Simulating Synchronized Queueing Networks

Christopher Gomez

Introduction

Education

- B.S. Computer Science
- B.S. Neuroscience and Psychology
- M.S. Computer Science
  - GPA: **4.0**, Concentration: Data Science

Related Work Experience

- MIT Lincoln Lab ML Intern (Embedded Systems Group)
- Precisely (Pitney Bowes) Exploratory ML Intern

Interests

- Hiking, board games, Monty Python, 70s rock, LoTR, Leonard Cohen, sci-fi/fantasy, tea, baking, volunteering

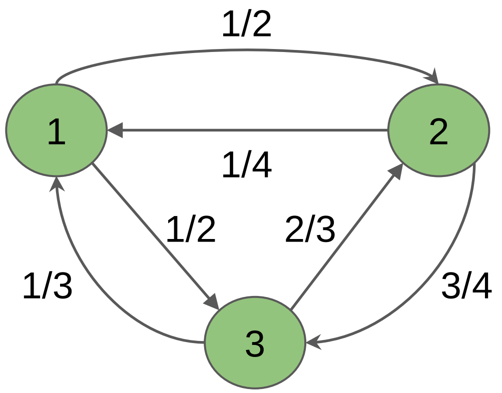
Presentation Overview

1. Background on Markov Processes
2. General Semi-Markovian Processes (GSMPs)
3. Synchronized Queueing Networks (SQNs)
4. Modeling Synchronized Queueing Networks
5. Generating Initial Graphs
6. Building Restructured Graphs into SQNs
7. Output and Analysis

Background on Markov Processes

Markov Chains

- A stochastic model for representing a sequence of states and state transitions. Often time-homogenous.
- Entering the next state depends ONLY on current state
- Examples: inventory analysis, slot machines

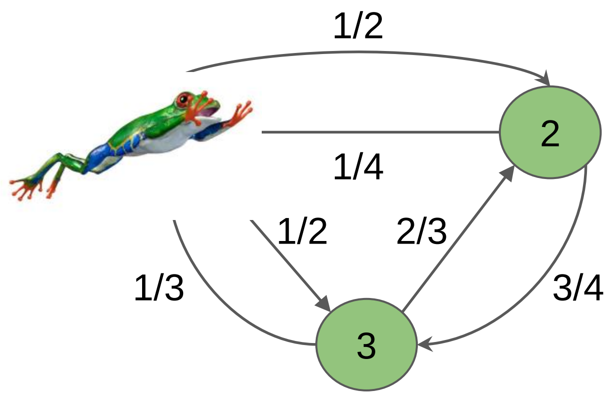


Transition Matrix

0	1/2	1/2
1/4	0	3/4
1/3	2/3	0

Markov Chains

- A stochastic model for representing a sequence of states and state transitions. Often time-homogenous.
- Entering the next state depends ONLY on current state
- Examples: inventory analysis, slot machines, frog leaps



Transition Matrix

0	1/2	1/2
1/4	0	3/4
1/3	2/3	0

General Semi-Markovian Processes (GSMPs)

General Semi-Markovian Process (GSMP)

- Specification of Markov chains allowing random and variable time intervals between state transitions
- **Events** are created on state transitions and have **clock timers**
  - Each event clock can have its own state-transition distribution
- Events compete to trigger the next state transition
- State transitions allow generating new events!
  - They also allow deactivating, reactivating, and deleting events

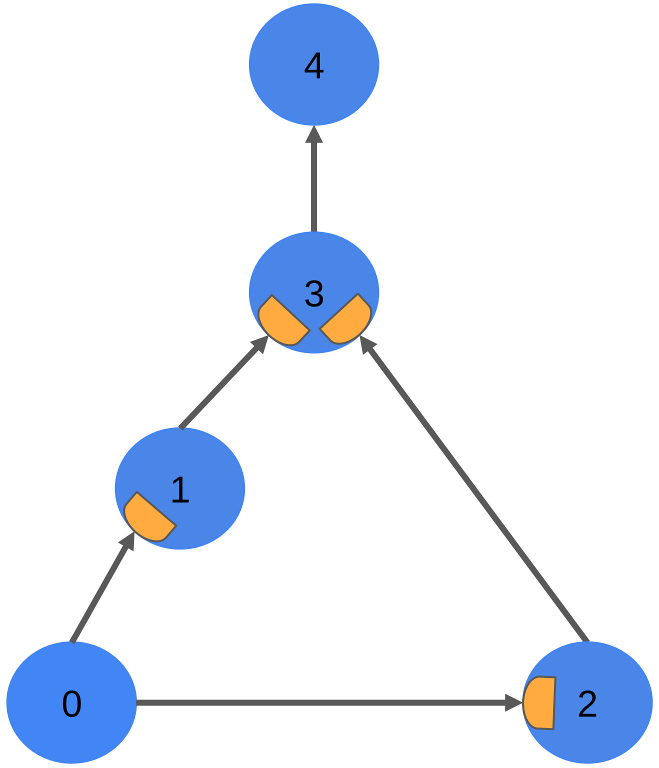
Benefits of GSMPs

- **Flexibility in Timing Modeling** - GSMPs allows modeling modeling complex and dynamic systems
  - Examples: Manufacturing processes, telecommunication systems, queueing networks, or even ambulatory care
- **Informed Decision Making -** resource management, “what if analysis”, policy changes!
- **Event-Based Analyses** - event creation on state transitions provides a structured framework for analyzing temporality
  - Example: tracking the average time duration between the event D (‘patient makes ER call’) and event G (‘ambulance picks up patient’)

