
Algorithms and Analysis

COSC2123/3119

Graph Algorithms in Action: Modeling a Disease Outbreak

Assessment Type	Individual assignment. Submit online via GitHub.
Due Date	Week 11, Thursday May 21, 13:00 (1pm). A late penalty will apply to assessments submitted after 13:00.
Marks	30

1 Learning Outcomes

This assessment relates to four learning outcomes of the course which are:

- CLO 1: Compare, contrast, and apply the key algorithmic design paradigms: brute force, divide and conquer, decrease and conquer, transform and conquer, greedy, dynamic programming and iterative improvement;
- CLO2: Compare, contrast, and apply key data structures: trees, lists, stacks, queues, hash tables and graph representations;
- CLO 3: Define, compare, analyse, and solve general algorithmic problem types: sorting, searching, graphs and geometric;
- CLO 4: Theoretically compare and analyse the time complexities of algorithms and data structures; and
- CLO 5: Implement, empirically compare, and apply fundamental algorithms and data structures to real-world problems.

2 Overview

Across multiple tasks in this assignment, you will model a disease outbreak in the city of Metropolis as a weighted graph to capture citizen's contact, implement and compare algorithms for tracing the most likely transmission pathways through the population, and design a dynamic programming strategy to allocate scarce antiviral treatment as effectively as possible.

You will address both the structural problem of representing and querying the contact network efficiently, and the algorithmic problem of finding optimal transmission paths and vaccination strategies under real constraints. Some components ask you to critically assess your solutions through both theoretical analysis and controlled empirical experiments, encouraging reflection on the relationship between algorithm design and real-world performance. The assignment emphasises strategic thinking and the ability to communicate algorithmic ideas clearly and effectively.

Important Notes

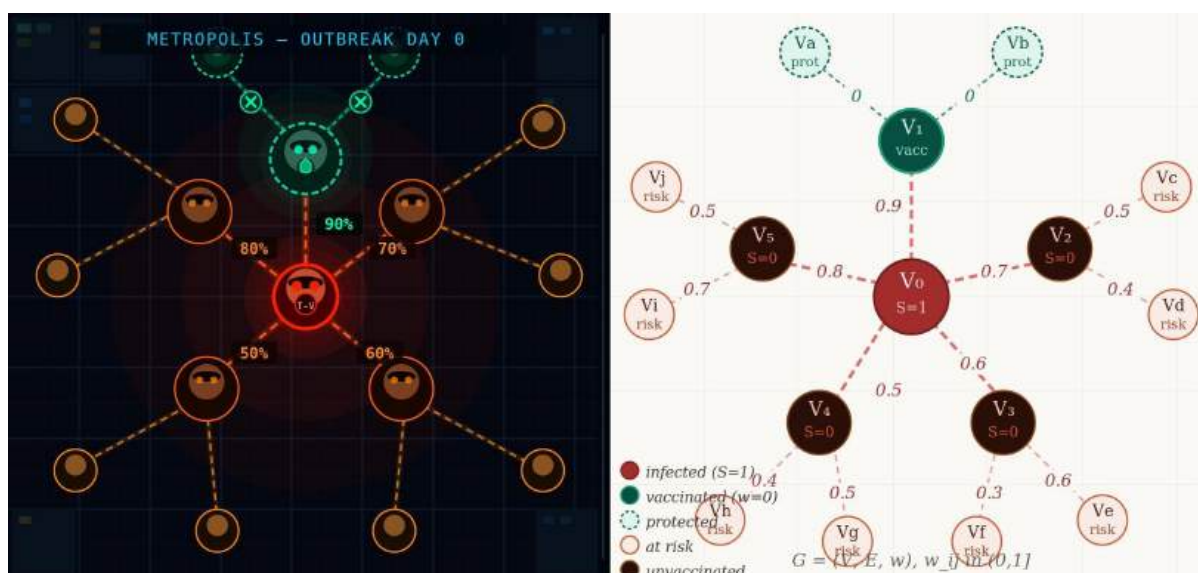
Please read all the following information before attempting your assignment.

- This is an *individual* assignment. You may not collude with any other person (or people) and plagiarize their work. Everyone is expected to present the results of their own thinking and writing. Never copy another student's work (even if they "explain it to you first") and never give your written work to others. Keep any conversation high-level and never show your solution to others. Never copy from the Web or any other resource. The use of any **generic or publicly available generative AI tool is strictly prohibited** for the assignment. Students are allowed to use Atlas for support. Suspected cases of collusion or plagiarism will be dealt with according to RMIT Academic integrity policy.
- Submit your written report and implemented code via **GitHub** Classroom by the deadline. Your submission must: (1) include your report file in the repository, and (2) be tagged with `submission`. Once submitted, you must also complete a confirmation form to verify that you have followed all instructions. Full submission instructions are available in the [assignment repository](#).
- Before implementing any code, please carefully read the `README` section provided in the skeleton code repository and add or modify code **only** within the files explicitly marked with the comment `# EDIT THIS FILE TO IMPLEMENT TASK X`. Any changes outside these marked sections may negatively affect the reliability of your solution and could cause your submission to fail our automated tests.
- You are expected to properly use git version control and **commit regularly, with clear and meaningful messages** to reflect the progress of your development. Submitting your entire solution in a single commit, or in just a few large commits, is considered poor practice. Please note that marks from automatic tests will be adjusted based on how well you follow these practices. Full marks will only be awarded when correct implementation is paired with clear evidence of good version control and development discipline.
- Respect the word and page limits specified for the report. Content beyond the specified limits will not be read and **will not be considered in marking**.
- Strictly adhere to the assignment submission deadline. The deadline will not be extended under any circumstances, except in the event of natural disasters or similar emergencies.
- As part of the assessment for this module, all students are required to attend a short interview in Week 12. This is a **hurdle requirement**: students who do not complete the interview will not have their assignment marked. Further details will be communicated via the Ed forum in due course.
- In the submission (your PDF file for the report) you will be required to certify that the submitted solution *represents your own work only* by including the following statement:

