LINE Solver Cheatsheet (C++)

http://line-solver.sf.net

Node Types

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

C++ nodes are created on the model, e.g. `model.add_queue("Q", SchedStrategy::FCFS)`, which returns the node index.

Job Classes

```
model.add_closed_class("name", N, refNode)
model.add_open_class("name")
model.set_signal(cls, SignalType::CATASTROPHE)
```

(no self-looping or disabled class)

Distributions

`APH(`α`, T)`	Acyclic PH
`Cox2(`μ`, `ϕ`)`	Two-phase Coxian
`Coxian(`μ`, `ϕ`)`	Coxian
`Det(v)`	Deterministic
`Disabled`	None
`Erlang(`λ`, k)`	Erlang-k
`Exp(`λ`)`	Exponential
`Gamma(`α`, `β`)`	Gamma
`HyperExp(p, `λ_1`, `λ_2`)`	Hyper-exp
`Immediate`	Zero service time
`Lognormal(`μ`, `σ`)`	Log-normal
`Normal(`μ`, `σ`)`	Normal
`Pareto(`α`, k)`	Pareto
`PH(`α`, T)`	Phase-type
`Uniform(a, b)`	Uniform
`Weibull(`α`, `β`)`	Weibull

Discrete

`Bernoulli(p)`	Bernoulli
`Binomial(p, n)`	Binomial
`DiscreteUniform(a, b)`	Discr. uniform
`Geometric(p)`	Geometric
`Poisson(`λ`)`	Poisson
`Zipf(s, n)`	Popularity

Correlated / trace-driven

`BMAP(D0..Dk)`	Batch MAP
`NHPP(bp,`λ`,cyclic)`	Non-homog. Poisson
`DMAP(D0, D1)`	Discrete MAP
`EmpiricalCDF(F)`	Empirical CDF
`MAP(D0, D1)`	MAP
`MarkedMAP`	Marked MAP
`ME(`α`, T)`	Matrix-exp.
`CME(mean, order)`	Concentr. matrix-exp.
`MMPP2(`λ_1`, `λ_2`, `σ_1`, `σ_2`)`	MMPP(2)
`RAP(H0, H1)`	Rational AP
`Replayer(trace)`	Trace replay

In C++ a distribution is a factory on `Distrib<double>`, e.g. `Distrib<double>::exp_rate(0.5)` or `::exp_mean(2.0)`.

Scheduling Strategies

`DPS`	Discriminatory PS
`EDD`	Earliest Due Date
`EDF`	Earliest Deadline First
`EXT`	External (source)
`FB`	Feedback (LAS)
`FCFS`	First-Come First-Served
`FCFSPI`	FCFS Preempt. Resume-idle
`FCFSPR`	FCFS Preemptive Resume
`FSP`	Fair Sojourn Protocol
`GPS`	Generalized PS
`HOL`	Head-of-Line Priority
`INF`	Infinite Servers
`LCFS`	Last-Come First-Served
`LCFSPI`	LCFS Preempt. Resume-idle
`LCFSPR`	LCFS Preemptive Resume
`LEPT`	Longest Expected PT
`LJF`	Longest Job First
`LPS`	Limited PS
`LRPT`	Longest Remaining PT
`OI`	Order Independent
`PAS`	Pass and Swap
`POLLING`	Polling (switchover)
`PS`	Processor Sharing
`PSJF`	Preemptive SJF
`SEPT`	Shortest Expected PT
`SETF`	Shortest Elapsed Time First
`SIRO`	Random
`SJF`	Shortest Job First
`SRPT`	Shortest Remaining PT

Solvers

`AUTO`	Auto-select best
`BA`	Bound analysis (closed)
`CTMC`	CTMC (exact, small)
`ENV`	Random environments
`FLD`	Fluid/mean-field ODEs
`LDES`	Discrete Event Sim
`LN`	Layered networks
`MAM`	Matrix Analytic Methods
`MVA`	Mean Value Analysis
`NC`	Normalizing Constant Methods
`SSA`	Stochastic simulation

Wrappers

`JMT`	JMT simulator
`LQNS`	LQNS interface
`QNS`	LQNS product-form solver

Solver Options

```
mva::MvaOptions o; o.method = "exact";
mva::solver_mva_run(sn, o, Matrix<double>());
ssa::SsaOptions s; s.samples = 5000;
ssa::solver_ssa(sn, s);
```

Output Metrics

`ArvR`	Arrival rate
`QLen`	Mean queue length
`ResidT`	Residence time (total across visits)
`RespT`	Response time (per visit)
`Tput`	Throughput
`Util`	Server utilization
`QLenVar`	Queue-length variance
`QLenSCV`	Queue-length squared coeff. of variation
`QLenSkew`	Queue-length skewness
`RespTVar`	Response-time variance (FCFS only)
`RespTSCV`	Response-time SCV (FCFS only)
`RespTSkew`	Response-time skewness (FCFS only)

Moments (product form)

Queue-length moments beyond the mean are not part of the C++ port. *order* is a set: scalar k = up to k;
a vector, e.g. `[2,3]`, = exactly those.
`RespT*` moments are FCFS-only (NaN elsewhere); exact methods cost $\prod_r (N_r+1)$, use `'lin'` beyond that.

