From Chaos to Clarity: Modelling ED Processes with simPy

Jinkett A. Yee <subscriptions@medium.com>
Reply-To: jinkett.yee.2022@mitb.smu.edu.sg
To: jinkett99@gmail.com
Mon, Jul 7, 2025 at 6:20 PM

From Chaos to Clarity: Modelling ED Processes with simPy

Jinkett A. Yee · July 8, 2025 · 14 min read · View on Medium

> *Imagine a dangerous scenario where patients are waiting for over two hours in a hospital emergency department. Or when company staff go on strike over absolutely nothing (or is it really nothing?)*

Image generated by DALL.E GPT using author's prompt

A Gentle Introduction to Operation Studies and Process Simulation

Imagine an Emergency Department (ED)—a bustling, chaotic environment filled with patients in need, nurses hurrying across stations, and doctors working under constant pressure.

As part of the operations research team, you're tasked with a critical mission—analysing the system's performance. *How is the patient experience? Are departments adequately staffed? Is the system design optimal?*

These are complex, multi-layered questions—too intricate to answer through surveys or (surprisingly) even traditional statistical analysis alone.

> *There comes our saviour, process simulation—A practical approach where we will build our very own process-driven workflow and supplement it with domain insights to accurately model our real-world system (in our case, ED operations).*
>
> *This lets us simulate, evaluate, and improve how the Emergency Department functions —without disrupting the actual hospital floor.*

Process Simulation with simPy

How exactly should we model, simulate, evaluate and gather insights on how we can design our ED system? It is actually made simple with simPy!

> *We shall turn to simPy—an open source, lightweight, discrete event simulation framework written in pure Python—With the ability to integrate with real-time data, dashboards and Machine learning workflows.*

Yes, that means it is especially coherent to model our ER system with simPy, where we can declare discrete events (like consultations, lab-tests etc.) and arrange them one after another in a time-dependent manner.

Next, we can simply (no punt intended) write up our entities, resources, and events in Python to set up our simulation environment.

> *I know what you're thinking, sounds simple right? It actually is—The next sections will discuss more on how to implement this in code—step by step.*

Step 1: Define our System Components

*Patients as **entities**—Doctors, Nurses & Hospital Beds as **limited resources**.* That's it! If you want a comprehensive template, here's the full table:

ED System Components

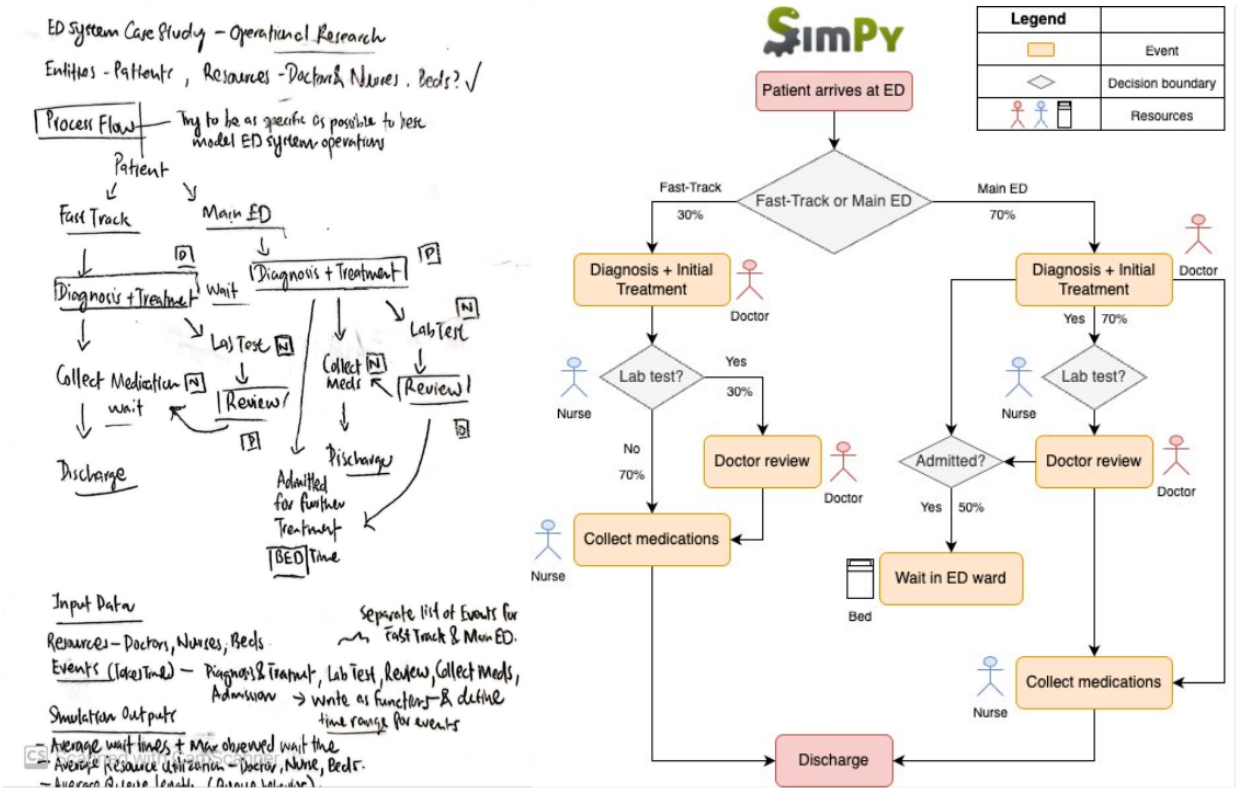
Component	Description
Entities	Fast-Track patients, Main ED patients
Resources	Doctors, Nurses
Events	Patient check-in, Diagnosis & treatment, Lab test, Doctor reviews test results, Collect medications, Admitted for further treatment, Patient discharge
State Variables	Wait times, Maximum wait time, Queue length, Doctor/Nurse/Bed utilisation percentages