I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show I agree to this honor code by typing "Yes":

Contents

1	Learning Outcomes	1
2	Overview	1
2.1	Suggested Four-Week Timeline	4
3	Assessment Details	5
3.1	Motivation	5
3.2	Mathematical Background	6
3.3	Task A: Graph Representation (5 marks)	7
3.4	Task B: Estimating Infection Risk Over Time (7 marks)	9
3.5	Task C: Analysis of Infection Risk Algorithms (8 marks)	12
3.6	Task D: Antiviral Allocation (10 marks)	13
4	Assessment	16
5	Submission	16
5.1	Submission Guidelines	16
5.2	Re-submission	18
5.3	Late Submission Penalty	18
5.4	Extension	19
6	Academic integrity and plagiarism (standard warning)	19
7	Getting Help	19



2.1 Suggested Four-Week Timeline

It is highly recommended that you set up your developer environment, read the specification, and run/experiment with the code and configuration file as soon as possible. **You should start by reading the README file in your repository** - this contains instructions on setting up and running the code. Doing so will allow you to gauge how challenging you believe this assignment will be for you personally and where you are likely to have the greatest difficulties. The tasks do, generally, build in difficulty, and so Task D is more difficult than Task C and so on.

Please do read the tasks early and try to understand what is being asked of you. When you know the motive and objective of each task, you will be able to plan how best to create a great solution to this assignment. On average, since the assignment is 30 marks we expect the average student to spend approximately 36 hours on it in order to achieve the average mark (according to the accreditation process). Some will spend more, and some will spend less - that is up to the individual. This amounts to 9 hours per week or around an hour and fifteen minutes per day. Marks for each task are a good proxy for difficulty, therefore you should look to spend approximately:

1. 17% of your time on Task A;
2. 23% of your time on Task B;
3. 27% of your time on Task C; and
4. 33% of your time on Task D.

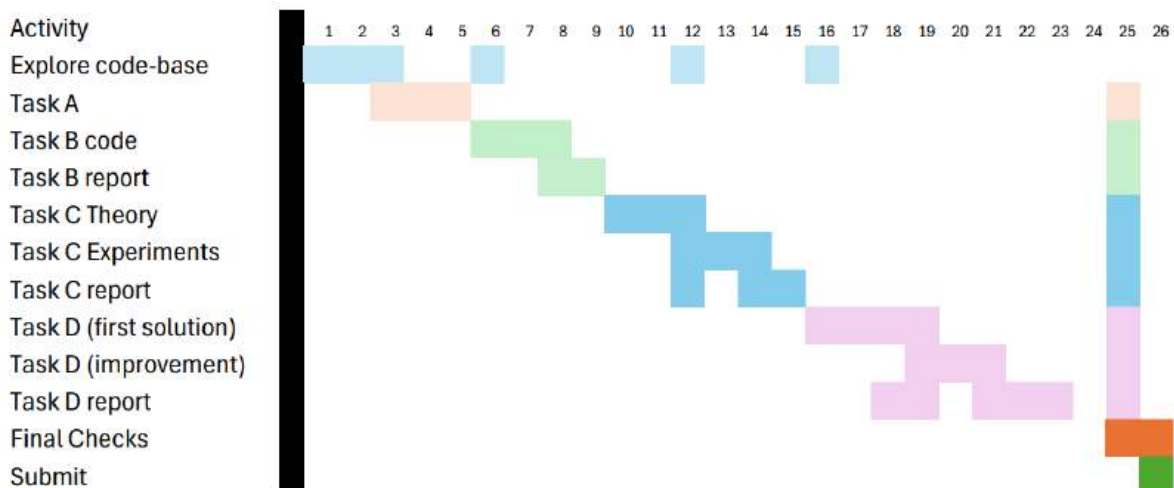


Figure 1: Gantt chart for a suggested timeline. Task D, which is more open ended, should (probably) have more time dedicated to it than other tasks to achieve top marks. Notice that coding tasks require re-exploring the code base. You may use any code in the code base to inspire your own solutions, so you need to have a look at how it works. It is not expected that you work on this assignment every single day - this is just a suggested timeline.

Do remember that we have access to your repository and commit history, and so can see when you start and work on the assignment. Do not panic - there is no punishment for starting the assignment late! But students who start early perform better, in general.

3 Assessment Details

The assignment is broken up into a number of tasks to help you progressively complete the project. Please note that although completing one task can assist with subsequent tasks, each task can be attempted independently. Moreover, the tasks are designed so that even if your implementation does not fully achieve the expected output, you can still discuss and present the theoretical aspects of your algorithmic design through your report.

3.1 Motivation

The T-Virus has broken out in the city of Metropolis. As the newly appointed Regional Health Director, you have been handed a dossier containing everything the epidemiologists know: a map of every known contact between residents, the probability that each contact transmits the virus in a single day, and the vulnerability of each resident. The clock is ticking — every day the virus spreads further, and your antiviral stockpile is limited.