Analysis Methods

```
mva::AvgResult<double> r =
mva::solver_mva_run(sn, o, Matrix<double>());
r.QN, r.UN, r.RN, r.TN, r.AN, r.WN
r.CN (system respT), r.XN (system tput)
```

Result Fields

`QN`	queue length
`UN`	utilization
`RN`	response time
`TN`	throughput
`CN, XN`	system totals

Routing

```
qn::RoutingMatrix<double> P;
P.set(n1, n2, 0.7);
P.set(n1, n3, 0.3);
model.link(P);
model.set_routing(n1, cls, RoutingStrategy::RROBIN);
```

Multi-Server & Finite Capacity

```
model.set_number_of_servers(queue, k);
model.set_capacity(queue, bufferSize);
```

Class Switching

```
Matrix<double> C(2,2);
C(0,0)=0.3; C(0,1)=0.7;
C(1,0)=0.5; C(1,1)=0.5;
model.add_class_switch("CS", C);
```

Fork-Join Networks

```
std::size_t fork = model.add_fork("Fork");
std::size_t join = model.add_join("Join", fork);
```

Layered Networks

```
lqn::LqnBuilder<double> lqn;
lqn.processor("P", 1, SchedStrategy::PS);
lqn.task("T", 1, SchedStrategy::REF, "P");
lqn.entry("E", "T");
lqn.activity("A", Distrib<double>::exp_mean(1), "T");
lqn.bound_to("A", "E");
```

Caching

```
qn::CacheParam<double> par;
par.nitems = nItems;
par.itemcap.assign(1, cap);
par.replacestrat = ReplacementStrategy::LRU;
par.pread[cls-1] = popularity;
model.add_cache("Cache", par);
```

ReplacementStrategy

`CLIMB`	Climb
`FIFO`	First-in first-out
`HLRU`	h-LRU
`LRU`	Least recently used
`QLRU`	q-LRU
`RR`	Random replacement
`SFIFO`	Static FIFO

Model Import/Export

```
auto model = io::read_network_json<double>("model.json");
// or from the shell:
line-cli -f model.json -s mva -a avg
```

Routing Strategies

`JSQ`	Join shortest queue
`KCHOICES`	Power of k choices
`PROB`	Probabilistic routing
`RAND`	Random selection
`RROBIN`	Round-robin
`WRROBIN`	Weighted round-robin

Stochastic Petri Nets

```
qn::TransitionParam<double> tp;
tp.nmodes = 1;
tp.enabling[0][p-1] = 1;
tp.inhibiting[0][p-1] = 3;
tp.firing[0][sink-1] = 1;
tp.firingproc.push_back(Distrib<double>::exp_rate(4));
model.add_transition("T1", tp);
```

Markov Chains (solved directly)

```
std::vector<double> pi = mc::ctmc_solve(Q);
std::vector<double> piD = mc::dtmc_solve(P);
```

Random Environments

```
env::Environment<double> e("R", 2);
e.set_stage(0,"Fast","operational",fast.get_struct());
e.set_stage(1,"Slow","degraded",slow.get_struct());
e.add_transition(0,1,Distrib<double>::exp_rate(0.5));
e.add_transition(1,0,Distrib<double>::exp_rate(1.0));
env::solver_env(e, env::EnvOptions());
```

Finite Capacity Regions

```
model.add_region({queue1, queue2},
{5.0, -1.0},
10.0,
{DropStrategy::DROP});
```

Load-Dependent Service

```
model.set_load_dependence(queue, alpha);
model.set_class_dependence(queue, fun, peak);
model.set_joint_dependence(queue, fun, peak);
```

Impatience & Retrials

```
model.set_retrial(queue, cls,
Distrib<double>::exp_rate(2), rate, maxAttempts);
// no patience or balking in the C++ port
```

State Probabilities

```
mva::solver_mva_get_prob_marg(sn, avg, o, node, cls);
mva::solver_mva_get_prob_aggr(sn, avg, o, node);
nc::solver_nc_getprob_sys_marg(sn, o, nvec);
ctmc::ctmc_get_generator(sn, o);
ctmc::ctmc_get_state_space(sn, o);
```

Transient & Distributions

```
ctmc::solver_ctmc_transient_analyzer(sn, o, t0, t1);
ctmc::solver_ctmc_cdf_respt(sn, o);
model.set_reward("Cost", fn);
ctmc::solver_ctmc_avg_reward(sn, o, rewards);
```