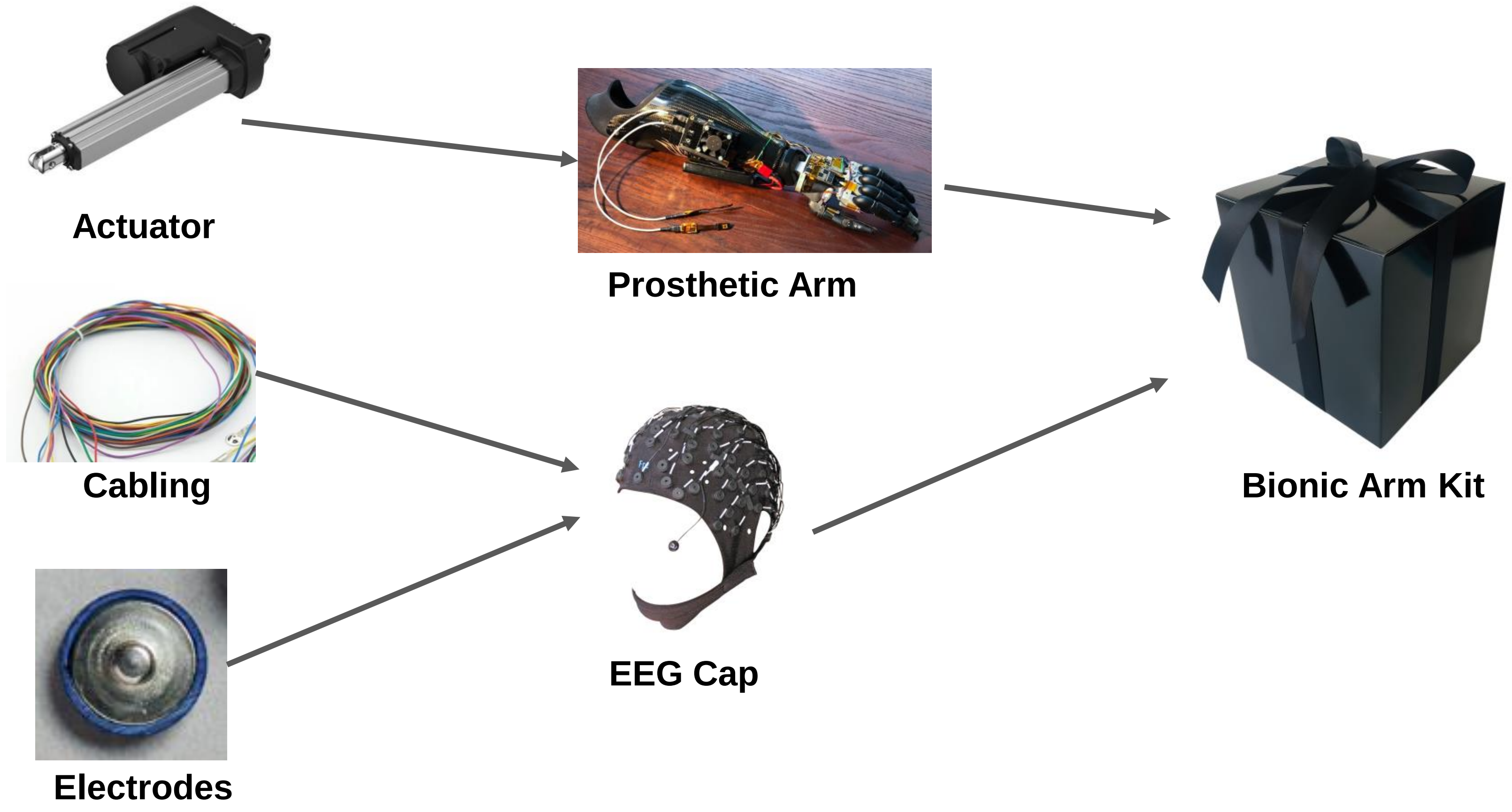
Synchronized Queueing Networks (SQNs)

Synchronized Queueing Network

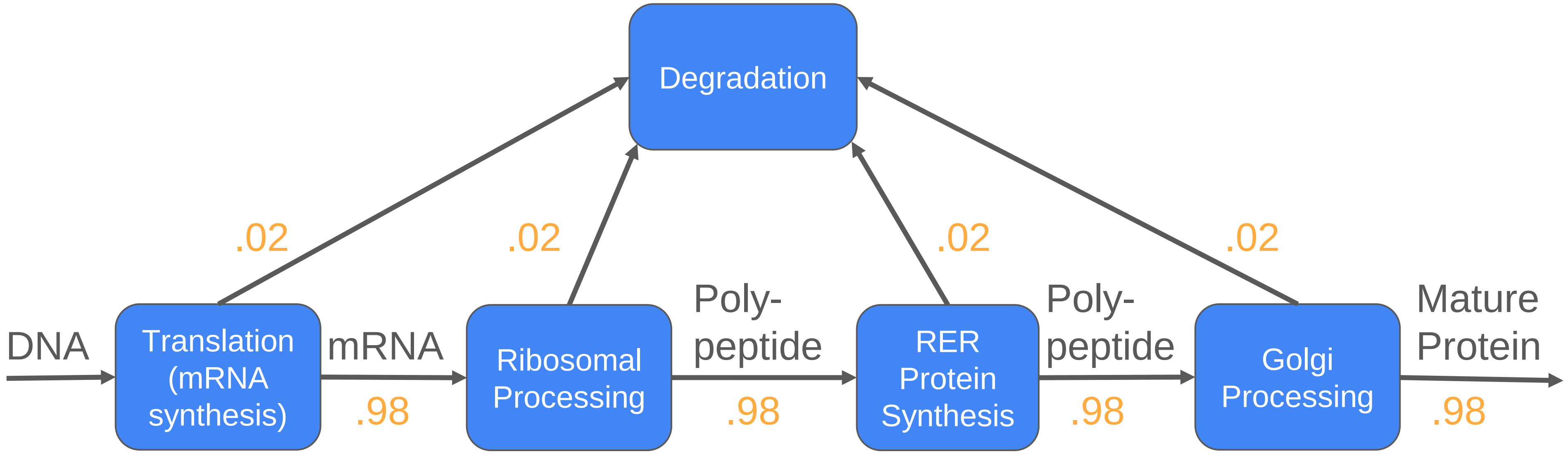
- Synchronized Queueing Networks enable representing the flow of tasks moving through a network of interconnected queues
- Every node requires at least one input from all of its incoming edges before being able to proceed
- Transitions are stochastic



SQN: The Factory Analogy



SQN Example: Modeling Protein Synthesis



Modeling Synchronized Queueing Networks

State Space

$$S \in \{(M_0, M_1, \ldots, M_n)\} = \{\{A_0\}, \{A_1, B_0, B_1, \ldots B_{J1}\}, \ldots \{A_n, B_0, B_1, \ldots B_{Jn}\}\}$$
$$\in \{\{1\}\} \times \{\{0,1\} \times \{0,1,\ldots,U\}_0 \ldots \{0,1,\ldots,U\}_{J1}\} \times \ldots \{\{0,1\} \times \{0,1,\ldots,U\}_0 \ldots \{0,1,\ldots,U\}_{Jn}\}$$

- M_i represents a production node with J incoming edges. Its state is $\{A_i, B_0, B_1, \ldots B_{Ji}\}$ which represents whether it is actively producing and the number of 'parts' acquired in each bin
  - Each bin corresponds to one of the J incoming edges.
- A_i is 1 if currently producing and 0 if idle/waiting for parts.
- There is a universal maximum cap of U number of parts for every bin.