Your job has four parts:

1. **Build the contact network** (Task A) — represent the city’s contact relationships as a graph, efficiently enough to run algorithms on in real time (implement `graph/adjacency_matrix.py`).
2. **Estimate infection risk** (Task B) — compute the probability that each resident becomes infected over the coming T days, identifying the most at-risk residents before the outbreak grows (implement `transmission/task_b.py` and write report).
3. **Analyse your solution** (Task C) — theoretically and empirically compare your algorithm against the provided baseline across different network structures (write report on analysis of task A and B).
4. **Allocate antiviral treatment** (Task D) — decide who receives the scarce supply of doses, maximising the benefit to the population within the limits of your stockpile (implement `treatment/task_d.py` and write report).

Figure 2 shows the contact network for a sub-population in Metropolis. Each resident is a node, and each contact relationship is an edge labelled with the daily probability of transmission. Patient zero V_0 is already infected — the question is how far the virus will spread, and who we should protect first.

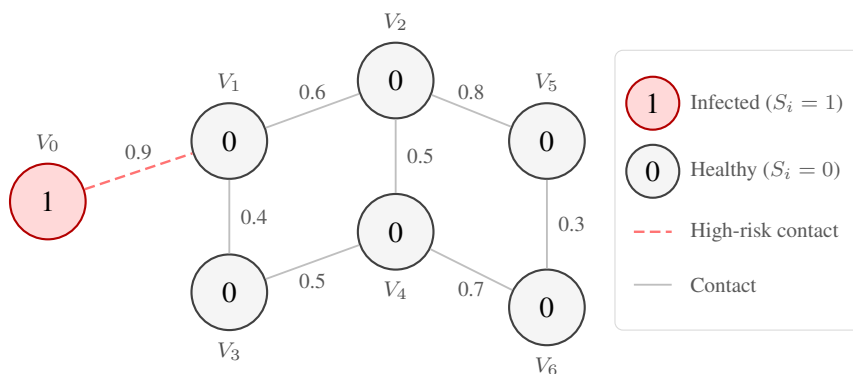


Figure 2: The contact network $G = (V, E)$ for Metropolis. V_0 is patient zero. Edge labels give the daily transmission probability $w_{ij} \in (0, 1]$. Dashed red edges originate from an infected resident.

By vaccinating even a small number of strategically chosen residents, we can cut the most dangerous transmission pathways through the network. Figure 3 shows an example: vaccinating V_1 — the single resident through whom all transmission from V_0 must pass — protects every downstream resident at once.

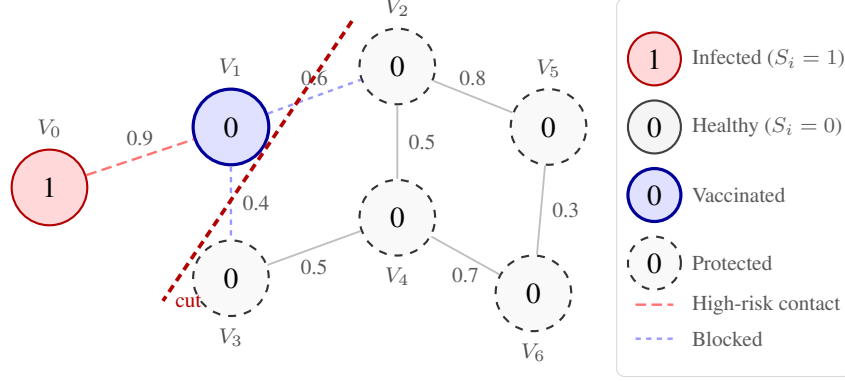


Figure 3: Vaccinating V_1 blocks every onward pathway at once since it is the single bottleneck through whom all transmission from V_0 must pass and protects V_2 through V_6 entirely. This illustrates the core challenge of Task D: identifying which residents to vaccinate in order to maximise protection across the network given a limited supply of doses.

3.2 Mathematical Background

We model the city of Metropolis as a weighted undirected graph $G = (V, E)$, where:

- each vertex $V_i \in V$ represents a resident, with infection state $S_i \in \{0, 1\}$ where $S_i = 0$ is healthy and $S_i = 1$ is infected;
- each edge $(V_i, V_j) \in E$ represents a contact relationship between two residents; and
- each edge carries a weight $w_{ij} \in (0, 1]$, the probability that an infected resident V_i transmits the virus to a healthy neighbour V_j in a single day. Since the graph is undirected, $w_{ij} = w_{ji}$.

Each resident V_i also has two personal attributes relevant to the vaccine allocation in Task D:

- a **dosage requirement** $c_i \in \mathbb{Z}^+$, the number of antiviral doses required to vaccinate resident V_i ; and
- a **vulnerability score** $\phi_i \in (0, 1]$, reflecting how severely resident V_i would be harmed if infected. This attribute is not used in the core allocation problem but becomes relevant in the extensions discussed in Task D.

Once infected, a resident remains infectious forever. Over a planning horizon of T days, the probability that an infected resident V_i infects neighbour V_j is:

$$p_{ij} = 1 - (1 - w_{ij})^T$$

As $T \rightarrow \infty$, $p_{ij} \rightarrow 1$ for any $w_{ij} > 0$ — given enough time, every resident reachable from patient zero will eventually be infected. This makes early intervention critical. The benefit of vaccinating resident V_i is therefore a function of how likely they are to be infected, and how much onward transmission they would cause — all of which are computed from G , and $r_{i,T}$ in Tasks B.

3.3 Task A: Graph Representation (5 marks)

Before the epidemiologists of Metropolis can run any algorithms on the contact network, they need an efficient way to store it. You learn that there are two standard ways to represent a graph: (1) an adjacency matrix; and (2) an adjacency list. The list representation is already implemented for you. Your task is to implement the adjacency matrix representation.

