LINE Solver Cheatsheet

Queueing Theory Algorithms

Model Structure

```
model = Network('name');
source = Source(model, 'Src');
queue = Queue(model, 'Q',
    SchedStrategy.FCFS);
sink = Sink(model, 'Snk');
oclass = OpenClass(model, 'C1');
source.setArrival(oclass, Exp(1));
queue.setService(oclass, Exp(2));
model.link(Network.serialRouting(
    source, queue, sink));
SolverMVA(model).avgTable()
```

Node Types

`Source`	Job arrivals (open)
`Sink`	Job departures (open)
`Queue`	Queueing station
`Delay`	Infinite server
`Router`	Routing node
`Fork`	Forks jobs
`Join`	Joins jobs
`Cache`	Caching station
`ClassSwitch`	Class switching
`Place`	Petri net place
`Transition`	Petri net transition

Job Classes

```
OpenClass(model, 'name')
ClosedClass(model, 'name', N,
refNode)
SelfLoopingClass(model, 'name', N,
ref)
```

Scheduling Strategies

`FCFS`	First-Come First-Served
`LCFS`	Last-Come First-Served
`PS`	Processor Sharing
`INF`	Infinite Servers
`RAND`	Random (SIRO)
`HOL`	Head-of-Line Priority
`DPS`	Discriminatory PS
`GPS`	Generalized PS
`FCFSPRIO`	FCFS with Priority
`PSPRIO`	PS with Priority
`LCFSPR`	LCFS Preemptive

Distributions

`Exp(rate)`	Exponential
`Erlang(r, k)`	Erlang-k
`HyperExp(p,r1,r2)`	Hyper-exp
`Coxian(mu, phi)`	Coxian
`APH(alpha, T)`	Acyclic PH
`PH(alpha, T)`	Phase-type
`MAP(D0, D1)`	MAP
`MMPP2(l1,l2,s1,s2)`	MMPP(2)
`Det(value)`	Deterministic
`Uniform(a, b)`	Uniform
`Gamma(a, b)`	Gamma
`Pareto(a, k)`	Pareto
`Weibull(a, b)`	Weibull
`Lognormal(m, s)`	Log-normal
`Replayer(trace)`	Trace replay
`Disabled`	None

Fitting: `Exp.fitMean(m)`
`HyperExp.fitMeanAndSCV(m,scv)`

Solvers

`SolverAUTO`	Auto-select best
`SolverCTMC`	CTMC (exact, small)
`SolverDES`	Discrete Event Sim
`SolverEnv`	Random environments
`SolverFluid`	ODE fluid approx
`SolverJMT`	JMT simulator
`SolverLN`	Layered networks
`SolverLQNS`	LQNS interface
`SolverAG`	Agent solver (RCAT)
`SolverMAM`	Matrix Analytic
`SolverMVA`	Mean Value Analysis
`SolverNC`	Normalizing Constant
`SolverQNS`	LQNS QN solver
`SolverSSA`	Stochastic simulation

Solver Methods

Use `listMethods()` to see available methods

MVA: `exact`, `mva`, `amva`, `bs`, `lin`, `qd`, `mm1`, `mmk`, `mg1`, `gigk`
NC: `exact`, `ca`, `comom`, `mva`, `ls`, `le`, `panacea`
SSA: `serial`, `para`, `nrm`
FLUID: `matrix`, `statedep`, `closing`

Solver Options

```
SolverMVA(model, 'method', 'exact')
SolverJMT(model, 'seed', 123,
    'samples', 10000)
SolverSSA(model, 'samples', 5000,
    'method', 'para')
```

Output Metrics

`QLen`	Mean queue length
`Util`	Server utilization
`RespT`	Response time (per visit)
`ResidT`	Residence time (total)
`Tput`	Throughput
`ArvR`	Arrival rate

Analysis Methods

```
solver.avgTable()
solver.avgQLen()
solver.avgUtil()
solver.avgRespT()
solver.avgTput()
solver.avgSysRespT()
solver.avgSysTput()
solver.tranAvg(t0, t1)
% Chain avg methods
solver.avgChainQLen()
solver.avgChainRespT()
solver.avgChainTput()
```

Routing

```
% Serial
model.link(Network.serialRouting(
    n1,n2,n3));
% Probabilistic
n1.setProbRouting(class, n2, 0.7);
% Round-robin / Random
n1.setRouting(class,
    RoutingStrategy.RROBIN);
```

Multi-Server & Finite Capacity

```
queue.setNumServers(k);
queue.setCapacity(bufferSize);
```

Class Switching

```
cs = ClassSwitch(model, 'CS');
P = [0.3 0.7; 0.5 0.5];
cs.setClassSwitching(P);
```

Fork-Join Networks

```
fork = Fork(model, 'Fork');
join = Join(model, 'Join', fork);
```

Layered Networks

```
lqn = LayeredNetwork('name');
P = Processor(lqn,'P',1,
    SchedStrategy.PS);
T = Task(lqn,'T',1,
    SchedStrategy.REF).on(P);
E = Entry(lqn,'E').on(T);
A = Activity(lqn,'A').on(T)
    .boundTo(E)
    .setHostDemand(Exp(1));
T.addPrecedence(
    ActivityPrecedence.Serial(A));
```

Caching

```
cache = Cache(model,'Cache',
    nItems,cap,
    ReplacementStrategy.LRU);
cache.setPopularity(Zipf(1.4));
```

Random Environments

```
env = Environment(model, 'Env');
env.addParameter('rate', 0.5);
Queue(model,'Q').setRate(
    env.get('rate'));
SolverEnv(model).avgTable()
```

State Probabilities

```
SolverCTMC(model).getStateProb()
SolverNC(model).getProb(state)
```

Model Import/Export

```
model.view()    % Open in JMT
model.save('file.jsimg')
Network.load('file.jsimg')
```

Gallery Models

```
gallery_mm1(),          gallery_mmk(),
gallery_mg1(),   gallery_mm1_tandem(),
gallery_cqn()
```

Verbosity

```
GlobalConstants.setVerbose(
    VerboseLevel.SILENT)
```

Quick Start

MATLAB: `cd matlab; lineStart`
Java/Kotlin: `cd jar; mvn package -Pb`
Python: `pip install -r requirements.txt`

Resources

`http://line-solver.sf.net`
Examples: `examples/`, `gallery/`