Event Space

- $e_0 \in E(s)$
  - e_0 is the event representing the part completion for the source production node (M_0)
  - This event can always occur.
- $e_i \in E(s)$ if $\sum_{j=0}^{J} I(B_j, M_i) == J$
  - Indicator function $I(B_j, M_i) == 1$ if bin $B_j \geq 1$ for node M_i
  - Node M_i can only begin production once all J of its bins are filled

Transition Probabilities

- $N(s, M_i)$ = $\sum_{j=0}^{n} I(s)$ with indicator function $I(s)$ == 1 if M_i has an outgoing edge to M_j, this function totals all outgoing edges from arbitrary node M_i.
- $p(s', s, e_i) = 1/N(s)$ with $s = (M_0(t), \ldots, M_n(t))$ having an $M_i = (A_i, B_0, B_1, \ldots, B_{Ji})$ with an $A_i = 1$ and an $M_j = (A_j, B_0, B_1, \ldots, B_{Jj})$ with a $B_j = k$. The new state s' has an $M_i = (A_i, B_0, B_1, \ldots, B_{Ji})$ with an $A_i = 0$ and an $M_j = (A_j, B_0, B_1, \ldots, B_{Jj})$ with a $B_j = k + 1$.
    - Upon node M_i's completion of a new part (event e_i), node M_i is marked as deactivated and then each of the nodes that has an incoming connection from M_i has equal chance of being selected. On selection, the chosen node M_j has the bin corresponding to M_i incremented by 1.
- $p(s', s, e_0) = 1$ with $s = (M_1(t), \ldots, M_n(t))$ having an $M_i = (0, k_0, k_1, \ldots k_J)$ and $s' = (M_1(t), \ldots, M_n(t))$ having an $M_i = (1, k_0 - 1, k_1 - 1, \ldots k_J - 1)$
    - Once an inactive node M_i has at least 1 in all of its bins it is certain to become active and have all of its bins decremented by 1.

Clock Distribution Timer

- $F(x; s', e', s, e^*) = F_p(x)$ with $F_p(x) = 1 - e^{-1/\beta x}$ being an exponential process with an average time of β minutes.
- For trials β was set to 0.5

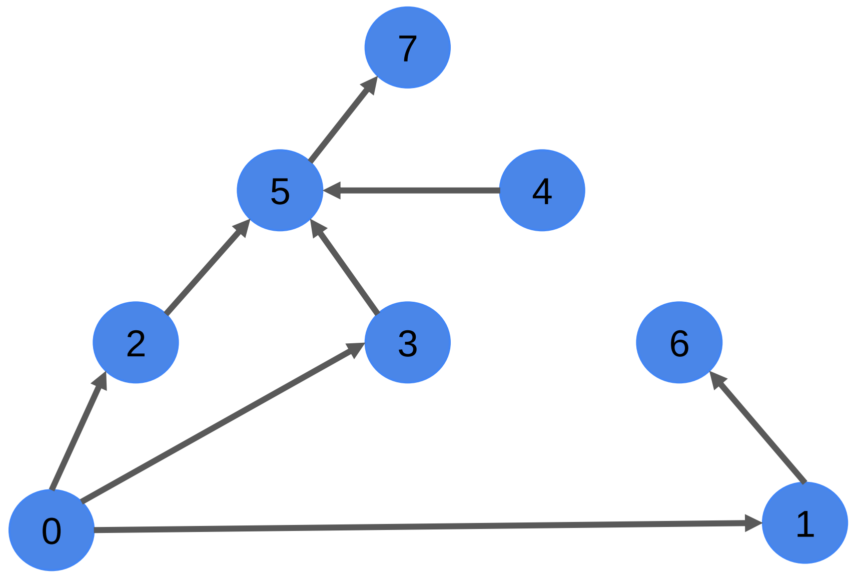
Initial State Distribution

- $V(s_0) = \{M_0(0), M_1(0), \dots, M_n(0)\}$
  $= \{(A_0, (A_1, B_0, B_1, \dots, B_{J1}), \dots (A_n, B_0, B_1, \dots, B_{Jn})\}$
  $= \{\{1\}, \{0, 0 \dots 0\}, \dots \{0, 0 \dots 0\}\}$
- Start with nothing produced, all nodes inactive and requiring each of its J_i bins ≥ 1 to begin production.
- Note that every Node M_i has a different number of incoming edges, so every J_i varies per node.
- The source node M_0 is always producing and has no incoming edges, hence A_0 is always 1 and it has no bins.

Generating Initial Graphs

Initial Graph Generation

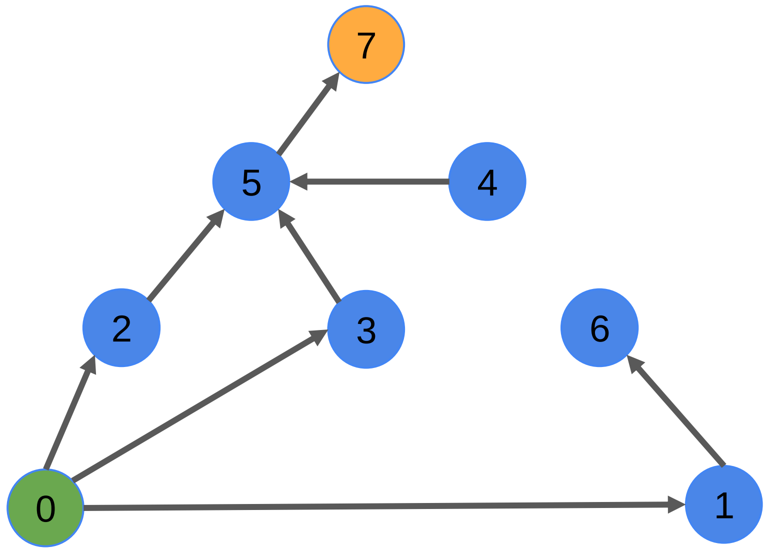
- Generate a graph with an arbitrary number of nodes
- Edge establishment based on a random coin flip
- Restriction: Nodes are permitted to establish outgoing edges solely to nodes with numerically higher values