Implementation Task (5 marks)

The current implementation in `graph/adjacency_list.py` stores the contact graph as an adjacency list. You must implement the class in `graph/adjacency_matrix.py` to represent the same graph as an adjacency matrix. Your implementation must expose exactly the same public interface as the matrix, so that all downstream algorithms in Tasks B, C, and D work correctly with either representation by changing a single configuration variable.

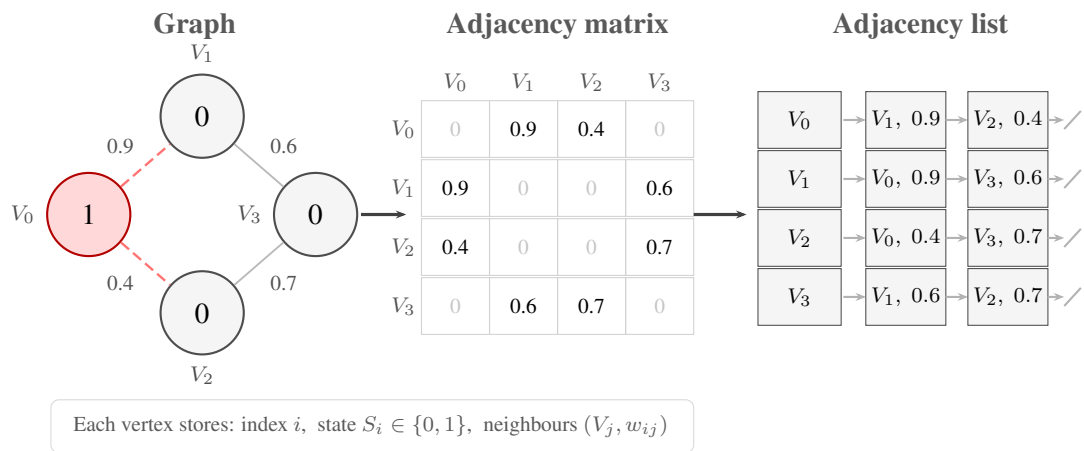


Figure 4: Flow of representation: contact graph \rightarrow adjacency matrix \rightarrow adjacency list. Nodes show infection state $S_i \in \{0, 1\}$ (red = infected, grey = healthy). The matrix stores an entry for every pair of residents, including zeros where no contact exists. The adjacency list stores only actual contacts as (neighbour, weight) pairs per vertex, using $O(|V| + |E|)$ space compared to $O(|V|^2)$ for the matrix.

Testing your Solution

To run using adjacency matrix representation, change the configuration variable `graph_type` from "list" to "matrix" in your configuration file. You are provided a test suite to check your solution — to run it, invoke the command `python -m tests.task_a`.

Report Task (0 marks)

You are **not** required to write anything in your report for Task A.

Tips

- Your matrix representation should exactly mimic the behaviour of the list implementation: algorithms will be identical in output, but the underlying data structure and the cost of operations will differ.
- Each cell `matrix[i][j]` must store the **edge weight** w_{ij} if a contact exists between V_i and V_j , or 0 if no contact exists.

- Since the contact graph is undirected, adding edge (V_i, V_j, w_{ij}) must set both `matrix[i][j]` and `matrix[j][i]`.
- Setting `print_struct = true` in the configuration file will print your data structure to the console, which is useful for debugging.

3.4 Task B: Estimating Infection Risk Over Time (7 marks)

With the contact network built, the epidemiologists of Metropolis need to estimate how dangerous the outbreak will become.

Monte Carlo Simulation

Your task is to compute, for every resident, the probability that they will be infected within a planning horizon of T days. We model the outbreak as a discrete-time process. Each day, every infected resident independently attempts to transmit the virus to each of their neighbours, with the probability of a single-day transmission given by the edge weight $w_{ij} \in (0, 1]$. Once infected, a resident remains infectious for the remainder of the horizon.

A Monte Carlo baseline is provided in `transmission/monte_carlo.py`. It simulates the outbreak X times, randomly deciding each day whether transmissions succeed according to the edge weights, and estimates $r_{i,T}$ as the fraction of simulations in which V_i was infected. It converges to the correct answer for large X , but is slow, approximate, and non-deterministic. Algorithm 1 shows its structure — note the nested loops, which will be important in Task C.

Algorithm 1 MonteCarlo(G, s, T, X)

Require: Graph $G = (V, E)$; source s ; horizon T ; simulations X

Ensure: Estimated risk table where $\text{table}[t][i] \approx r_{i,t}$

```

1: for each day  $t = 1$  to  $T$  do
2:   for each simulation  $x = 1$  to  $X$  do
3:     Restart from scratch — only  $s$  is infected
4:     for each day  $d = 1$  to  $t$  do
5:       for each uninfected resident  $V_i$  do
6:         for each infected neighbour  $V_j$  of  $V_i$  do
7:           Infect  $V_i$  with probability  $w_{ij}$ 
8:    $\text{table}[t][i] \leftarrow$  fraction of simulations where  $V_i$  infected
9: return table

```

A Dynamic Programming Alternative

The nested loop structure of Algorithm 1 iterates over days, simulations, residents, and neighbours, which makes it computationally expensive, and results vary between runs. Rather than estimating risk by simulation, we can compute it exactly. Notice that infection on day t can only be caused by infections that occurred by day $t - 1$: this temporal ordering means we can build the solution one day at a time, reusing previously computed values. The following recurrence captures this idea:

We define $r_{t,i}$ as the probability that resident V_i has been infected by the end of day t , with base cases:

$$r_{0,i} = \begin{cases} 1 & \text{if } V_i = V_0 \\ 0 & \text{otherwise} \end{cases}$$

