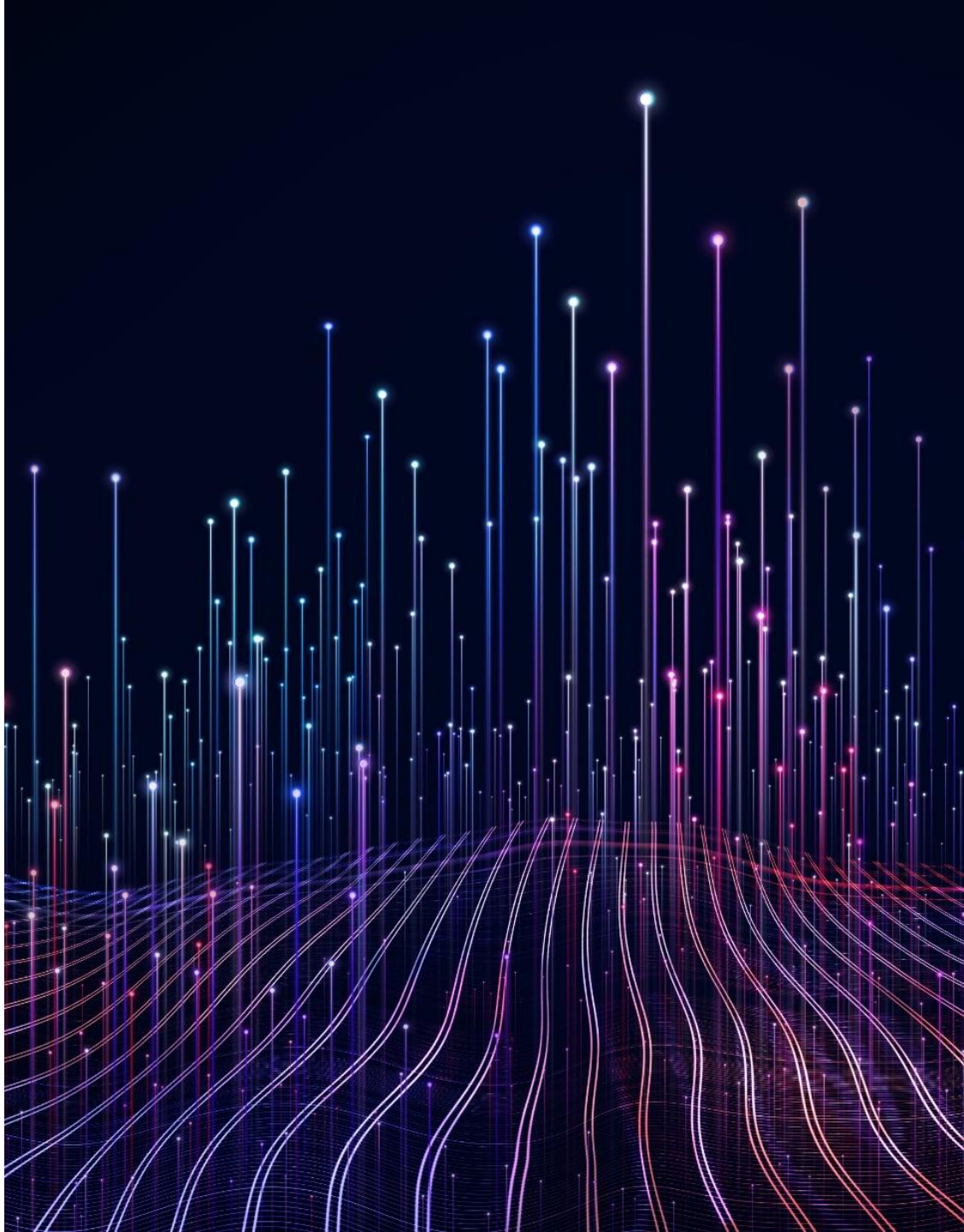
wood.