MD Table for System Components of ED operations. Table generated by Author.

> ***P.S.** I find it much easier to first define entities and resources, before drawing out the **Process Flow Chart** (visualisation of events) and finally, planning out the state variables for output tracking for actionable insights.*

Step 2: [The most important step] The Process Flow Chart

In my humble opinion, the Process Flow Chart is the crux of process simulation. *Once you start drawing, you'll complete your codes for sure!*



The Golden Snitch for Process Simulation—Process Flow Chart (Drawing: Left, Full Chart: Right)

Notice the definitions of **stateful events** and **decision boundaries** in the Process Flow Chart. In my Hospital ED process, I detailed a **fast-track** route for patients who are not very sick—patients' needs can be met quickly with a low probability of lab tests needed and a certainty of discharge.

Patients' with more serious needs would be routed to the **Main ED**—These patients will have a higher probability of undergoing more lab tests and has a chance of being admitted for further treatment and observation. If they are admitted, they would require a bed in the ED until they are transferred to an appropriate hospital department.

> ***P.S.*** *When drawing your Process Flow Chart, it is very crucial to be as detailed as possible with each event and decision boundary in order to effectively represent the structure of your system and operations.*

Step 3: Defining Input Data Required for Simulation

Here's a breakdown of input parameters required for our simulation:

1. Number of available resources—*Doctors, Nurses, Beds*
2. Number of patients—*144 patients*

3. Simulation time—*1 full day = 1440 minutes*
4. Patients' inter-arrival times
5. Idle wait times for events
6. Probabilities for decision boundaries—*Fast-track vs Main ED? Lab test required? Is the patient admitted?*

> ***P.S.*** *We will talk more about the numbers and their derivation when we discuss the code implementations.*

Step 4: Defining our Simulation Outputs

This is yet another crucial step as the outputs from our simulations defines the **actionable insights** that we process.

> ***I.e.*** *Based on the simulations of patient flow and resource allocation: How can we* ***evaluate our system's performance****? How do we* ***better design*** *system capacity? Etc.*

In context of our Hospital ED operations, it is crucial that triaged patients are given appropriate treatment in a **timely and efficient manner**.

Let us define the following metrics to track actionable insights:

1. Average Patient Wait Time (mins)
2. Maximum Patient Wait Time (mins)
3. Average Queue Length (mins)—for Fast-Track and Main ED patients
4. Utilisation Metrics (%)—Doctors, Nurses, Beds (Resources)

> ***P.S.*** *Wait times and queue times give an indication of* ***treatment timeliness and patient experience*** *while resource utilisation rates track* ***system efficiency*** *(i.e. whether our resources are overcooked). We will also plot out queue times and resource utilisation percentages across the simulation timeline.*

Code Implementation: Simulating our ER System

Finally! If you have made it this far, you'll make it to the end!