Rather than summing over all the ways infection could occur, we compute the probability that V_i *escapes* infection and subtract from 1. Since transmission attempts from different

neighbours are independent, this gives the recurrence:

$$r_{t,i} = 1 - \underbrace{\left(1 - r_{t-1,i}\right)}_{\text{uninfected before day } t} \prod_{V_j \in \mathcal{N}(V_i)} \underbrace{\left(1 - r_{t-1,j} \times w_{ij}\right)}_{\text{neighbour } V_j \text{ fails to transmit}}$$

$r_{t,i}$ probability that resident V_i has been infected by the end of day t

$r_{t-1,i}$ infection probability of V_i at the previous time step

w_{ij} daily transmission probability along contact (V_i, V_j)

$\mathcal{N}(V_i)$ set of neighbours of V_i in the contact network

$\prod_{V_j \in \mathcal{N}(V_i)}$ is the product

Crucially, each column t of the table depends only on column $t - 1$ — we fill the table top to bottom, and within a row the order we process residents does not matter.

Worked Example

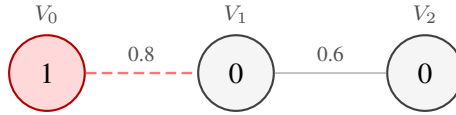


Figure 5: Chain network used in the worked example. V_0 is patient zero.

We trace the recurrence on Figure 5 with $T = 3$ days, filling one column at a time. At $t = 1$, V_1 has neighbours V_0 ($w = 0.8$) and V_2 ($w = 0.6$):

$$r_{1,1} = 1 - \underbrace{(1 - 0)}_{V_1 \text{ uninfected}} \underbrace{(1 - 1 \times 0.8)}_{V_0 \text{ fails}} \underbrace{(1 - 0 \times 0.6)}_{V_2 \text{ fails}} = 1 - (1)(0.2)(1) = 0.800$$

V_2 remains at 0 since its only neighbour V_1 is uninfected at $t = 0$. The complete table after $T = 3$ days:

Resident	$t = 0$	$t = 1$	$t = 2$	$t = 3$
V_0	1.000	1.000	1.000	1.000
V_1	0.000	0.800	0.960	0.994
V_2	0.000	0.000	0.480	0.780

Table 1: Risk table for the chain network, $T = 3$. Entry in row V_i , column t shows $r_{t,i}$. In code, this corresponds to `table[t][i]`.

Reading across each column shows how a resident's infection risk accumulates over time as the outbreak grows. V_2 , for instance, starts at zero risk but reaches 0.780 by day 3 — entirely because V_1 progressively became more likely to carry and transmit the virus. The final column $r_{T,i}$ summarizes each resident's total exposure risk over the planning horizon, and is the value used directly in Task D to prioritize who receives treatment. Note that Patient zero has $r_{t,0} = 1$ for all t to enforce this at every time step, not just $t = 0$.

Implementation Task (3 marks)

Your task: implement the exact dynamic programming solution in `transmission/task_b.py` using the recurrence above for $r_{t,i}$ (in the box). Your implementation must:

- correctly initialise the base cases at $t = 0$;
- fill the risk table row by row, where `table[t][i]` stores $r_{t,i}$; and
- work correctly with *both* graph representations from Task A.

Testing your Solution

To run using your dynamic programming solution, change the configuration variable `risk_solver` from `"monte_carlo"` to `"task_b"` in your configuration file. You are provided a test suite to check your solution — to run it, invoke the command `python -m tests.task_b`.

Report Task (4 marks)

1. (1 marks) **Pseudo-code.** Write concise pseudo-code for your dynamic programming solution in the style of Algorithm 1.
2. (3 marks) **Limitations of the recurrence.** The recurrence is a simplifying approximation of true disease spread — rather than tracking the full joint probability distribution over all possible infection states across the network, which would be exponentially complex, it makes a structural assumption about how transmission pathways interact.
 - (a) (1 mark) What structural property of our contact network makes the recurrence an approximation rather than an exact calculation?
 - (b) (1 mark) With reference to Figure 5, explain why the infection risk computed for V_1 at $t = 3$ is only approximate. Identify the specific term that introduces the error and state whether the recurrence over- or under-estimates the true risk.
 - (c) (1 mark) Identify the class of graphs on which the recurrence gives an exact answer rather than an approximation for $r_{t,i}$. Justify your answer.

Tips

- For large X (5000) and T (10), the Monte Carlo output and your DP table should agree closely — use this to verify correctness.
- Run `python -m tests.task_b` to check your implementation.

3.5 Task C: Analysis of Infection Risk Algorithms (8 marks)

With both algorithms implemented, the epidemiologists of Metropolis need to know which approach to deploy. A large, sparsely connected network (e.g. a city) over a short intervention window and a small, densely connected network (e.g. a submarine crew confined together for months, requiring a long planning horizon) present very different computational challenges — the best algorithm and graph representation for one may perform poorly on the other. In this task, you will theoretically analyse and empirically compare the Monte Carlo baseline and your dynamic programming solution from Task B across both graph representations from Task A.

Implementation Task (0 marks)

While you are required to run timing experiments as part of this task, no marks are awarded for the implementation itself.

Report Task (8 marks, maximum 2 pages)

Your report should include:

1. **Algorithm Complexity Analysis. (2 marks)** Strongly justify, with reference to the pseudo-code and the graph representation being used, the theoretical time complexity of all four combinations:
 - (a) Monte Carlo with an adjacency matrix;
 - (b) Monte Carlo with an adjacency list;
 - (c) Dynamic programming with an adjacency matrix; and
 - (d) Dynamic programming with an adjacency list.For each combination, identify the specific operation that dominates the runtime and explain how the choice of representation affects its cost.
2. **Empirical Design. (2 marks)** Identify which variables are most important when measuring the runtime of the four combinations. Explain which parameters you will vary in your experiments, which you will hold fixed, and why.
3. **Empirical Analysis. (2.5 marks)** Provide a thorough analysis using clearly labeled plots that allow you to compare the runtime of all four combinations.
4. **Reflection. (1.5 marks)** Discuss whether your empirical results align with your theoretical complexity predictions. Make a claim as to which algorithm/representation combination works the best for Metropolis and why. Would your answer change if transmission rates and connections updated daily?