Unaltered

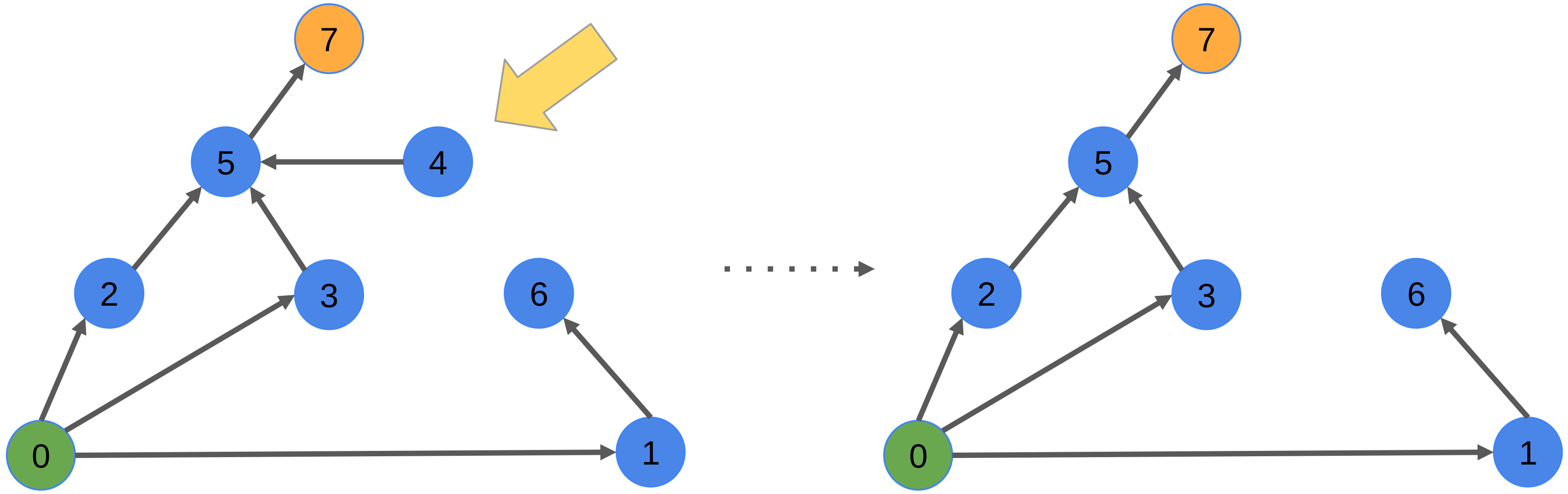
Optional Network Adjustments

- **Pruning** - removing nodes depending on their number of incoming/outgoing edges
- **Restructuring** - force every node in the generated graph to have at least one incoming and outgoing edge
    - Exceptions: source node and sink node



Unaltered

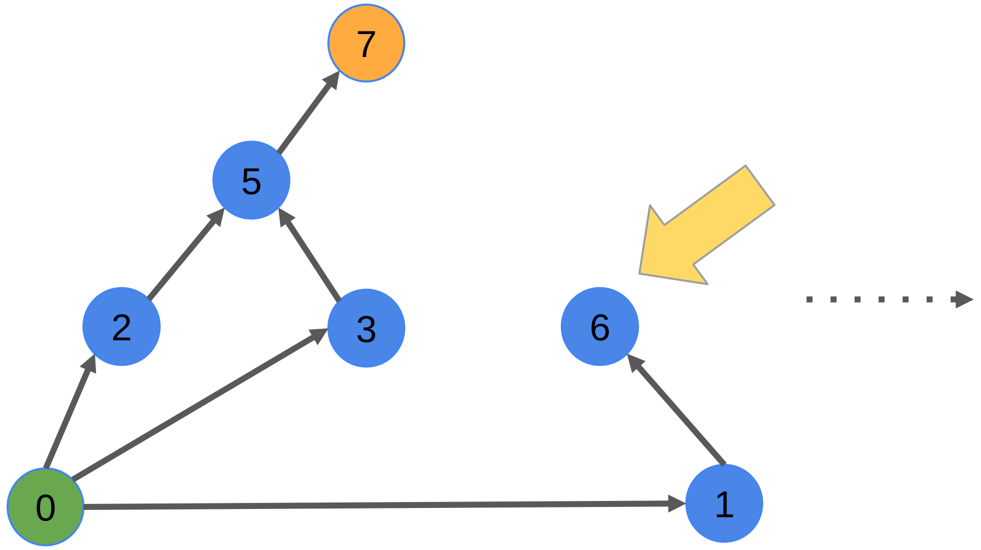
Pruning non-source nodes with no incoming edges



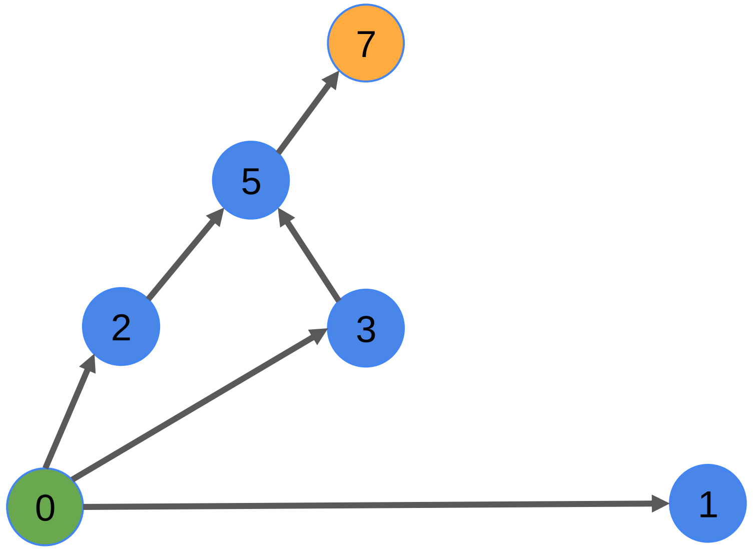
Unaltered

Once-pruned

Pruning non-sink nodes with no outgoing edges

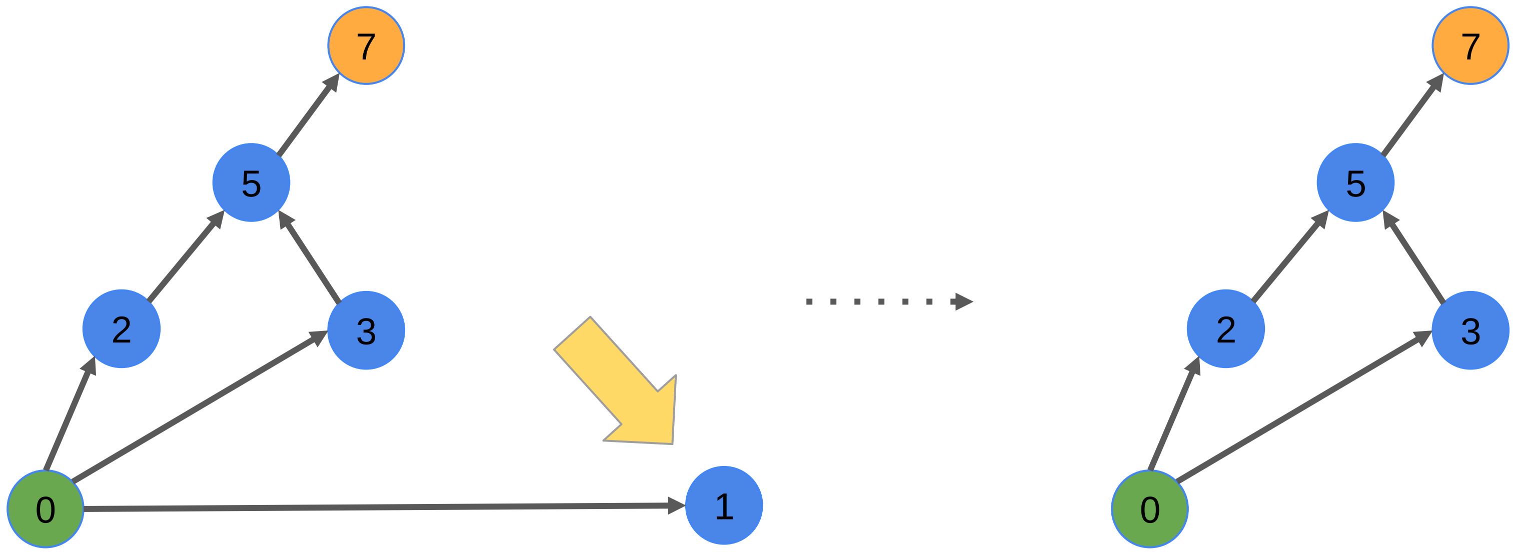


Once-pruned



Twice-pruned

Continuous Pruning



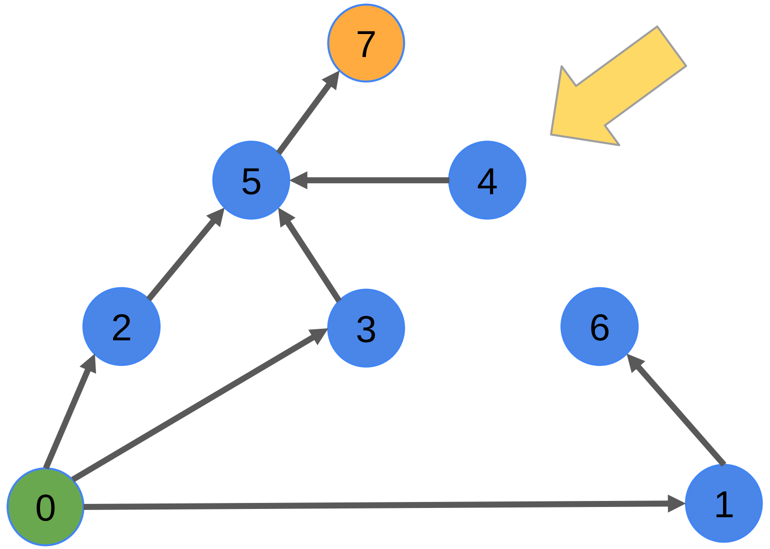
Twice-pruned

Thrice-pruned

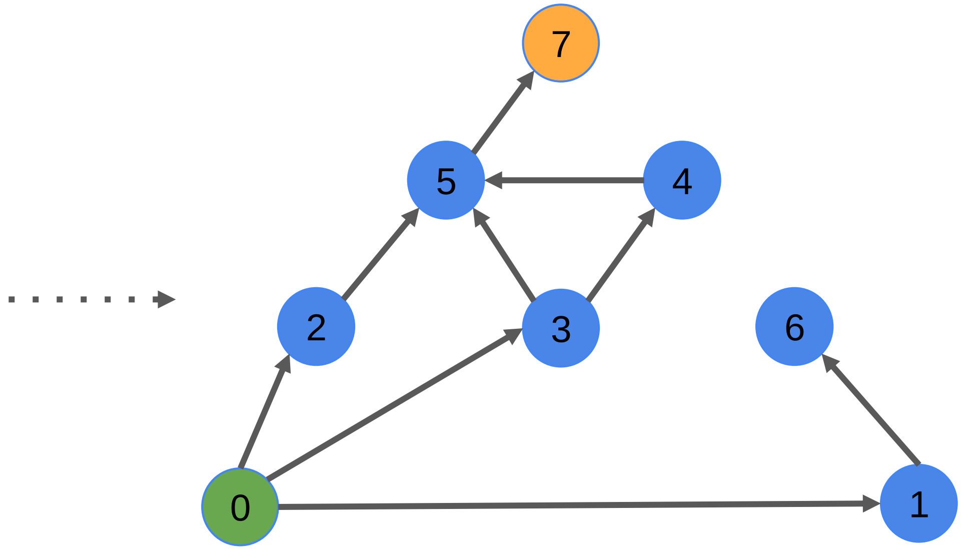
Restructuring

- Add an incoming edge to non-source nodes with no incoming edges
    - Randomly select a predecessor node among the possible lower-valued nodes
- Add an outgoing edges to non-sink nodes with no outgoing edges
    - Randomly select a successor node among the possible higher-valued nodes
- Benefit: resultant graphs retain all of their nodes