> *To be very honest, this is the very first time I've been exposed to, and experimented with the simPy simulation library—Full credits to their **"simPy in 10 minutes"** documentation, I quickly grasp concepts and abstractions. Finally, I have this article from Real Python to thank when I was scrambling for cookbooks to build up my very own ED process flow.*

Python Class to store Resources and Processes/Events

To scaffold our simulation environment, we have to first build a blueprint for our system—I.e. to create an ED environment. We do this by initialising a Hospital Class Definition, where we will add our resources as attributes and events as class methods. *Recall our Process Flow Chart definition earlier.*

We utilise **`simpy.Resource(env, number)`** to declare how many resources (Doctors, Nurses or Beds) can be available in the environment at any given time. **`self.env.timeout(service_time)`** tells simPy to trigger a timeout (in minutes) during the event sequence.

We decided to model wait times with the Normal Probability Distribution—Under the assumption that population ED wait times follow a normal distribution, with their respective means and standard deviations. I've attached the simple justifications in my *GitHub Repo folder*.

> ***P.S.** We define a separate resource (i.e. Doctors & Nurses) for each sub-system—Fast-Track and Main ED, which allows us to track the output state variables across our sub-systems.*

```
# Create Hospital class to store resources and events/processes
class Hospital(object):
    def __init__(self, env, num_fast_doctors, num_fast_nurses,
num_ed_doctors, num_ed_nurses, num_beds):
        self.env = env
```

```
        self.fast_doctor = simpy.Resource(env, num_fast_doctors)
        self.fast_nurse = simpy.Resource(env, num_fast_nurses)
        self.ed_doctor = simpy.Resource(env, num_ed_doctors)
        self.ed_nurse = simpy.Resource(env, num_ed_nurses)
        self.beds = simpy.Resource(env, num_beds)

    # consulting a doctor
    def consult(self, patient):
        service_time = np.random.normal(20, 5)
        service_time = max(5, service_time) # no negative
durations
        yield self.env.timeout(service_time)

    # wait for medications
    def medication(self, patient):
        service_time = np.random.normal(15, 3)
        service_time = max(5, service_time)
        yield self.env.timeout(service_time)

    # wait for nurse to perform lab-test (fast-track) - Involves
nurse
    def fast_lab(self, patient):
        service_time = np.random.normal(6, 3)
        service_time = max(1, service_time)
        yield self.env.timeout(service_time)

    # wait for lab-test (fast-track) - No resource used.
    def fast_lab_wait(self, patient):
        service_time = np.random.normal(25, 5)
        service_time = max(10, service_time)
        yield self.env.timeout(service_time)

     # wait for lab-test (main ED) - Involves nurse
    def ed_lab(self, patient):
        service_time = np.random.normal(10, 4)
        service_time = max(3, service_time)
        yield self.env.timeout(service_time)

    # wait for lab-test (main ED) - No resource used.
    def ed_lab_wait(self, patient):
        service_time = np.random.normal(40, 10)
        service_time = max(15, service_time)
        yield self.env.timeout(service_time)

    # review - after taking lab-test
    def review(self, patient):
        service_time = np.random.normal(10, 3)
        service_time = max(3, service_time)
        yield self.env.timeout(service_time)

    # re-admission to other dept for further treatment
```

```
    def admit(self, patient):
        service_time = np.random.normal(30, 5)
        service_time = max(5, service_time)
        yield self.env.timeout(service_time)
```

Next up, we create lists to store our output state variables—Wait time, Queue length and Utilisations. *Note: We could also use a default dict to simplify things.*

```
## Store variables - Queue tracking
queue_fast, queue_ed = [], []

## Store wait times & timeline (for plots)
wait_times, timeline = [], []

## Utilizations
util_fast_doc = []
util_fast_nurse = []
util_ed_doc = []
util_ed_nurse = []
util_beds = []
```

Function for Patient Events

The next critical code block illustrates our ED Process Flow. In this code block, we **define a patient function** for our ED patients to step through events in a time-dependent manner, utilising respective resources.

Let me explain this in detail, using the first example.