Tips

- A timer utility is provided in `utils/timer.py`. Time only the algorithm itself — do not include graph construction or file I/O in your measurements.
- Average over multiple random contact networks for each data point to reduce noise. Set the random seed in the configuration file for reproducibility.
- Does the Monte-Carlo method consider any additional parameters?

3.6 Task D: Antiviral Allocation (10 marks)

The epidemiologists of Metropolis have used the infection risk scores from Task B to assess the risk to every resident. The city's medical authority has now issued a limited supply of C antiviral doses. Vaccinating a resident requires a course of treatment — different residents may require different numbers of doses depending on their age and health status. Your job is to decide who gets vaccinated. You have a limited supply of doses to distribute across the population. Using the infection risk scores computed in Task B as the benefit of vaccination, your goal is to choose which residents to vaccinate so that the total benefit is maximised within the dose budget, with ties broken in favour of the selection using the fewest doses.

Problem Formulation

Given: C total antiviral doses; $n = |P|$ residents, each with dosage requirement $c_i \in \mathbb{Z}^+$ (the number of doses required to vaccinate resident V_i). A resident with high infection risk stands to gain the most from vaccination, hence we use the infection risk score computed in Task B directly as the benefit score, i.e., $b_i = r_{T,i}$.

Objective: Select a subset P of residents to vaccinate that maximises total benefit:

$$\max\left(\sum_{i \in P} b_i\right) \quad \text{subject to} \quad \sum_{i \in P} c_i \leq C$$

Note that $P \subseteq V$ contains only healthy residents, i.e., $S_i = 0$ for all $V_i \in P$. Also, each resident is either fully vaccinated or not vaccinated at all — partial treatment is not permitted.

Tiebreaking: Among all selections achieving the maximum benefit, choose the one using the *minimum* total doses. This preserves as many doses as possible for future use.

Worked Example

Consider $C = 10$ doses and the following residents:

Resident	Benefit b_i	Doses c_i
V_1	0.600	6
V_2	0.600	3
V_3	0.900	4

Two selections both achieve the maximum benefit of 1.500:

Selection	Benefit	Doses used
$\{V_1, V_3\}$	$0.600 + 0.900 = 1.500$	$6 + 4 = 10$
$\{V_2, V_3\}$	$0.600 + 0.900 = 1.500$	$3 + 4 = 7$

Both are feasible and achieve equal benefit. The tiebreaking rule selects $\{V_2, V_3\}$ since it uses only 7 doses, leaving 3 doses available for future use. Note that V_1 and V_2 carry identical benefit scores — V_2 is simply cheaper to vaccinate, making it strictly preferable whenever doses are limited.

Implementation Task (4 marks)

Implement your solution in `treatment/task_d.py`. Your function receives the list of residents (each with attributes `cost` and `benefit`) and the total antiviral doses C , and returns the total benefit achieved and the list of selected resident IDs. Your implementation will be tested against a reference optimal solution. Marks are awarded based on how close your solution is to optimal across test cases of increasing size and complexity.

Marks	Optimality	Errors	Description
4	Optimal	None	Correct on all test cases and computes minimal subproblems.
3	Sub-optimal	Minor	Correct on all test cases or over-computes subproblems.
2	Sub-optimal	Minor	Over-computes subproblems and incorrect on a small number of cases.
1	(Sub)-optimal	Major	Runs but incorrect on the majority of cases.
0	Invalid	—	No solution submitted, times out, or violates constraints.

Testing your Solution

To run using your task D solution, change the configuration variable `vaccine_strategy` from `"brute_force"` to `"task_d"` in your configuration file. You are provided a test suite to check your solution — to run it, invoke the command `python -m tests.task_d`.

Tips

- Think about whether you have seen a similar problem structure before in this course.
- The benefit scores $b_i = r_{i,T}$ are real-valued rather than integer-valued — consider whether this affects your approach.
- Run `python -m tests.task_d` to evaluate your implementation on the provided test cases.

Report Task (6 marks, maximum 2 pages)

Your report should include:

1. **Algorithm design (1 mark).** Describe the algorithm you have designed to solve this problem, including any tables or data structures you use, your base cases, and how results are recovered.
2. **Complexity analysis (1 mark).** Justify the time complexity of your solution in terms of n (number of residents) and C (total capacity), explaining where each factor comes from.
3. **Extension 1: triage (2 marks).** Suppose residents are classified into K vulnerability tiers, where the most vulnerable must be allocated doses optimally before any doses go to lower tiers. Describe how you would modify your solution and state its complexity. Construct a small numerical example showing that ignoring vulnerability can lead to a highly vulnerable resident being passed over in favour of a cheaper-to-vaccinate but less vulnerable one. You are **not** required to implement this extension.

4. **Extension 2: interdependent vaccinations (2 marks).** The current model assumes $b_i = r_{i,T}$ is fixed regardless of who else is vaccinated. In reality, vaccinating V_i reduces the risk of V_i 's neighbours, making vaccinations interdependent. Explain why independence is necessary for your DP to be valid, construct a small example showing interdependence leads to a sub-optimal outcome, and explain why the true optimal problem is significantly harder to solve.