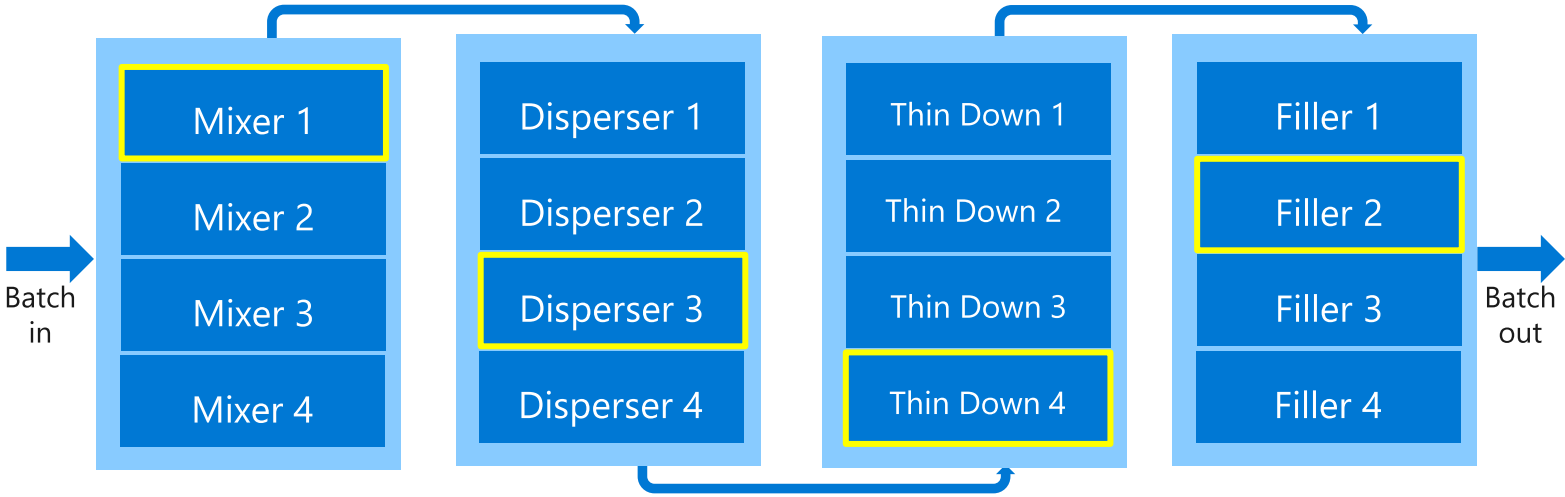
Factory Scheduling Demo



Microsoft

Problem Statement

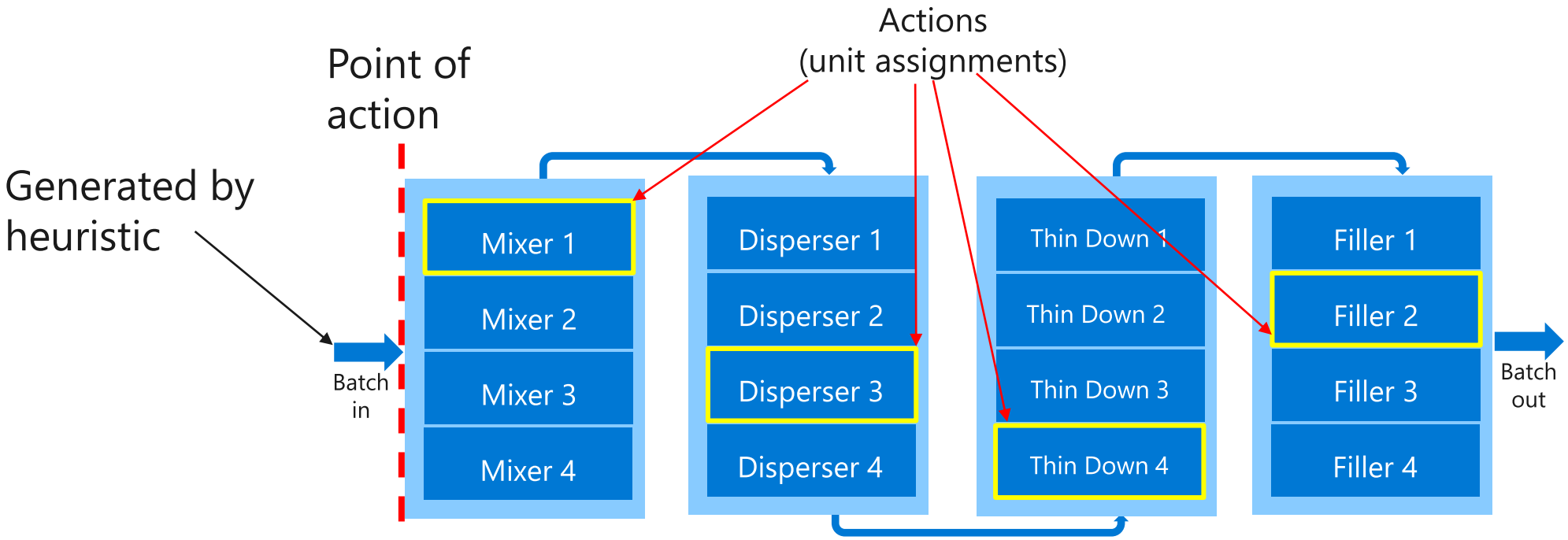
- Schedule operations within a paint factory with the following process stages:
  1. Mixing (4 units each)
  2. Dispersing (4 units each)
  3. Thinning down (4 units each)
  4. Filling (4 units each)
- 6 different products, each with a recipe (set of compatible process units)
- Order list with volumes and delivery dates for each product
- Each unit has a unique capacity and processing sped
- Objectives:
  - Minimize order completion delays