Since Doctors are a **shared resource**, we will need to include a waiting behaviour when a patient consults him. Here is the breakdown:

1. **`hospital.Doctor.request()`** —Generate a request to consult a Doctor.
2. **`yield request`** — Patient waits for the Doctor to become available.
3. **`yield env.process(hospital.consult(patient)`** —Patient uses an available doctor to complete the consultation, calling the **`hospital.consult(patient)`** function to integrate timeout.

`yield` pauses the simulation until the request or process is completed before moving on to the next event.

You might also notice decision boundaries involving probabilities. These boundaries are defined using **`random.random() < decision_probability` —**For example, we assume that majority of ED patients (70%) will be triaged to main ED for a closer observation. More details on decision probabilities can be found in my *GitHub Repo folder*.

> ***P.S.** Notice that we are keeping track of the total wait time state variable using **`env.now`** and a **`wait_times`** list—to be collected in output metrics.*

```
# Define patient process - Refer to process flow above.
def patient(env, patient, hospital):
    arrival_time = env.now
    total_time = 0 #initialize to track total waiting time
    is_fast_track = random.random() < 0.3  # 30% go to fast-track

    if is_fast_track:
        arrival_time = env.now
        # Step 1: consult doctor
        with hospital.fast_doctor.request() as request:
            yield request
            total_time += env.now - arrival_time
            yield env.process(hospital.consult(patient))

        # Step 2: lab test - 30% chance
        arrival_time = env.now
        if random.random() < 0.3:
            with hospital.fast_nurse.request() as request:
                yield request
                total_time += env.now - arrival_time
                yield env.process(hospital.fast_lab(patient))
            ## patient waits alone - Ignore in wait time
computation
            yield env.process(hospital.fast_lab_wait(patient))
            arrival_time = env.now
            # doctor's review
            with hospital.fast_doctor.request() as request:
                yield request
                total_time += env.now - arrival_time
                yield env.process(hospital.review(patient))

        # Step 3: medication
        arrival_time = env.now
```

```
            with hospital.fast_nurse.request() as request:
                yield request
                total_time += env.now - arrival_time
                yield env.process(hospital.medication(patient))

        else:
            # Step 1: consult ED doctor
            arrival_time = env.now
            with hospital.ed_doctor.request() as request:
                yield request
                total_time += env.now - arrival_time
                yield env.process(hospital.consult(patient))

            # Step 2: lab test - 70% chance
            if random.random() < 0.7:
                arrival_time = env.now
                with hospital.ed_nurse.request() as request:
                    yield request
                    total_time += env.now - arrival_time
                    yield env.process(hospital.ed_lab(patient))
                ## patient waits alone - Ignore in wait time
computation
                yield env.process(hospital.ed_lab_wait(patient))
                arrival_time = env.now
                # doctor's review
                arrival_time = env.now
                with hospital.ed_doctor.request() as request:
                    yield request
                    total_time += env.now - arrival_time
                    yield env.process(hospital.review(patient))

            # Step 3: admission OR medication
            if random.random() < 0.5:
                arrival_time = env.now
                with hospital.beds.request() as bed_req:
                    yield bed_req
                    total_time += env.now - arrival_time
                    yield env.process(hospital.admit(patient))
            else:
                arrival_time = env.now
                with hospital.ed_nurse.request() as nurse_req:
                    yield nurse_req
                    total_time += env.now - arrival_time
                    yield env.process(hospital.medication(patient))

        wait_times.append(total_time)
```

Function for Tracking Simulation Outputs—Evaluation Metrics

> *At last, let us write a function to collect our **simulation outputs/metrics** from the patient process.*

Thankfully, simPy makes it super easy for us to track our state variables at progressive time points of the patient process. Here, we define a monitor function to track state variables at every interval of 5 minutes.

The `simpy.Resource` instances allow us to extract attributes like queue length, count and capacity. We can access these attributes directly from each resource. For instance, `Resource.queue` returns the list of patients waiting.

