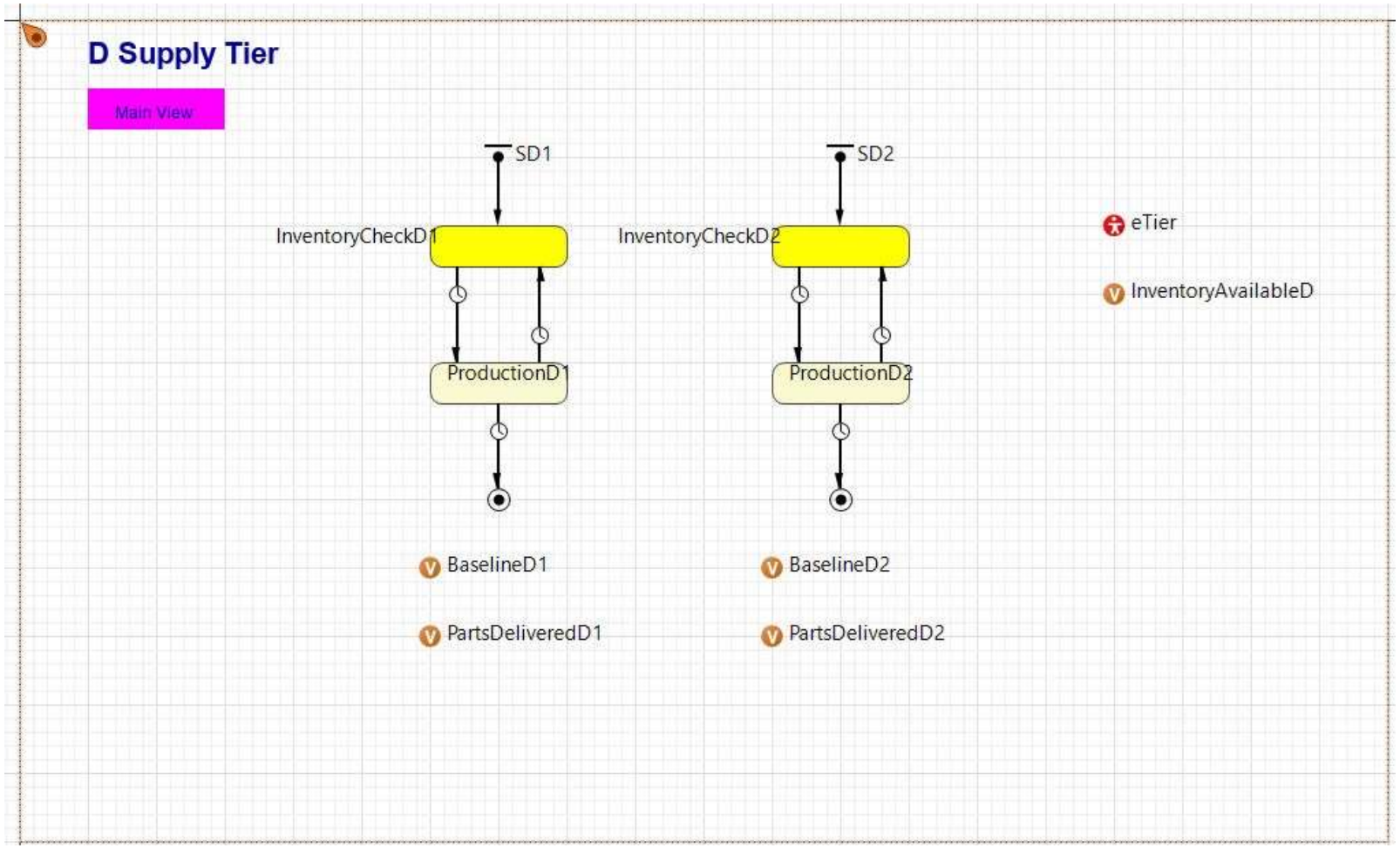
Main Agent

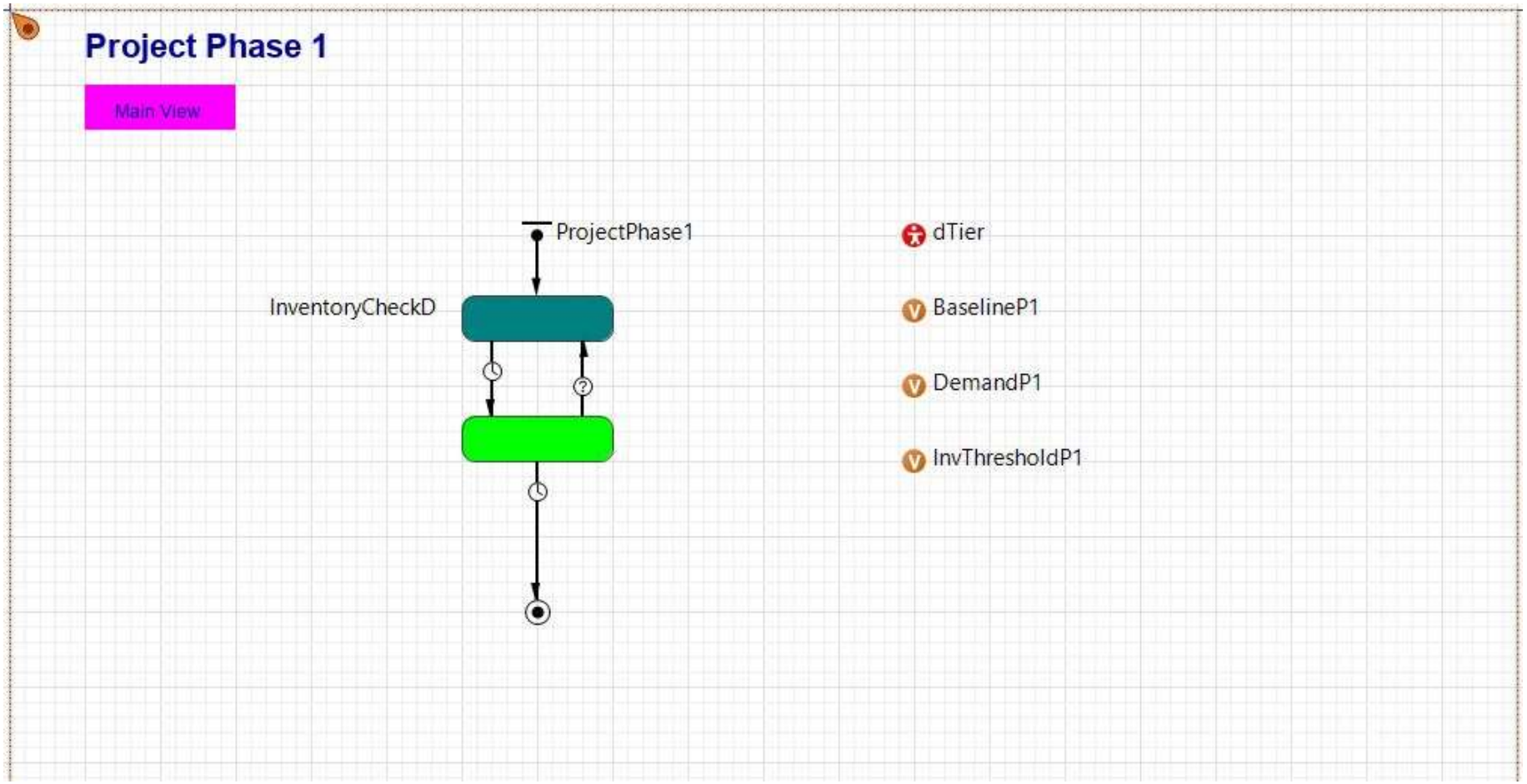
- Home to all other variables and parameters used in coding calculations
- Set parameters
  - Part costs
  - Bonus/penalty factors
  - Demand
  - Inventory Threshold
  - Order size
- Display to show results of single run simulation
  - Baselines, Completions, Deviations, Costs, Bonus/Penalties