Problem Statement

- Useful definitions
  - *Make-span*
    - Time for a batch to go from one end of process train to the other (start to finish)
  - *Changeover Time*
    - Time required to clean a process unit when switching from one product to another
  - *Order Completion Delay*
    - Time to manufacture the product (to the volumes required in the order) – Time at which the customer needs the product

Framing as an RL problem



States:

- Equipment unit status (availability, assigned products)
- Current batch information (remaining volume, due date)
- Process stats (makespan, delays, changeovers)

Rewards:

- Minimizing estimated delays
- Minimizing estimated make-span

Simulator Overview

- Built in Python using SimPy library (discrete-event-simulation)
- Batch generation heuristic
  - Priority heuristic based on order volumes and due dates
- Once all batches have been assigned simulation skips to the end of manufacturing process
  - Estimated makespan
  - Estimated delays
  - Estimated changeovers
- After simulation, a machine wise schedule and an product manufacturing summary is generated
- Fast run time (few seconds)

```python
for iteration in count(1):
    print(f"Running iteration #{iteration}")
    # only take bonsai actions when there is an order
    if sim.next_order is not None:
        sim_state = sim.get_state()
        if args.brain:
            action = run_exported_brain(sim_state)
        else:
            action = random_policy(sim_state)
        sim.episode_step(action)
    # otherwise keep progressing the simulator
    else:
        sim.env.run()
    # if all products have been manufactured, finish the simulation
    if sim.complete:
        sim_state = sim.get_state()
        print("Simulation complete")
        break
```

Order summary

	Volume Produced [Gallons]	# Batches [-]	Delay [days]
0	1000.0	1	-1.6
1	12500.0	10	-2.2
2	7500.0	8	-3.9
3	5000.0	5	-3.0
4	15000.0	8	-0.4
5	10000.0	10	-3.3
6	2500.0	1	-1.3
7	10000.0	10	-3.2
8	1000.0	1	-2.0
9	7500.0	8	-2.1
10	10000.0	10	-2.4
11	5000.0	1	-1.7
12	1000.0	1	-2.1
13	12500.0	13	-2.4
14	15000.0	15	-1.4
15	15000.0	11	-0.2
16	1000.0	1	-1.0
17	1000.0	1	-2.5
18	15000.0	4	0.1
19	5000.0	5	-3.5