4 Assessment

The project will be marked out of 30. The assessment in this project will be broken down into several parts for each Task. The criteria discussed in Table 1 will be considered when allocating marks.

5 Submission

This assignment is due before **Thursday May 21st at 13:00 (1pm in the afternoon)**.

5.1 Submission Guidelines

Your assignment is to be submitted via the Github Classroom. When accepting the assignment via Github Classroom, the template code is forked creating a new, personal repository for you. To begin working on your code, you should clone the repository to your local machine. All development of your assignment's code should be documented via *Git commits* in this repository.

When you are happy with your code, you may begin the submission process.

Step 1: Github Classroom Begin by checking that you are connected properly to the github classroom. To do this, navigate to this page on EdStem and download the latest version of the *classroom_roster_mmd.csv* file. Check that your student number matches the GitHub username you used to create your repository. If it does not, leave a comment on the post with:

- your name;
- your student number; and
- your GitHub username

and we will help you resolve it.

Step 2: Upload PDF The report for your assignment should be stored as a PDF document. Please ensure that it is uploaded into the root folder of your repository (where the main python file `simulate_outbreak.py` is) with a reasonable name, such as *(student_number)_report.pdf*. The report makes up 18 marks of this assignment, and if it is not in the repository then we cannot mark it.

Step 3: Tag Submission When you are happy that your repository is in the correct state, you may make a submission. We do this by **tagging the commit you want marked as “submission”**. Doing this requires you to:

1. Find the commit you want to submit (red box below shows an example commit message);
2. Copy the commit's hash (green box below) by pressing the copy button (the two overlapping squares);



Figure 6: Example of what a commit history looks like. The red box is a commit message, and the green box is a commit's hash.

3. run the following commands in git bash:

- (a) `git tag -a submission <commit hash> -m "assignment submission"`
- (b) `git push origin submission`

This will create a submission tag, tag the commit `commit hash` as this tag and then push it to your repository.

What this essentially does is label a particular version of your code/report as the version you want marked. If you ever want to make a change to your submission, you may delete the tag using the commands:

1. `git tag -d submission`
2. `git push origin :refs/tags/submission`

and then go through the tagging process again with a different commit hash.

Step 4: Validation At this point, you should do a quick sanity check to ensure everything above has been done correctly. You should:

1. Check your student number is assigned to the right github account;
2. Check that your repository:
 - (a) has a tag and that the tag is called “submission” (not upper case or anything else):

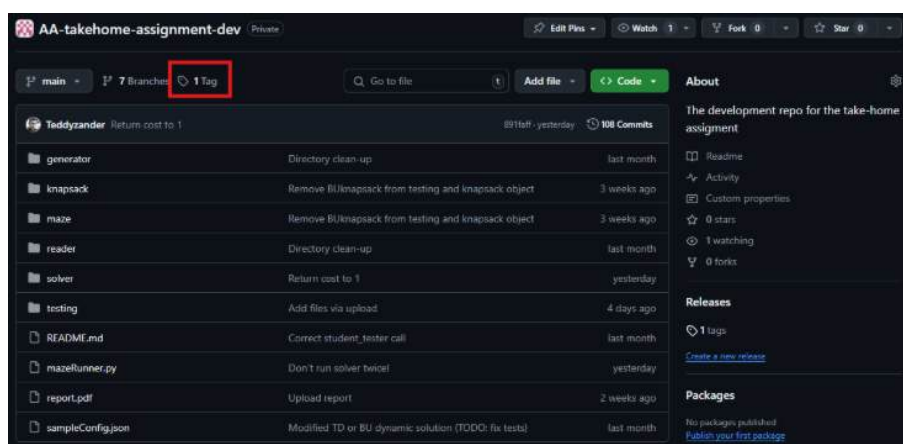


Figure 7: State of a repository when it has a tag. Observe that the red box shows us we have 1 tag. Clicking it will take us to the tags.

- (b) and going into tags and clicking on the submission tag takes you to your submission folder:

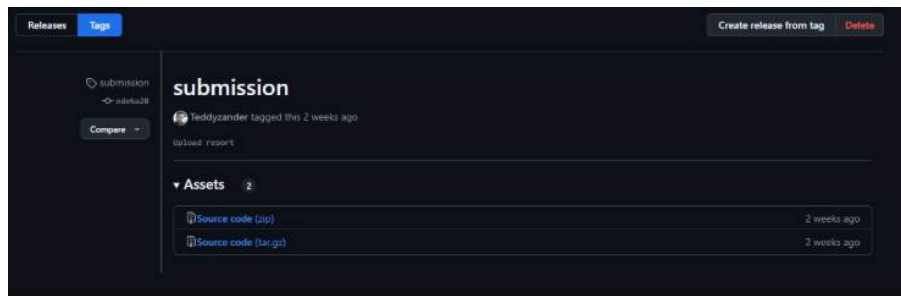


Figure 8: Your submission tag contains a source file (the state of your code at the tagged commit). This is what will be downloaded and marked.

3. Download the source file and check that:
- (a) the source file contains the report you want marked; and
 - (b) the source file contains the code you want marked.

Step 5: Confirm Submission Your final step is to fill in this form that confirms you have completed the above tasks. During this process, you will sign-off that you have checked that all the above has been completed correctly.

LINK TO THE FORM

You are responsible for resolving any issues with your assignment, though please do talk to us so we can assist you with resolving them.

5.2 Re-submission

We understand that mistakes can happen during submission — a missing report, an incorrectly named tag, or a file uploaded to the wrong location. Because all work is tracked via Git, we can verify exactly when files were created and last modified, meaning a re-submission does not provide an opportunity to complete additional work after the deadline.