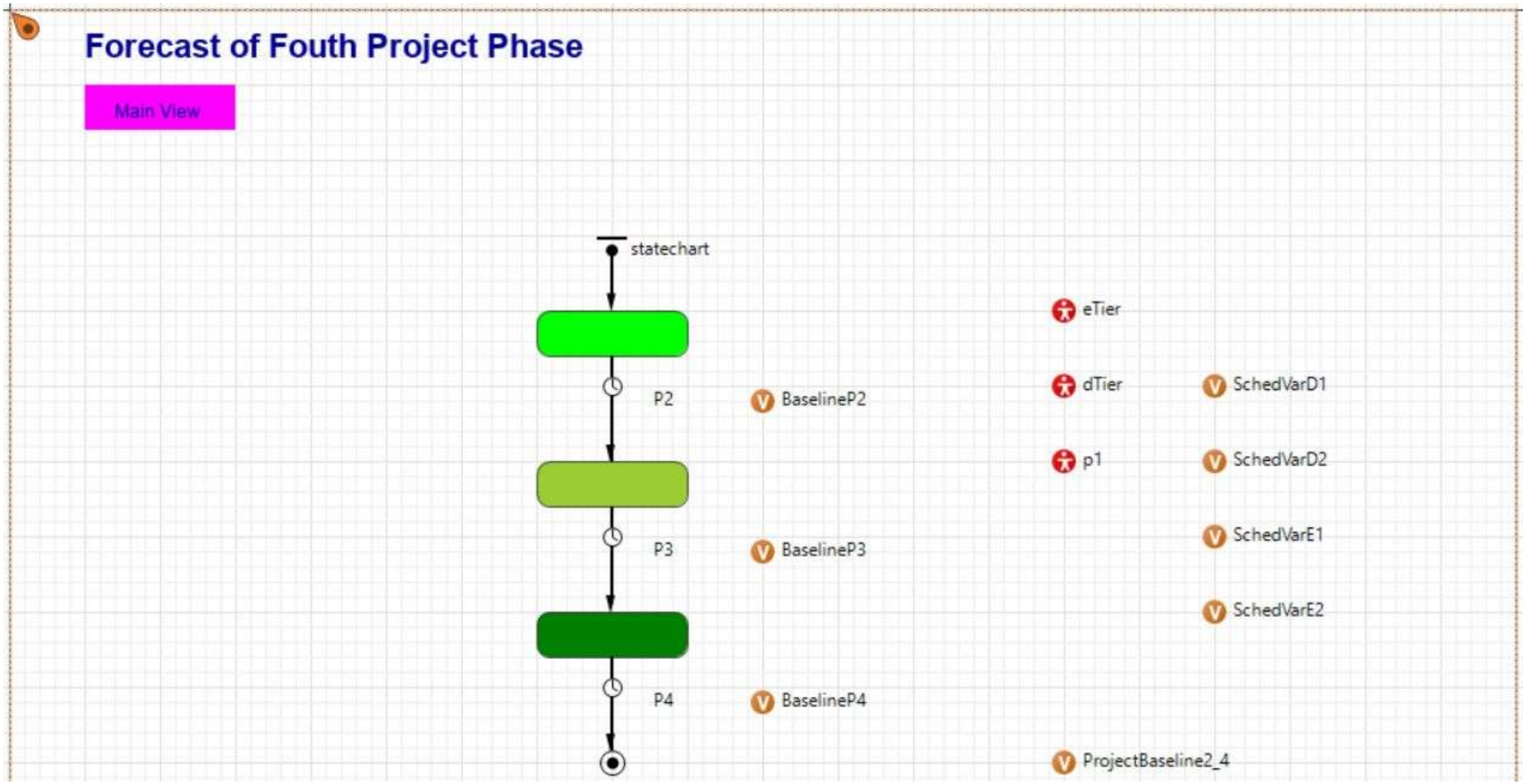
Simulation

- Single run of all agents, calculations, and coding
- Set with random number generator for all probability distributions
- Displays results in Main Agent


E Supply Tier
Main View
SE1
SE2
InventoryCheckE1
InventoryCheckE2
ProductionE1
ProductionE2
fTier
BaselineE1
BaselineE2
InventoryAvailE
PartsDeliveredE1
PartsDeliveredE2


D Supply Tier
Main View
SD1
SD2
InventoryCheckD1
InventoryCheckD2
eTier
InventoryAvailableD
ProductionD1
ProductionD2
BaselineD1
BaselineD2
PartsDeliveredD1
PartsDeliveredD2


Project Phase 1
Main View
ProjectPhase1
InventoryCheckD
dTier
BaselineP1
DemandP1
InvThresholdP1


Forecast of Fouth Project Phase
Main View
statechart
P2
BaselineP2
P3
BaselineP3
P4
BaselineP4
eTier
dTier
p1
SchedVarD1
SchedVarD2
SchedVarE1
SchedVarE2
ProjectBaseline2_4

AnyLogic Optimization Module

Properties: Top/Main
- Set top level agent to optimize (simulation model to optimize)
- Define objective function
- Set number of iterations, here set to 500

Properties: Parameters
- Determine which parameters are decision variables
- Select type discrete and set Min, Max, Step, Suggested Values
  - Minimum order size per supplier incorporate through min/max setting for decision variable
- Set all other parameters for optimization run

Properties: Constraints
- Add constraints that are tested before each simulation run
- Sum of supplier orders per tier must be equal to demand (Equation 6.5)

Properties: Requirements
- Add constraints that are tested after each simulation run to determine if solution is feasible
- Project Budget Equation 6.6- Manually calculated and set as upper boundary
- Bonus Penalty Equations 6.8 and 6.10: Boundary set as percentage of baseline