Total number of changeovers: 189

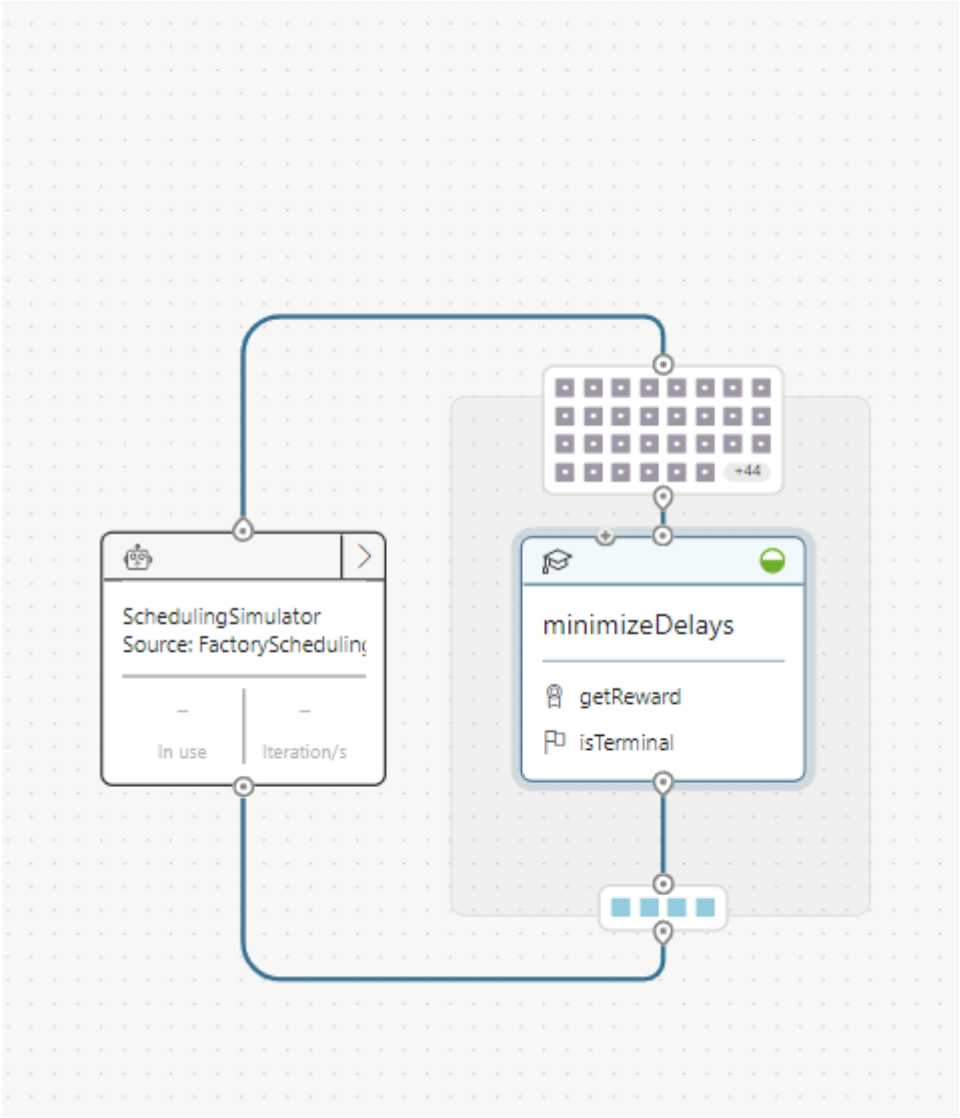
Machine wise schedules were as follows

unit_type	unit_name	product_name	start_time	end_time	utilization	processed_volume
mixer	e1	a	48.5	51.0	50.0	1000.0
		e	28.0	30.5	100.0	2000.0
		e	32.0	34.0	50.0	1000.0
		f	88.0	90.5	50.0	1000.0
		f	92.0	94.0	50.0	1000.0
		f	96.0	98.0	50.0	1000.0
		f	100.0	102.0	50.0	1000.0
		f	104.0	106.0	50.0	1000.0
		f	108.0	110.0	50.0	1000.0
		h	152.0	154.5	50.0	1000.0
		h	156.0	158.0	50.0	1000.0
		h	160.0	162.0	50.0	1000.0
		h	164.0	166.0	50.0	1000.0
		h	168.0	170.0	50.0	1000.0
		k	130.0	132.5	50.0	1000.0
		k	136.0	138.0	50.0	1000.0
		m	54.0	56.5	50.0	1000.0
		p	8.0	10.0	100.0	2000.0
		p	14.0	16.0	50.0	1000.0
		p	16.0	18.0	50.0	1000.0
		p	20.0	22.0	50.0	1000.0
		q	46.0	48.5	50.0	1000.0
		s	38.0	40.5	100.0	2000.0
		s	40.5	42.5	100.0	2000.0
		s	44.0	46.0	50.0	1000.0

Brain Training

- States:
  - Current product information
  - Unit information
  - Manufacturing statistics:
    - Changeovers, Makespan and delays
- Actions:
  - Unit number for each process stage

```
type SimAction {
    dispersion: number<e5 = 1, e6 = 2, e7 = 3, e8 = 4>,
    filler: number<e13 = 1, e14 = 2, e15 = 3, e16 = 4>,
    mixer: number<e1 = 1, e2 = 2, e3 = 3, e4 = 4>,
    thindown: number<e9 = 1, e10 = 2, e11 = 3, e12 = 4>,
}
```



Action Masking

- Each product has compatible equipment units
- Defined in Inkling file as a mask function

```
if s.current_product == 1 {
    return constraint SimAction {
        dispersion: number<in [2]>,
        filler: number<in [1, 2, 3, 4]>,
        mixer: number<in [1]>,
        thindown: number<in [1, 2, 4]>,
    }
```

Brain Training Notes

- In the published sample, the order list, manufacturing recipes and equipment limitations are fixed
- A reward + terminal function formulation is required


wood.