Add an incoming edge to nodes with zero incoming

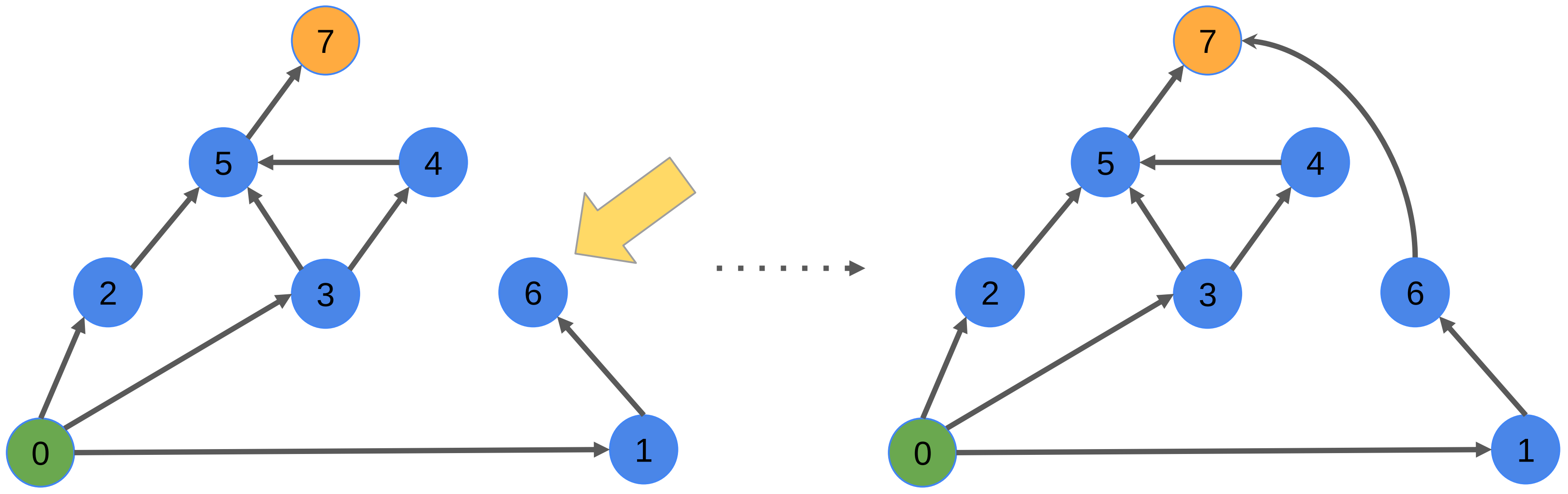


Unaltered



Once-restructured

Add an outgoing edge from nodes with zero outgoing



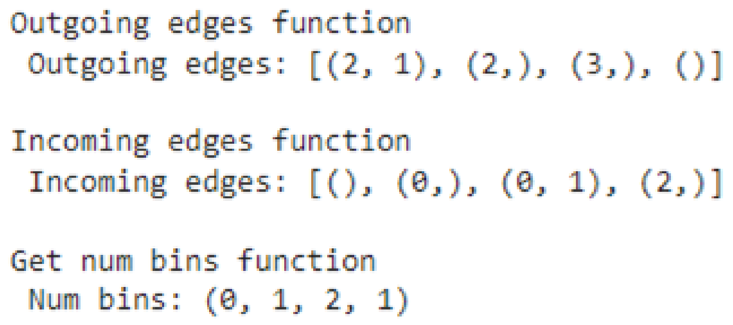
Once-restructured

Twice-restructured

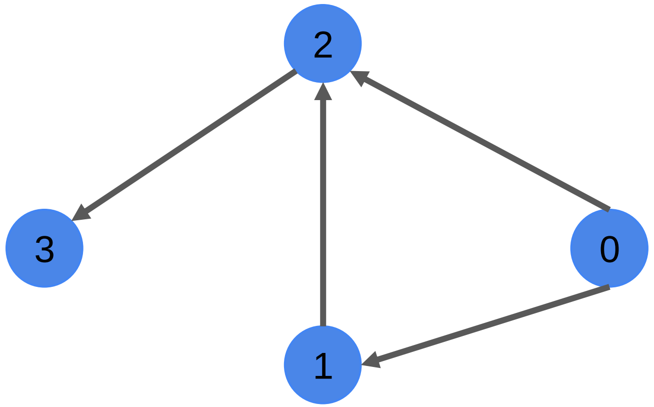
Building Restructured Graphs into SQNs

Functions for Making the Output Map

- “get_outgoing_edges” - returns a list of tuples representing outgoing edges for every node
- “get_incoming_edges” - returns a list of tuples representing incoming edges for every node
  - Required reversing the get_outgoing_edges function, as networkx adjacency matrices are outgoing-focused
- “get_num_bins” - returns number incoming edges for every node



```
Outgoing edges function
 Outgoing edges: [(2, 1), (2,), (3,), ()]

Incoming edges function
 Incoming edges: [(), (0,), (0, 1), (2,)]

Get num bins function
 Num bins: (0, 1, 2, 1)
```

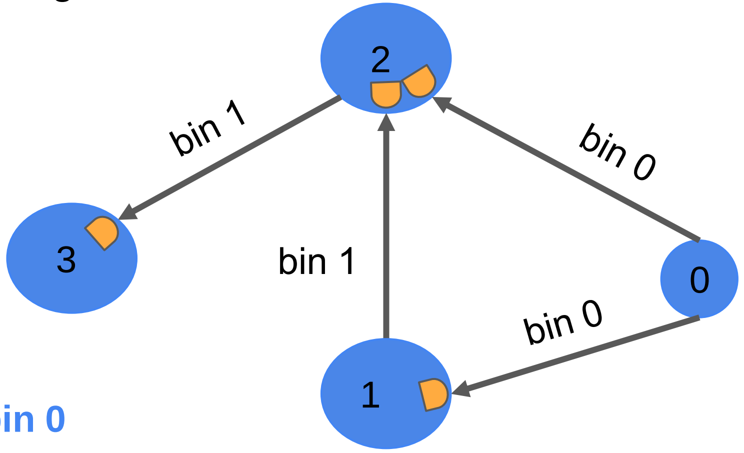


Output Map Structure

- **"Output map"** directly maps, for every output node with outgoing edges, the "bin" label for input nodes connected to that output node
    - This translates into a list, of lists, of tuples.
    - It essentially labels the outgoing edges in order
    - Requires both list of incoming and outgoing edges

```
Outgoing edges function
 Outgoing edges: [(2, 1), (2,), (3,), ()]

Incoming edges function
 Incoming edges: [(), (0,), (0, 1), (2,)]

Get num bins function
 Num bins: (0, 1, 2, 1)

Output map function
 Output map: [[(2, 0), (1, 0)], [(2, 1)], [(3, 0)], []]
```

Node 0 is connected to node 2, bin 0 and node 1, bin 0
Node 1 is connected to node 2, bin 1
Node 2 is connected to node 3, bin 0



Convert the generated graph into an SQN Class

- This SQN follows the GSMP formalization from earlier
  - Clock generation for event timers utilizes an exponential distribution with default beta = 0.5
  - Every state transition can potentially trigger new events
- Event lists were maintained in (event type, absolute event time) through a priority queue using a min heap
  - O(1) fetch time for minimum value
  - Output node selection on event completion is uniformly generated, which is then guided to the correct bin via the output map
- Two stopping conditions: MAX_TIMESTEPS or MAX_TIME
  - If MAX_TIMESTEPS is set then the simulation will stop based on number of iterations, otherwise the default is 500 seconds elapsing

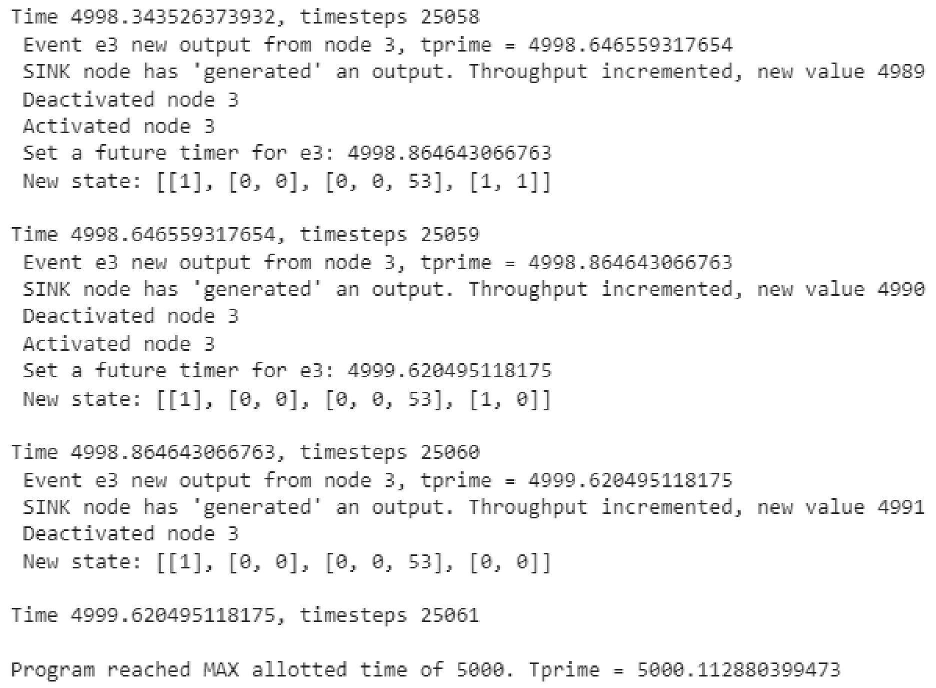
SQN Class Helper Functions

- **“choose_output”** looks at possible incoming edges to an output node and randomly selects one of them using the output_map data structure
- **“check_ready_fire”** checks if a given node has ALL of its input node
- **“decrement_bins”** this function decrements every bin by 1 for a given node
  - It’s called immediately after “check_ready_fire” is true
- **“state_swap”** alters contents of state s’ with those of state s to avoid allocating more memory
- **“state_printout”** prints state/time/timestep for debugging state transitions

Output and Analysis

Output and Cognitive Walkthrough Analysis

- State history with timestamps
- Queue size at every timestamp
    - Requires supplying MAX_TIMESTEPS
- Queue size at termination
- Throughput

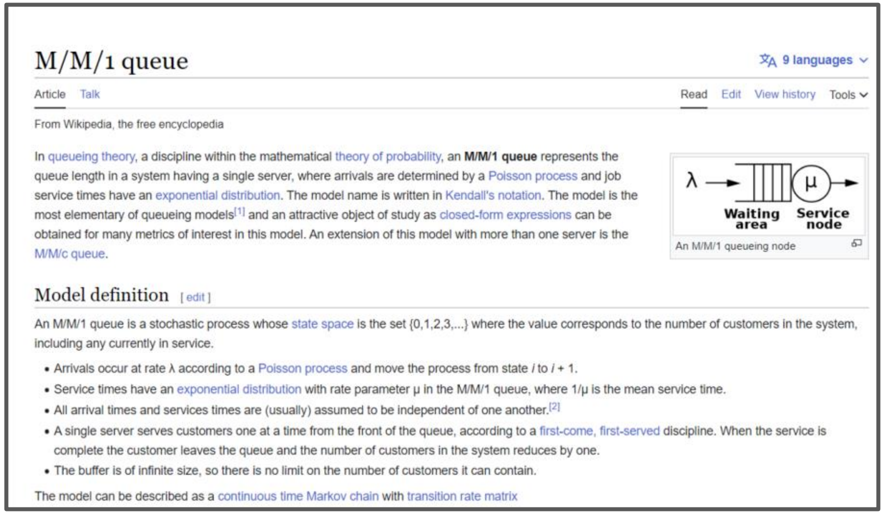


```
Time 4998.343526373932, timesteps 25058
 Event e3 new output from node 3, tprime = 4998.646559317654
 SINK node has 'generated' an output. Throughput incremented, new value 4989
 Deactivated node 3
 Activated node 3
 Set a future timer for e3: 4998.864643066763
 New state: [[1], [0, 0], [0, 0, 53], [1, 1]]

Time 4998.646559317654, timesteps 25059
 Event e3 new output from node 3, tprime = 4998.864643066763
 SINK node has 'generated' an output. Throughput incremented, new value 4990
 Deactivated node 3
 Activated node 3
 Set a future timer for e3: 4999.620495118175
 New state: [[1], [0, 0], [0, 0, 53], [1, 0]]

Time 4998.864643066763, timesteps 25060
 Event e3 new output from node 3, tprime = 4999.620495118175
 SINK node has 'generated' an output. Throughput incremented, new value 4991
 Deactivated node 3
 New state: [[1], [0, 0], [0, 0, 53], [0, 0]]

Time 4999.620495118175, timesteps 25061

Program reached MAX allotted time of 5000. Tprime = 5000.112880399473
```

Sample run with MAX_TIME = 5000

M/M/1 Queue Verification

- A simple queue length model with a single server
- Job service times have exponential distribution
- In the simplest case of the SQN, with n = 2, it is exactly an M/M/1 queue
  - **There is a transient solution for the probability mass function of M/M/1 queues at time t**



M/M/1 queue

From Wikipedia, the free encyclopedia

In queueing theory, a discipline within the mathematical theory of probability, an **M/M/1 queue** represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. The model name is written in Kendall's notation. The model is the most elementary of queueing models[1] and an attractive object of study as closed-form expressions can be obtained for many metrics of interest in this model. An extension of this model with more than one server is the M/M/c queue.

An M/M/1 queueing node

Model definition [edit]

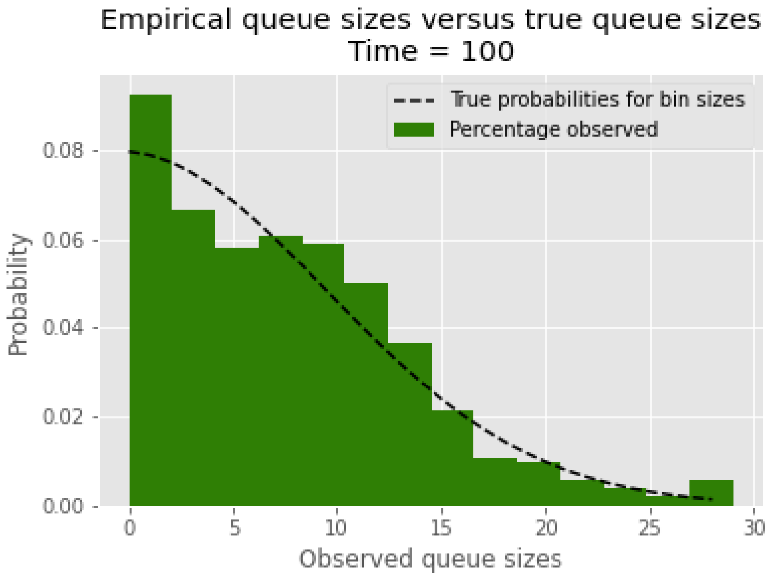
An M/M/1 queue is a stochastic process whose state space is the set {0,1,2,3,...} where the value corresponds to the number of customers in the system, including any currently in service.

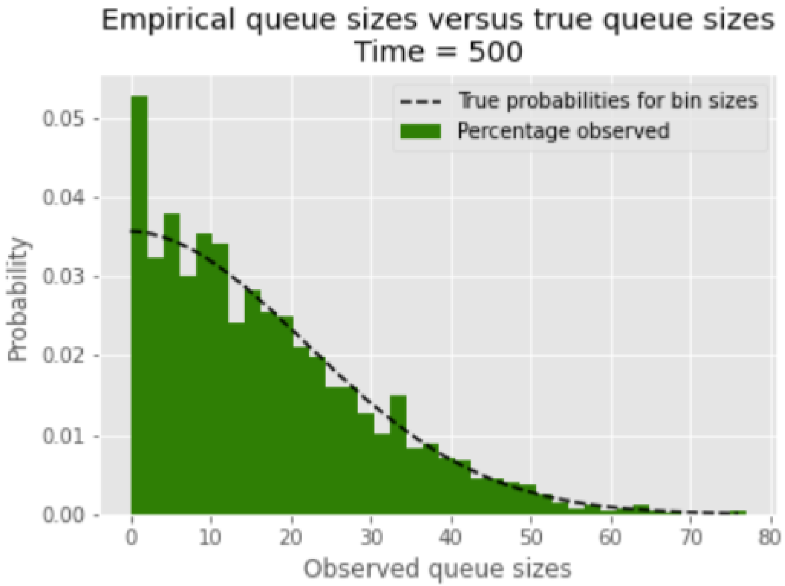
- Arrivals occur at rate λ according to a Poisson process and move the process from state *i* to *i* + 1.
- Service times have an exponential distribution with rate parameter μ in the M/M/1 queue, where 1/μ is the mean service time.
- All arrival times and services times are (usually) assumed to be independent of one another.[2]
- A single server serves customers one at a time from the front of the queue, according to a first-come, first-served discipline. When the service is complete the customer leaves the queue and the number of customers in the system reduces by one.
- The buffer is of infinite size, so there is no limit on the number of customers it can contain.

The model can be described as a continuous time Markov chain with transition rate matrix

$$p_k(t) = e^{-(\lambda+\mu)t}\left[\rho^{\frac{k-i}{2}} I_{k-i}(at) + \rho^{\frac{k-i-1}{2}} I_{k+i+1}(at) + (1-\rho)\rho^k \sum_{j=k+i+2}^{\infty} \rho^{-j/2} I_j(at)\right]$$

Analysis via Closed Form MM1 Verification





Estimator Class

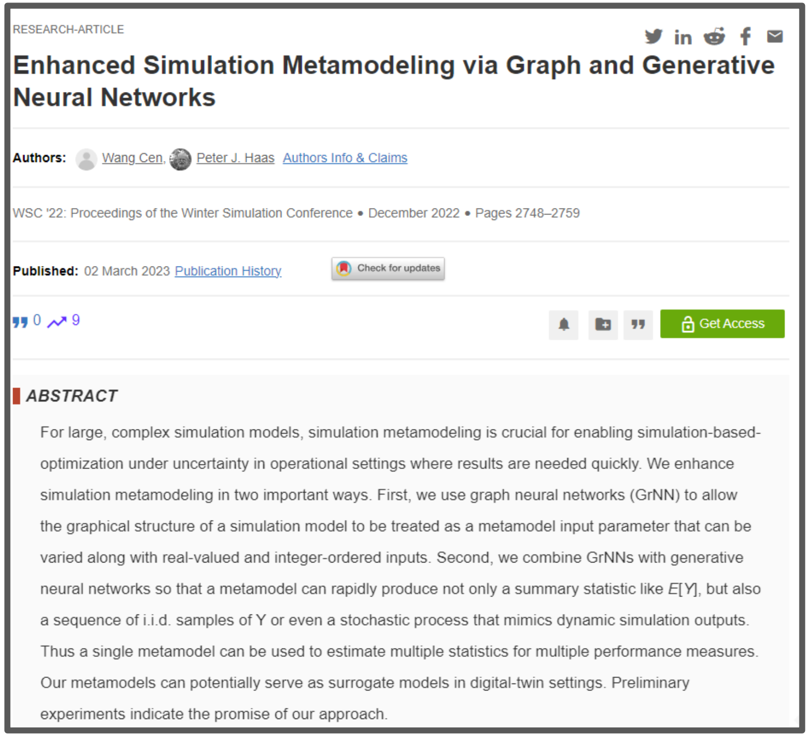
- Stored flagged point estimate of interest at the return of every simulation trial
    - Default point estimate is throughput
- Used point estimate observations to automatically calculate mean, variance, and a 95% confidence interval

Export Trials into JSON Objects

- An export JSON object is compiled dependant on user-flagging
- Trial Flags
  - State History
  - Throughput
- Structural Metadata Flags
  - Output map data structure
  - MAX_BIN integer
  - Incoming edges structure
  - Outgoing edges structure
- The customized export object is then converted into a JSON object and exported as file “output.json”

The Larger Metamodeling Project

- The project this contributes to is based on using graph structures and output (summary statistics, time-series) to train a metamodel capable of generating that data on their own
    - This is done through combining graph neural networks (GrNN) and generative neural networks
- The SQN project and further future analysis will be part of the journal submission to the Winter Simulation Conference



RESEARCH-ARTICLE

Enhanced Simulation Metamodeling via Graph and Generative Neural Networks

Authors: Wang Cen, Peter J. Haas Authors Info & Claims

WSC '22: Proceedings of the Winter Simulation Conference • December 2022 • Pages 2748–2759

Published: 02 March 2023 Publication History

Check for updates

0 9

Get Access

ABSTRACT

For large, complex simulation models, simulation metamodeling is crucial for enabling simulation-based-optimization under uncertainty in operational settings where results are needed quickly. We enhance simulation metamodeling in two important ways. First, we use graph neural networks (GrNN) to allow the graphical structure of a simulation model to be treated as a metamodel input parameter that can be varied along with real-valued and integer-ordered inputs. Second, we combine GrNNs with generative neural networks so that a metamodel can rapidly produce not only a summary statistic like $E[Y]$, but also a sequence of i.i.d. samples of Y or even a stochastic process that mimics dynamic simulation outputs. Thus a single metamodel can be used to estimate multiple statistics for multiple performance measures. Our metamodels can potentially serve as surrogate models in digital-twin settings. Preliminary experiments indicate the promise of our approach.