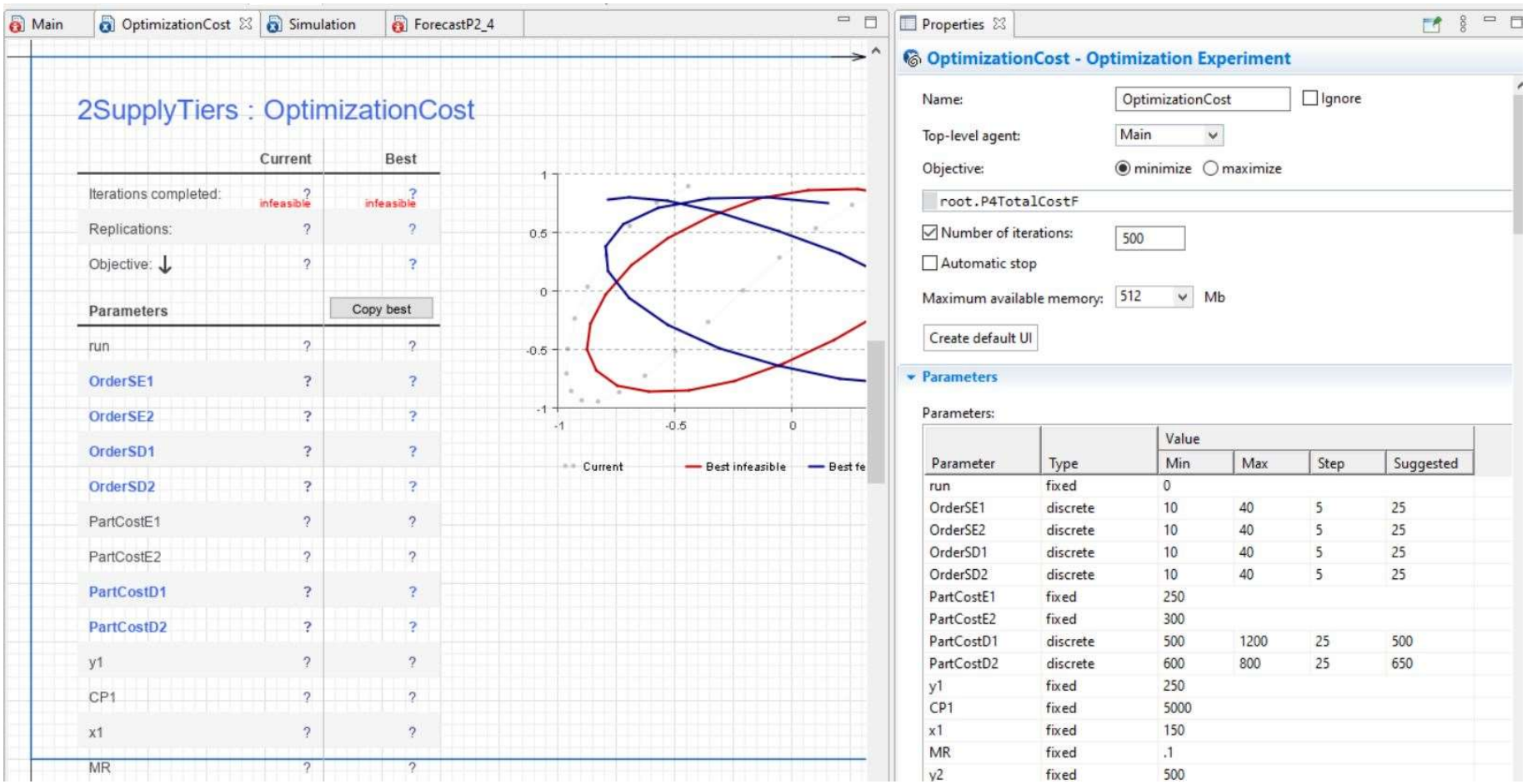
Properties: Randomness
- Set to Random seed to simulate uncertainty in probability distributions for supplier lead times and time to completion for project phases