Re-submission is only available to students who have completed the Week 7 lab. The Week 7 lab covers Git version control and walks through the submission process in detail. Completing it demonstrates that you made a genuine effort to learn the submission process before the deadline. Students who did not complete the Week 7 lab have not demonstrated this, and re-submission will not be granted.

5.3 Late Submission Penalty

Late submissions will incur a 10% penalty on the total marks of the corresponding assessment task per day or part of day late, i.e, 3 marks per day. Submissions that are late by 5 days or more are not accepted and will be awarded zero, unless special consideration has been granted. Please ensure your submission is correct as re-submissions after the due date and time will be

considered as late submissions. We strongly advise you submit **at least one hour before the deadline**. Late submissions due to slow Internet will not be looked upon favorably, even if it is a few minutes late. Any claim of late submission due to slow Internet will require documentation and evidence that submission attempts were made at least **one hour before the deadline**.

5.4 Extension

Extension requests must be submitted through **Canvas at least 24 hours before the deadline**, and must be accompanied by supporting documentation (e.g., a medical certificate). Requests without documentation will be rejected. If your circumstances arise within 24 hours of the deadline, you must submit a **Special Consideration application** via this link. Please refrain from contacting the course coordinator or teaching team via email for extension-related matters.

6 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another source without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offense constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to this link.

7 Getting Help

There are multiple venues to get help. There are two weekly consultation hours as well as weekly decode & conquer sessions (see the Ed Forum for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor. We will also be posting common questions on the Ed Forum and we encourage you to check and participate in the discussion. However, please **refrain from posting solutions**, particularly as this assignment is focused on algorithmic design.

Table 1 - Assessment Criteria

Criteria	Ratings	Pts
Task A Implementation Automatic Testing Results	5 to 0 pts Number of passed tests normalised by appropriate software engineering practices (subject to valid implementation)	5 pts
Task B Implementation Automatic Testing Results	3 to 0 pts Number of passed tests normalised by appropriate software engineering practices (subject to valid implementation)	3 pts
Task B Report	<p>Complete - 4 to 3 pts The report is well-organised. The pseudo-code is correct, clearly presented, and follows the style of Algorithm 1. The discussion of the recurrence's limitations is accurate: the structural property responsible for the approximation is correctly identified, the worked example with reference to Figure 5 is sound and the correct term is identified, and the class of graphs giving an exact answer is correctly stated with a clear justification.</p> <p>Satisfactory - 3 to 2 pts The pseudo-code is present but contains minor ambiguities or errors. The limitations discussion addresses most of the three sub-questions but lacks depth or precision in one area — for example, the graph class is stated without sufficient justification, or the error term is not precisely identified.</p> <p>Incomplete - 2 to 0 pts The pseudo-code is missing or significantly flawed. The limitations discussion is absent, incorrect, or addresses fewer than two of the three sub-questions meaningfully.</p>	4 pts
Task C Report	<p>Complete - 8 to 6 pts The report is well-organised and all four aspects are comprehensively addressed: (i) complexity analysis is correct for all four combinations and the justification is clear and sound, referencing the pseudo-code and graph representation; (ii) the empirical design is sound, all key variables are identified, and the inclusion and exclusion of parameters are well-justified; (iii) the empirical results are clearly presented with labelled plots comparing all four combinations; and (iv) the reflection aligns theoretical and empirical results, makes a clear recommendation for Metropolis, and discusses the impact of dynamic transmission rates.</p>	8 pts

	<p>Satisfactory - 6 to 4 pts The report is generally understandable but may have some unclear sections in regards to one or two of the listed aspects above.</p> <p>Incomplete - 4 to 0 pts The report does not include a proper discussion on the empirical design and/or the empirical analysis lacks depth and proper justification. The theoretical analysis is incorrect and/or not well-explained and the report lacks a sound comparison between the theoretical and empirical analyses.</p>	
Task D Implementation Automatic Testing Results	<p>The marking will be awarded following a comparison with our solution. All bands below are determined based on the number of passed tests normalised by appropriate software engineering practices (subject to valid implementation):</p> <p>Optimal — 4 pts Correct on all test cases and computes minimal subproblems.</p> <p>Sub-optimal, minor errors — 3 pts Correct on all test cases or over-computes subproblems.</p> <p>Sub-optimal, minor errors — 2 pts Over-computes subproblems and incorrect on a small number of cases.</p> <p>(Sub)-optimal, major errors — 1 pt Runs but incorrect on the majority of cases.</p> <p>Invalid — 0 pts No solution submitted, solution times out, or violates constraints.</p>	4 pts
Task D Report	<p>Complete - 6 to 4.5 pts The algorithm design is clearly described with correct base cases, recurrence, and backtracking. The complexity analysis is correct and each factor is well-justified. The triage extension is clearly described, the complexity is correctly stated, and a sound numerical counterexample is provided showing that ignoring vulnerability leads to a suboptimal outcome. The interdependence extension correctly identifies why the independence assumption is necessary, constructs a valid counterexample, and clearly explains why the true optimal problem is significantly harder to solve.</p>	6 pts

	<p>Satisfactory - 4.5 to 2.5 pts</p> <p>The algorithm design and complexity analysis are present but may contain minor errors or lack precision. The extensions are addressed but one or both lack depth — for example, the counterexample is present but not well-justified, or the hardness argument for interdependence is incomplete.</p> <p>Incomplete - 2.5 to 0 pts</p> <p>The algorithm design is missing or significantly flawed. Complexity analysis has incorrect reasoning. One or both extensions are absent or contain fundamental errors.</p>	
--	--	--