Note that **resource utilisation at each time point** is computed by taking `Resource.count` (representing count of utilised resources) divide by `Resource.capacity` (representing total capacity of resources).

```
# Monitor function
def monitor(env, hospital, timeline,
            queue_fast, queue_ed,
            util_fast_doc, util_fast_nurse,
            util_ed_doc, util_ed_nurse,
            util_beds,
            interval=5):
    while True:
        # Time snapshot
        timeline.append(env.now)

        # Queue lengths
        q_fast = len(hospital.fast_doctor.queue) +
len(hospital.fast_nurse.queue)
        q_ed = len(hospital.ed_doctor.queue) +
len(hospital.ed_nurse.queue)
        queue_fast.append(q_fast)
        queue_ed.append(q_ed)

        # Utilizations - .count = how many patients are currently
served..capacity = total number.
        util_fast_doc.append(hospital.fast_doctor.count /
hospital.fast_doctor.capacity)
        util_fast_nurse.append(hospital.fast_nurse.count /
hospital.fast_nurse.capacity)
        util_ed_doc.append(hospital.ed_doctor.count /
hospital.ed_doctor.capacity)
        util_ed_nurse.append(hospital.ed_nurse.count /
hospital.ed_nurse.capacity)
        util_beds.append(hospital.beds.count /
```

```
    hospital.beds.capacity)

            yield env.timeout(interval)
```

Putting everything together—Running our Simulation

> *Finally, the moment we've been waiting for! It is now time to run a simulation to model our ED system, collecting actionable insights along the way~*

Main script set-up and defined steps:

> *Note that we have included a runnable script (main.py) in our GitHub repository*

1. **Set input variables**—arrival rate, number of patients & simulation time. *We selected a full-day simulation (1440 mins), with patients arriving at 10 minute intervals (on average), giving us a total of 144 patients.*
2. **Resource configuration**—*basic ED set-up with 4 doctors and 3 nurses in the main ED, which anticipates greater patient flow.*
3. **Instantiate `simpy.Environment()`** *which will move the simulation through each time step.*
4. **Instantiate Hospital class**—with resource configurations.
5. **Schedule patients' arrival**—*we define a* **`source`** *function here, where we will run* **`env.process(patient_process)`** *which will tell simPy to schedule the patient events in the current environment*
6. **Schedule `monitor` function**
7. **Call `env.run(until=SIM_TIME)`** *to run the simulation within the defined time range.*

> ***P.S. Arrival Rate (or mean inter-arrival time) of 10 minutes** is set in* **`source`** *to model patients' inter-arrival times with an exponential probability distribution. Which is the probability distribution of time between events in a Poisson point process.*

> *Also notice that we included a* ***`report`*** *function (not shown) to print and visualise our simulation outputs as evaluation metrics.*

```
# Set input variables
ARRIVAL_RATE = 10 # mean inter-arrival time
N_PATIENTS = 144
SIM_TIME = 1440 # mins - 24 hours

# Set configurations
config = {
            "fast_doctors": 1,
            "fast_nurses": 1,
            "ed_doctors": 4,
            "ed_nurses": 3,
            "beds": 5,
        }

# Set random seed for reproducibility
random.seed(42)
np.random.seed(42)

# --- Run Simulation ---
def run_simulation(config, n_patients, sim_time, arrival_rate):
    env = simpy.Environment()
    hospital = Hospital(
    env,
    config["fast_doctors"],
    config["fast_nurses"],
    config["ed_doctors"],
    config["ed_nurses"],
    config["beds"]
)

    # Schedule patient arrivals
    def source(env, Hospital, n_patients):
        for i in range(n_patients):
            env.process(patient(env, f"Patient {i+1}", hospital))
            yield env.timeout(np.random.exponential(arrival_rate))

    env.process(source(env, hospital, N_PATIENTS))
    env.process(monitor(env, hospital, timeline,
                        queue_fast, queue_ed,
                        util_fast_doc, util_fast_nurse,
                        util_ed_doc, util_ed_nurse,
                        util_beds))
    env.run(until=SIM_TIME)

    # --- Report ---
```

```
        report(wait_times, timeline,
            queue_fast, queue_ed,
            util_fast_doc, util_fast_nurse,
            util_ed_doc, util_ed_nurse,
            util_beds)

        # revert

    # Run simulation.
    run_simulation(config, N_PATIENTS, SIM_TIME, ARRIVAL_RATE)
```

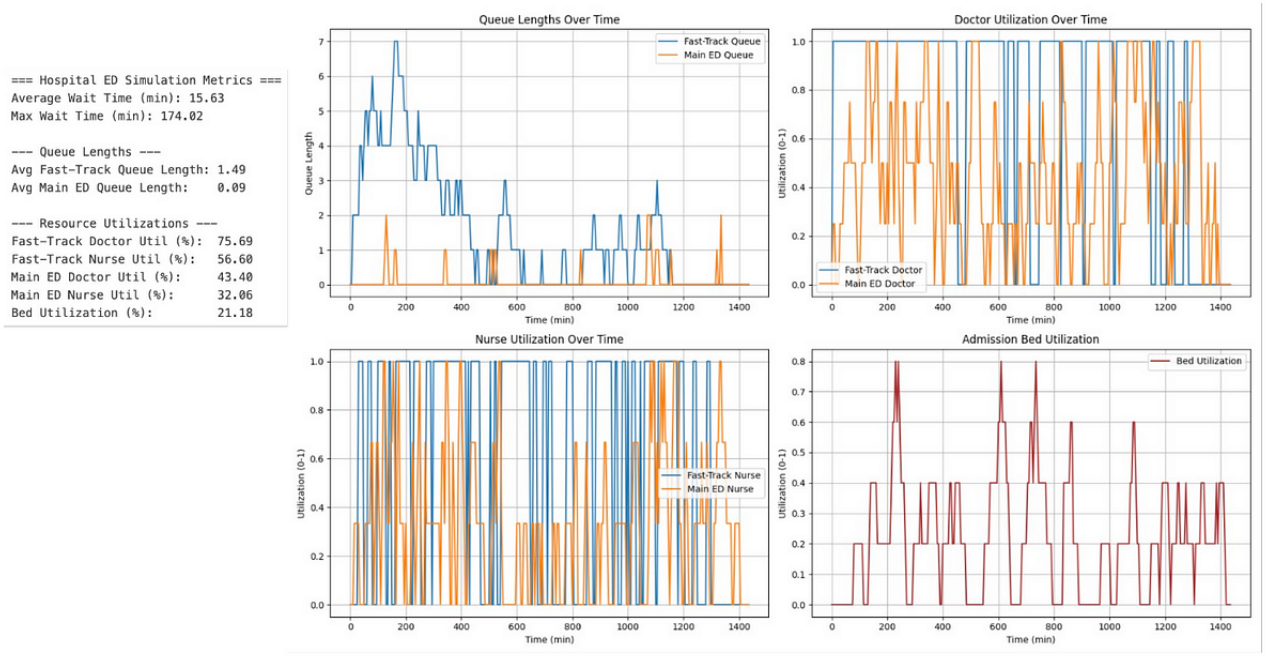
> *Run, run, run away~*

Results & Discussion: Putting Together our Insights

While our defined process flow of ED operations includes an initial triage to split patients into respective treatment groups (Fast-Track and Main ED), it is still critical to evaluate the system's performance on different loads. As discussed earlier, we'll evaluate our system based on the following metrics:

1. **Treatment timeliness** and patient experience
2. **System efficiency**—via Resource utilisation

This simulation provides actionable insights on the performance of our hospital ED operations staffed by **[1 doctor, 1 nurse] in Fast-Track** and **[4 doctors, 3 nurses, 5 beds] in the Main ED treatment route** and serving **144 patients** over a simulated period of **1440 minutes (1 full day)**.



System Performance under Normal Patient Load. Charts generated by Author.

> *Interesting results? Is the system resilient to the current patient load?*

Wait Times & Queue Behaviour

Average wait times are reasonable, **at around 15 minutes on average**. However, there was a patient waiting for almost 3 hours—174 minutes!

On top of going through long consults, reviews and admissions, it was also likely that arrivals happened faster than usual—leading to queues, with an accumulation of longer process times for earlier arrivals. *Unlucky patient!*

Queue lengths were initially **high—up to 7 for the fast-track route** but eventually dispersed while the **main ED sees virtually negligible queues**. In the main ED, we see occasional brief peaks of up to 2 in queue but it isn't frequent or sustained—Indicating timely treatments to patients in need.

> ***Insights:*** *Perform a review on processes on the main ED route to reduce wait time for anomalies—I.e. unlucky patients like the one described above! The queues at the fast-track route could also accumulate at times—ED should consider hiring more staff to the fast-track department.*

Resource Utilisation

Is the ED adequately staffed and equipped?

Visualising resource utilisation percentages over time is an effective way to derive the ED system's **resilience levels** at current patient loads. For instance, frequent and sustained high utilisation rates (above 90%)—signals a risk of staff burnout or system collapse. Conversely, low utilisation rate (below 20%) can indicate wasted system capacity.

In our fast-track department, our **doctor utilisation rate averaged 75% while nurses 56%**—suggesting that staff are engaged and reasonably busy, but not overburdened. The same could be said for the main ED department—with utilisation rates of **43% and 32% respectively**.

> ***Insights:*** *These insights point to* ***system resilience*** *to current loads—I.e. Our ED can handle unexpected spikes in patient inflow without breaking down or compromising on wait times or queue lengths.*
>
> *In summary—The system seems healthy at current loads,* ***wait times were reasonable (except anomaly), short queue lengths and good staffing****. However, the ability of our system to respond to higher/peak patient loads still remains to be tested.*
>
> *As future work, we could test the system on higher patient loads, compromised conditions like staff on sick leaves or even time-dependent arrival patterns (by tweaking the model for inter-arrival times).*
>
> ***P.S.*** *I did a stress-test with mean inter-arrival time reduced to 5 minutes and the system broke! We might need to redesign the system processes and/or reallocate resources.*

Additional Insights—System Optimisation

> *After evaluating our system performance, our next natural progression is to determine an optimal allocation of resources and (possibly) system design.*
>
> *Ahh! It is disappointing that simPy does not inherently include build-in optimisation algorithms to optimise system parameters. How then?*

In this section, we attempt to implement a **simple search strategy—Random Search**, to derive an optimal/sub-optimal resource configuration with the objective function of minimising average wait times. *I.e. If we want to minimise wait times, what is an ideal allocation of doctors, nurses and beds?*

In the attached code block, we run 10 simulations with the same ED process flow—with 10 random configurations of doctors, nurses and hospital beds using **`random.randint(min, max)`** to generate random no.s.

```
import pandas as pd
def optimize_ed_resources(n_iter=10):
    results = []

    for i in range(n_iter):
        config = {
            "fast_doctors": random.randint(1, 3),
            "fast_nurses": random.randint(1, 3),
            "ed_doctors": random.randint(3, 8),
            "ed_nurses": random.randint(3, 8),
            "beds": random.randint(3, 10),
        }

        run_simulation(config, N_PATIENTS, SIM_TIME, ARRIVAL_RATE)
# how to reset input variables after each simulation? Fix*
        avg_wait = np.mean(wait_times)
        results.append({**config, "avg_wait_time": avg_wait})

    df = pd.DataFrame(results).sort_values(by="avg_wait_time")
    return df

# print results
df_results = optimize_ed_resources(n_iter=10)
df_results
```

> *Here are the results.*

	fast_doctors	fast_nurses	ed_doctors	ed_nurses	beds	avg_wait_time
3	2	3	7	5	10	1.684672
6	2	2	7	6	6	2.303828
0	2	3	6	5	9	4.465567
9	3	2	5	7	4	6.058555
4	3	3	4	8	8	8.001794
7	2	2	5	3	8	11.818001
2	2	2	4	7	4	24.051424
5	1	1	8	4	8	55.069734
1	1	3	7	4	5	69.937253
8	3	3	4	8	10	87.124314

Results from Random Search over 10 configurations of resources (doctors, nurses & beds).

Food for thought: Even though we managed to reduce the average wait time drastically, do we really need 10 beds?

Conclusions and Future Works

> *Our ED operations still isn't, and might never be—perfect. But we've definitely learnt something, haven't we?*

Process modelling and simulation is a field with overlaps with traditional data analytics workflows—playing a crucial role for data science practitioners to understand process flow, design domain-backed experiments, evaluate complex systems and re-design them.

This project walks us through the intricacies of simulating a hospital ED operation with simPy, *fast!*—from **thought process** to **understanding mid-level code implementations** to **drawing insights on system performance** and a **simple random search algorithm** to derive potentially optimal resource configurations.

> *However, this is probably—only the tip of the iceberg. There are always more, improvements to be made, and more experimentations to improve our process flow and simulation accuracy.*

Here are a few key areas for future iterations—*Our future roadmap!*

1. We made **many assumptions** from number of patients to inter-arrival times to event wait times in our process flow—It's time to back them up with real domain knowledge and data from actual ED operations.

2. We skipped the **stress-test** this time—A critical step to comprehensively evaluate our system's resilience to stress—from high traffic of patients at certain times to red-tail days where resources were unexpectedly compromised.

3. Deriving an optimal system capacity—*How can we optimise resource configurations? I'll need to read up on this!*

> *Until next time!* 👋✨

Respond to this story View story