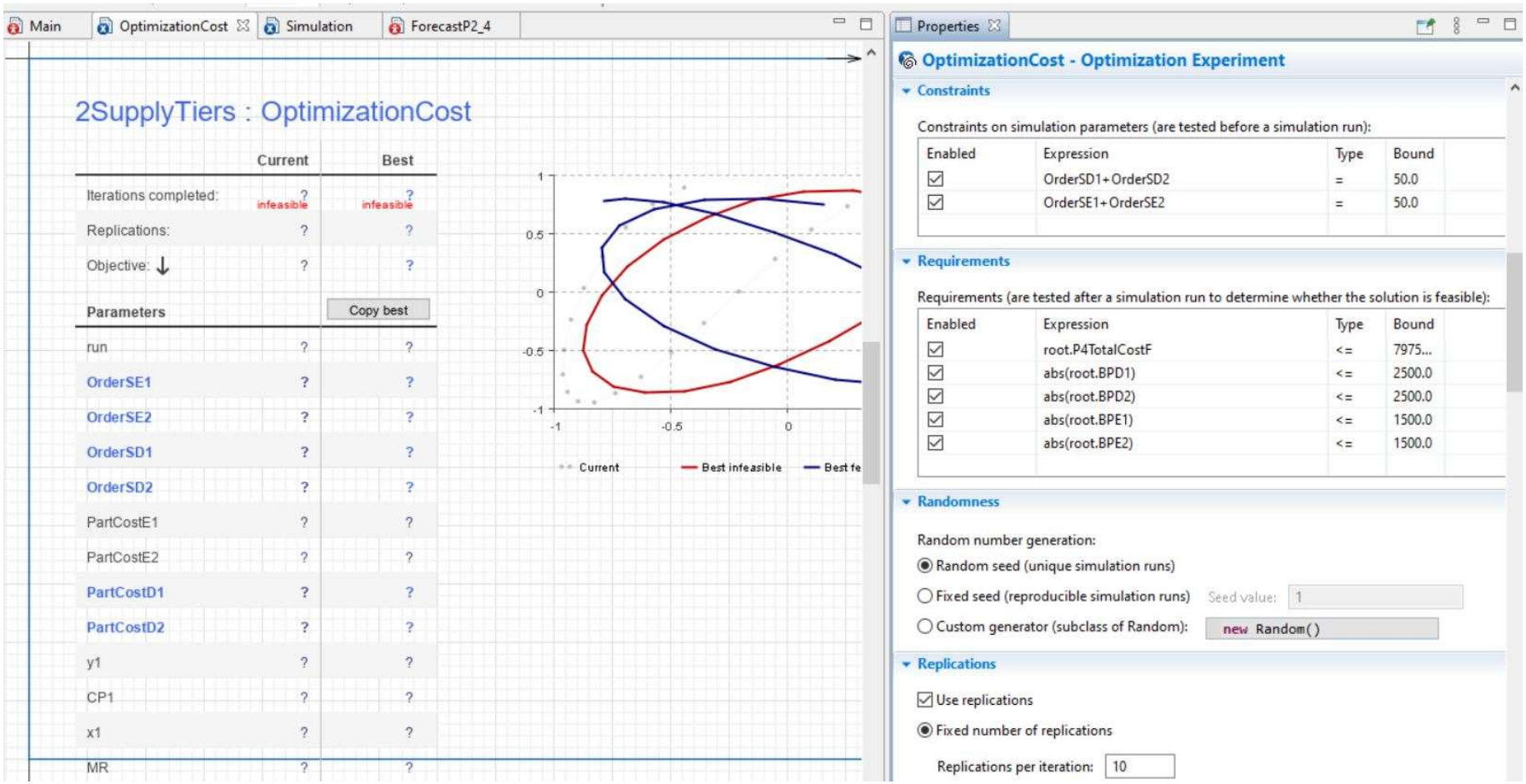
Properties: Replications
- Set number of replications
- Here set to 10 replications for each iteration

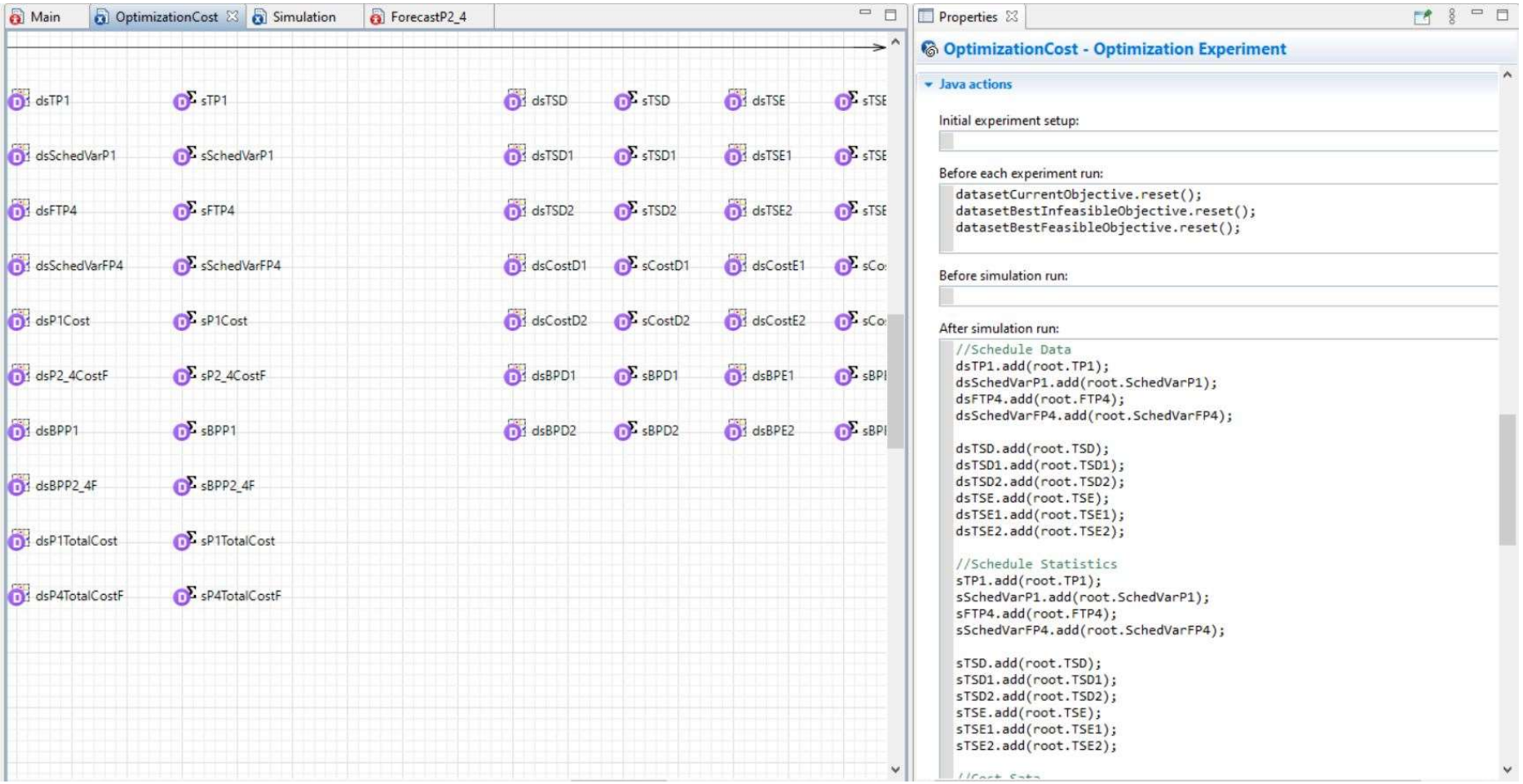
UI and Data
- Back in Property main, generate User Interface (UI) that will display best case and feasibility of solution
- Add data sets and statistics to collect information on variables with each run

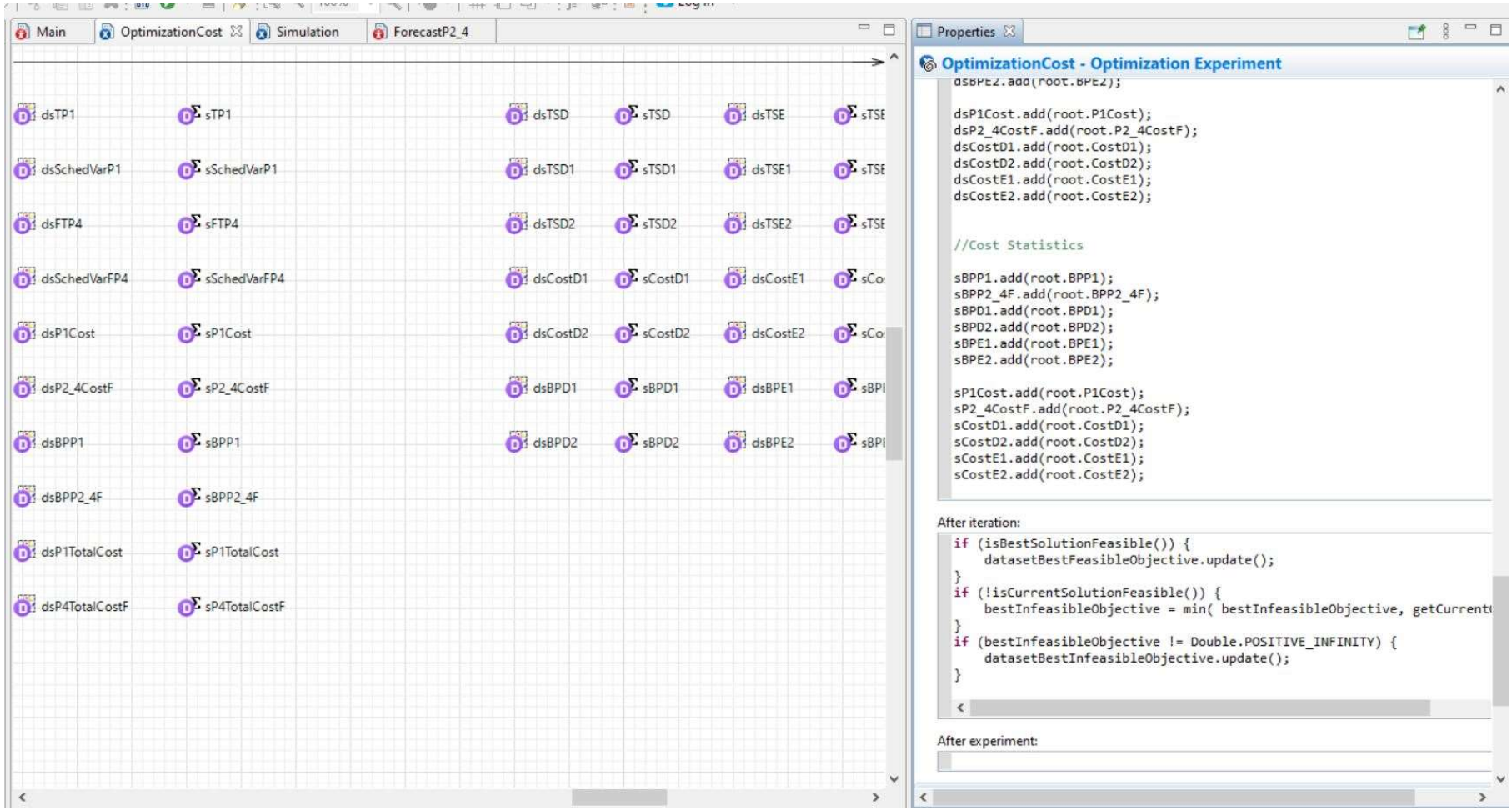
Properties: Java
- Further down on Simulation Property window, java actions for feasibility will be automatically generated with UI
- Add java to "After Simulation Run" to collect the data and statistics for the sets added to UI

Run
- Run optimization model to determine optimum order size for each supplier
- Runs the simulation model with properties set


Main
OptimizationCost
Simulation
ForecastP2_4
2SupplyTiers : OptimizationCost
Current
Best
Iterations completed:
infeasible
Replications:
Objective:
Parameters
Copy best
run
OrderSE1
OrderSE2
OrderSD1
OrderSD2
PartCostE1
PartCostE2
PartCostD1
PartCostD2
y1
CP1
x1
MR
Current
Best infeasible
Properties
OptimizationCost - Optimization Experiment
Name:
OptimizationCost
Ignore
Top-level agent:
Main
Objective:
minimize
maximize
root.P4TotalCostF
Number of iterations:
500
Automatic stop
Maximum available memory:
512
Mb
Create default UI
Parameters
Parameters:
Parameter
Type
Value
Min
Max
Step
Suggested
run fixed 0
OrderSE1 discrete 10 40 5 25
OrderSE2 discrete 10 40 5 25
OrderSD1 discrete 10 40 5 25
OrderSD2 discrete 10 40 5 25
PartCostE1 fixed 250
PartCostE2 fixed 300
PartCostD1 discrete 500 1200 25 500
PartCostD2 discrete 600 800 25 650
y1 fixed 250
CP1 fixed 5000
x1 fixed 150
MR fixed .1
y2 fixed 500


Main
OptimizationCost
Simulation
ForecastP2_4
2SupplyTiers : OptimizationCost
Current
Best
Iterations completed:
Replications:
Objective:
Parameters
Copy best
run
OrderSE1
OrderSE2
OrderSD1
OrderSD2
PartCostE1
PartCostE2
PartCostD1
PartCostD2
y1
CP1
x1
MR
Current
Best infeasible
Properties
OptimizationCost - Optimization Experiment
Constraints
Constraints on simulation parameters (are tested before a simulation run):
Enabled
Expression
Type
Bound
OrderSD1+OrderSD2
OrderSE1+OrderSE2
Requirements
Requirements (are tested after a simulation run to determine whether the solution is feasible):
root.P4TotalCostF
abs(root.BPD1)
abs(root.BPD2)
abs(root.BPE1)
abs(root.BPE2)
Randomness
Random number generation:
Random seed (unique simulation runs)
Fixed seed (reproducible simulation runs)
Seed value:
Custom generator (subclass of Random):
new Random()
Replications
Use replications
Fixed number of replications
Replications per iteration:


Main
OptimizationCost
Simulation
ForecastP2_4
dsTP1
sTP1
dsSchedVarP1
sSchedVarP1
dsFTP4
sFTP4
dsSchedVarFP4
sSchedVarFP4
dsP1Cost
sP1Cost
dsP2_4CostF
sP2_4CostF
dsBPP1
sBPP1
dsBPP2_4F
sBPP2_4F
dsP1TotalCost
sP1TotalCost
dsP4TotalCostF
sP4TotalCostF
dsTSD
sTSD
dsTSD1
sTSD1
dsTSD2
sTSD2
dsCostD1
sCostD1
dsCostD2
sCostD2
dsBPD1
sBPD1
dsBPD2
sBPD2
dsTSE
dsTSE1
dsTSE2
dsCostE1
dsCostE2
dsBPE1
dsBPE2
Properties
OptimizationCost - Optimization Experiment
Java actions
Initial experiment setup:
Before each experiment run:
datasetCurrentObjective.reset();
datasetBestInfeasibleObjective.reset();
datasetBestFeasibleObjective.reset();
Before simulation run:
After simulation run:
//Schedule Data
dsTP1.add(root.TP1);
dsSchedVarP1.add(root.SchedVarP1);
dsFTP4.add(root.FTP4);
dsSchedVarFP4.add(root.SchedVarFP4);
dsTSD.add(root.TSD);
dsTSD1.add(root.TSD1);
dsTSD2.add(root.TSD2);
dsTSE.add(root.TSE);
dsTSE1.add(root.TSE1);
dsTSE2.add(root.TSE2);
//Schedule Statistics
sTP1.add(root.TP1);
sSchedVarP1.add(root.SchedVarP1);
sFTP4.add(root.FTP4);
sSchedVarFP4.add(root.SchedVarFP4);
sTSD.add(root.TSD);
sTSD1.add(root.TSD1);
sTSD2.add(root.TSD2);
sTSE.add(root.TSE);
sTSE1.add(root.TSE1);
sTSE2.add(root.TSE2);


Main
OptimizationCost
Simulation
ForecastP2_4
dsTP1
sTP1
dsTSD
sTSD
dsTSE
dsSchedVarP1
sSchedVarP1
dsTSD1
sTSD1
dsTSE1
dsFTP4
sFTP4
dsTSD2
sTSD2
dsTSE2
dsSchedVarFP4
sSchedVarFP4
dsCostD1
sCostD1
dsCostE1
dsP1Cost
sP1Cost
dsCostD2
sCostD2
dsCostE2
dsP2_4CostF
sP2_4CostF
dsBPD1
sBPD1
dsBPE1
dsBPP1
sBPP1
dsBPD2
sBPD2
dsBPE2
dsBPP2_4F
sBPP2_4F
dsP1TotalCost
sP1TotalCost
dsP4TotalCostF
sP4TotalCostF
Properties
OptimizationCost - Optimization Experiment
dsP1Cost.add(root.P1Cost);
dsP2_4CostF.add(root.P2_4CostF);
dsCostD1.add(root.CostD1);
dsCostD2.add(root.CostD2);
dsCostE1.add(root.CostE1);
dsCostE2.add(root.CostE2);
//Cost Statistics
sBPP1.add(root.BPP1);
sBPP2_4F.add(root.BPP2_4F);
sBPD1.add(root.BPD1);
sBPD2.add(root.BPD2);
sBPE1.add(root.BPE1);
sBPE2.add(root.BPE2);
sP1Cost.add(root.P1Cost);
sP2_4CostF.add(root.P2_4CostF);
sCostD1.add(root.CostD1);
sCostD2.add(root.CostD2);
sCostE1.add(root.CostE1);
sCostE2.add(root.CostE2);
After iteration:
if (isBestSolutionFeasible()) {
datasetBestFeasibleObjective.update();
}
if (!isCurrentSolutionFeasible()) {
bestInfeasibleObjective = min(bestInfeasibleObjective, getCurrent
}
if (bestInfeasibleObjective != Double.POSITIVE_INFINITY) {
datasetBestInfeasibleObjective.update();
}
After experiment: