

Summative Assignment

Module code and title	COMP2261 Artificial Intelligence
Academic year	2025-26
Coursework title	AI Search coursework
Coursework credits	5 credits
% of module's final mark	25%
Lecturer	Iain Stewart
Submission date*	Thursday, 19 February, 2026 14:00
Estimated hours of work	10
Submission method	Ultra

Additional coursework files	AI_Searchfile012.txt, AI_Searchfile017.txt, AI_Searchfile021.txt, AI_Searchfile026.txt, AI_Searchfile042.txt, AI_Searchfile048.txt, AI_Searchfile058.txt, AI_Searchfile175.txt, AI_Searchfile180.txt, AI_Searchfile535.txt, AI_Search_Proforma.docx, alg_codes_and_tariffs.txt, skeleton.py, validate_before_handin.py
Required submission items and formats	AlgA_basic.py, AlgB_basic.py, AlgA_enhanced.py, AlgB_enhanced.py, AlgA_AI_Searchfile012.txt, AlgA_AI_Searchfile017.txt, AlgA_AI_Searchfile021.txt, AlgA_AI_Searchfile026.txt, AlgA_AI_Searchfile042.txt, AlgA_AI_Searchfile048.txt, AlgA_AI_Searchfile058.txt, AlgA_AI_Searchfile175.txt, AlgA_AI_Searchfile180.txt, AlgA_AI_Searchfile535.txt, AlgB_AI_Searchfile012.txt, AlgB_AI_Searchfile017.txt, AlgB_AI_Searchfile021.txt, AlgB_AI_Searchfile026.txt, AlgB_AI_Searchfile042.txt, AlgB_AI_Searchfile048.txt, AlgB_AI_Searchfile058.txt, AlgB_AI_Searchfile175.txt, AlgB_AI_Searchfile180.txt, AlgB_AI_Searchfile535.txt, AI_Search_Proforma.pdf, AI_Search_Validation_Feedback.txt

* This is the deadline for all submissions except where an approved extension is in place. For assessments carried out in weekly practicals or other scheduled sessions, the date shown will be the Monday of the week in which the assessments take place.

Late submissions received within 5 working days of the deadline will be capped at 40%.

Late submissions received later than 5 days after the deadline will receive a mark of 0.

It is your responsibility to check that your submission has uploaded successfully and obtain a submission receipt.

Your work must be done by yourself (or your group, if there is an assigned groupwork component) and comply with the university rules about plagiarism and collusion. Students suspected of plagiarism, either of published or unpublished sources, including the work of other students, or of collusion will be dealt with according to University guidelines (<https://www.dur.ac.uk/learningandteaching.handbook/6/2/4/>).

AI SEARCH ASSIGNMENT

***** *It is really important that you read this document* *****
***** *thoroughly before you start. I'm sorry to appear* *****
***** *so dramatic with my bold-italic-red script but it* *****
***** *really is important that you follow the guidelines.* *****

*It is best if you watch the accompanying video
before reading further.*

Overview

You are to implement **two different algorithms** (call them AlgA and AlgB) in **Python** to solve the *Travelling Salesperson Problem (TSP)*. Your algorithms will almost surely be drawn from those we cover in the lectures but you might also implement other algorithms you have devised or discovered for yourself (though I do not advise this: doing so will need my **explicit consent** and will entail additional information on the proforma that you hand in; more later). Your implementations should seek to obtain the best TSP tours that you can, given 10 collections of cities and their city-to-city distances that I will supply to you. You will need to hand in the following items:

- between **2 and 4 correct Python programs** comprising of:
 - at least a basic implementation of each of your chosen algorithms, necessarily named `AlgAbasic.py` and `AlgBbasic.py`; and
 - possibly an enhanced implementation of each of your chosen algorithms, necessarily named `AlgAenhanced.py` and `AlgBenhan-ced.py`

(moreover, all implementations need to be precisely formatted and based around skeleton code `skeleton.py` that I will supply to you; more later)

- for each of the two chosen algorithms that you implement, **10 tours**, one for each city file that I give you, detailing the best tours you have found with *any* implementation of that algorithm, basic or enhanced, over the course of the assignment (moreover, each tour needs to be in a specifically named and formatted tour file; more later); so this amounts to 20 tour files

- a ***proforma*** (a pdf document) that clearly describes the enhancements you have made in your enhanced implementations, in comparison to the basic implementations, and also describes non-standard algorithms that you may have implemented that have not been covered in lectures
- a ***validation file*** (a text file) that contains the output from a run of the validation program that I supply to you immediately prior to hand-in (more on this later).

Mark scheme

The mark scheme is complicated so ***please read it carefully*** as it will determine which algorithms you ultimately choose to implement and help you plan your time. While reading the mark scheme, questions might occur to you. Hopefully they will be answered in the FAQs on ULTRA and which contain more details and clarification; but if they aren't then please email me.

Note that for students who deviate from the instructions in this document, I may impose ***mark fines***. These fines will be explained in the feedback you receive after your submission has been marked (and more information on mark fines is given in the accompanying video).

Marks will be awarded for:

- (a) the ***sophistication*** of the algorithms that you implement
- (b) the ***correctness*** of your basic implementations (`AlgAbasic.py` and `AlgBbasic.py`)
- (c) any ***enhancements*** you have made so as to obtain enhanced implementations (`AlgAenhanced.py` and `AlgBenhanced.py`) of your basic implementations
- (d) the ***quality*** of the tours that you obtain.

Your codes

I will measure ***sophistication*** and ***correctness*** as follows.

- **Sophistication:** Each algorithm has a tariff associated with it (as specified in the text file `alg_codes_and_tariffs.txt` that I supply to you) where this tariff gives the maximum number of (normalized) marks (out of 10) that you can secure with your basic implementation (`AlgAbasic.py` or `AlgBbasic.py`) of the ‘vanilla’ version of that particular algorithm (by ‘vanilla’ I mean the standard version such as that covered in lectures). The tariff is such that the more

technically complex the algorithm, the more marks you can secure. However, ... your **sophistication** mark will be derived from the maximum tariff from your **correct** basic implementations (the definition of ‘correctness’ follows).

- **Correctness:** I will run your basic implementations (`AlgAbasic.py` and `AlgBbasic.py`) on 2 secret sets of cities that I will not divulge to you. One city file will have around 50 cities and the other city file will have around 100 cities. Your basic implementations need to be **correct**: when I run your basic implementations on my secret city files, a legal tour is produced for both city files (I don’t care about the tour length). A full **correctness** mark is obtained if both of your basic implementations are correct; half the full **correctness mark** is obtained if one of your basic implementations is correct; and a **correctness** mark of 0 is obtained if none of your basic implementations is correct.

So, for example, if you have only one correct basic implementation then your **sophistication** mark will be obtained from the tariff corresponding to the correct basic implementation and you will get half the full **correctness** mark. (Of course, if neither of your basic implementations is correct then you will not be awarded either a **sophistication** mark or a **correctness** mark).

As mentioned above, you can implement an algorithm not covered in the lectures. If you choose to do so then you *must email me beforehand and provide details and a reference for your chosen algorithm*; if everything is OK then I will email you my consent. Also, you *must include in the pro-forma a brief description of the algorithm, using pseudocode and natural language* and also a *full reference as to the source used for your algorithm* so that I can consult this reference (you’ll see from the template I supply, `AISeachProforma.docx`, how you should do this). If you don’t supply these details satisfactorily then both your basic and enhanced implementations may be deemed to be incorrect. Remember: you need to get my **explicit consent** if you wish to implement a non-standard algorithm.

As well as **sophistication** and **correctness marks**, you can pick up extra **enhancement** marks by enhancing and experimenting with your basic implementations to obtain two enhanced implementations (`AlgAenhanced.py` and `AlgBenhanced.py`). For example, you might try different versions of crossover for a genetic algorithm and this experimentation might secure extra bonus marks at my discretion, though the more extensively and imaginatively you experiment, the more marks you’ll receive. Note that it is difficult to see how to significantly enhance some of the more elementary algorithms and consequently it might be difficult to secure many enhancement marks; this may affect your initial choice of algorithms to implement.

- You will be awarded an enhancement mark for each of your enhanced implementations separately (of course, if you only supply one enhanced implementation then the maximum **enhancement** mark you

can be awarded is only half the potential total). However, in order to be awarded an **enhancement** mark for an enhanced implementation, this implementation must be correct, with the definition of ‘correct’ as above. No matter how much you have enhanced a basic implementation, if the enhanced implementation is not correct then it will receive no **enhancement** mark.

Your **enhancement** mark will be derived primarily by the commentary and explanation you submit in the proforma together with, secondarily, a code inspection, so you should ensure that your code is *properly commented* (if your proforma is blank then it will be assumed that your enhanced implementation does not exist). You should also ensure that you adhere to the guidelines and formatting as regards the proforma, explained in the template I supply to you, as if you fail to respect these guidelines then I will consider the proforma to be invalid and you will receive a reduced **enhancement** mark. You should not relegate a description of your enhancement to your code: a full and clear enhancement description should be in the proforma with code comments used to show how the code implements the described algorithm (I expect all codes to be heavily commented). If you do not comment your code and I look at your code to check your claims then do not be surprised if you receive both a reduced enhancement mark and a fine for having poorly commented code.

It is up to you as to whether you wish to try and enhance your basic implementations (though once you have your basic implementations, it is not particularly time-consuming to do a little bit of enhancement). Ideally, you should hand in 2 basic implementations ([AlgAbasic.py](#) and [AlgBbasic.py](#)) of two different TSP algorithms *as well as* 2 enhanced implementations ([AlgAenhanced.py](#) and [AlgBenhanced.py](#)) of *the same* algorithms. Note that you are *not allowed* to implement 4 different TSP algorithms!

Some students are very imaginative and thorough and really make a good job of enhancement. In order to reward such students, I am usually quite tough on the award of enhancement marks. Just to reiterate: you don’t get an enhancement mark just for enhancing; you get an enhancement mark for how imaginatively, thoroughly and deeply you have enhanced (and only if the enhanced implementation is correct). Note also that I do not consider parameter manipulation and similar basic experimentation as enhancement: this is something you should be doing when collecting your tours.

In addition, the output tours from all of your implementations must have been obtained by an execution of code that implements the stated algorithm; so, if you claim to have implemented a genetic algorithm, say, but the code just outputs a randomly chosen tour then your implementation will be deemed incorrect. Alternatively, if you claim to have implemented an

ant colony algorithm, say, but the code actually implements a hill-climbing search then your implementation will be deemed incorrect. If a code is deemed to be incorrect as above then you will be fined a certain number of marks or if I think that you have maliciously cheated then you may be awarded a mark of 0 for the whole assignment. (With this in mind, you should make sure that you fully understand the difference between a basic greedy search and a greedy best-first search as students often confuse the two.)

Note also that *you should not edit an output tour-file obtained by any run of your codes.* The only thing you can do with these files is to rename them. Edited tour-files will be deemed illegal and might result in an accusation of plagiarism if I think that you have cheated and supplied an improperly-obtained tour.

Your tours

You will also receive a **quality** mark consisting of the sum of a **basic quality** mark and an **enhanced quality** mark. The **basic quality** mark corresponds to how good the tours are that you have found. For each of the 10 given city files, you should return: the best tour you have produced by *any* implementation of your first algorithm (AlgA), enhanced or otherwise; and the best tour you have produced by *any* implementation of your second algorithm (AlgB), enhanced or otherwise (you can vary the parameters as you see fit). However, ... the **basic quality** mark will be determined only by the *best tour* you have found for each of the 10 city files (regardless of whether this is via an implementation of AlgA or of AlgB).

- The **basic quality** mark consists of 10 components, one for each set of cities, and reflects how good your tours are relative to tours produced by others and by my own benchmarks.

You could also receive an **enhanced quality** mark.

- For each algorithm, I will run your basic implementation and your enhanced implementation on my secret city sets and depending upon how well your enhanced implementation does in comparison with your basic implementation, I will award an **enhanced quality** mark. Of course, to be awarded an **enhanced quality** mark for some algorithm, you need that both the basic implementation and the enhanced implementation are correct (the **enhanced quality** mark is relatively small).

When I compare a basic implementation with an enhanced implementation, please ensure that the core parameters are identically set, e.g., if your AlgA is a genetic algorithm then you should ensure that the number of iterations, size of population, and so on, are identically set in `AlgAbasic.py`

and `AlgAenhanced.py` when you hand them in; moreover, *you should set the values for all of these parameters right at the beginning of your code and in a clear and identifiable way*. If you fail to do this then I may consider your enhanced implementation to be incorrect and so you will not be awarded an enhancement mark or an enhanced quality mark.

In summary, *guidelines* for the possible marks for each category are as follows:

- **sophistication**: the maximum available mark accounts for around 30% of the total mark but your actual mark will depend upon the tariffs of the algorithms implemented and whether the basic implementations are correct
- **correctness**: if both basic implementations are ‘correct’ then a full mark of around 15% of the total mark is awarded; if one is ‘correct’ then a full mark of around 7.5% of the total mark is awarded; and if none is ‘correct’ then you receive a mark of 0
- **enhancement**: the maximum available mark accounts for around 25% of the total mark (with around 12.5% per enhanced algorithm) with the actual mark dependent upon how imaginative, thorough and deep your enhancements are
- **quality**: the maximum **basic quality** mark accounts for around 85% of the overall **quality** mark and the maximum **enhanced quality** mark for around 15%, with the overall **quality** mark accounting for around 30% of the total mark.

The format of your submission

All submissions should be named using your user-name as follows. I illustrate using the dummy user-name `abcd12` but you should substitute your own. *Please ensure that your user-name is all lower case.*

*If you don't follow the rules below then
you run the risk of being fined!*

All files should be in a folder called `abcd12`. Within this folder there should *only* be:

- 4 Python programs (.py) entitled
 - `AlgAbasic.py`
 - `AlgBbasic.py`
 - `AlgAenhanced.py`
 - `AlgBenhanced.py`

which are the basic implementations of your two chosen algorithms, AlgA and AlgB, along with the enhanced versions

- 10 tour files (.txt) entitled
 - [AlgA_AISearchfile012.txt](#)
 - [AlgA_AISearchfile017.txt](#)
 - [AlgA_AISearchfile021.txt](#)
 - [AlgA_AISearchfile026.txt](#)
 - [AlgA_AISearchfile042.txt](#)
 - [AlgA_AISearchfile048.txt](#)
 - [AlgA_AISearchfile058.txt](#)
 - [AlgA_AISearchfile175.txt](#)
 - [AlgA_AISearchfile180.txt](#)
 - [AlgA_AISearchfile535.txt](#)

with each tour file containing the best tour you have found using *any implementation* of your first chosen algorithm, AlgA, on the corresponding city set

- 10 tour files (.txt) entitled
 - [AlgB_AISearchfile012.txt](#)
 - [AlgB_AISearchfile017.txt](#)
 - [AlgB_AISearchfile021.txt](#)
 - [AlgB_AISearchfile026.txt](#)
 - [AlgB_AISearchfile042.txt](#)
 - [AlgB_AISearchfile048.txt](#)
 - [AlgB_AISearchfile058.txt](#)
 - [AlgB_AISearchfile175.txt](#)
 - [AlgB_AISearchfile180.txt](#)
 - [AlgB_AISearchfile535.txt](#)

with each tour file containing the best tour you have found using *any implementation* of your first chosen algorithm, AlgB, on the corresponding city set

- one proforma (.pdf) entitled
 - [AISeachProforma.pdf](#)

(I give you the template in Word but please save it and return it in the form of a pdf)

- one validation feedback file (.txt) entitled
 - `AIValidationFeedback.txt`

(I say more about this file in a moment).

Note that all of the above names are case-sensitive.

It is pointless including anything else in your folder `abcd12` (such as the folder of city files or the algorithm codes and tariffs file) or, indeed, anything else outside of your folder. On receiving your folder, the first thing I do is to *automatically delete everything inside the folder with a name different from the list of names above* (and also everything outside the folder).

Finally, note that your folder of files should be submitted via ULTRA where you should simply zip the folder `abcd12`, *and nothing else*, and submit the zip-file. So, when I open your zip file, I should see the folder `abcd12` within which lie all your submission files. Please use zip and not rar. In order to help you feel secure that you have named all the files correctly, I supply a validation program for you to run before hand-in (see below). As I state above, the output from this validation program (run immediately before hand-in) needs to be returned inside the folder `abcd12`.

More on Python

There are some Python restrictions on your codes and ***these are important***. I supply a file `skeleton.py` that ***you must use*** when supplying your implementations (when you have included your own code in `skeleton.py`, you should rename the file appropriately as, e.g., `AlgAbasic.py` or `AlgEnhanced.py`, or whatever, and then run your code and experiment on the sets of cities). There are *strict instructions* within `skeleton.py` that you should read and follow. The tour-files that you end up submitting will be tour-files provided as output from your codes. ***Do not manually edit these tour-files***. The only change you can make to a tour-file is to rename it. Any tour-file that has been manually edited will be deemed illegal.

The definitive list of algorithms and tariffs is supplied in the text file `alg_codes_and_tariffs.txt`. You need to download this file and use it as explained in `skeleton.py`. The file `alg_codes_and_tariffs.txt` may change as students request tariffs for algorithms as yet unencountered (I'll email all students when this happens and you can re-download it from ULTRA).

There are three important technical points to make. First, some Python IDEs automatically re-format Python files according to some coding convention, e.g., PEP 8, Prettier. Doing this will completely mess up your files so please check that your Python editor doesn't do this or you run the risk of your submission being deemed invalid. Second, I read your Python programs character by character and in doing so I assume that you have

used the ‘UTF-8’ encoding (which is the norm). Please ensure that your code respects this. Third, some type hints may cause difficulties. I advise that you do not use type hints.

In order to help you ensure that you adhere to the instructions, I supply a Python program `validate_before_handin.py`. This program will validate your submission before you hand it in. *If your submission does not validate then you run a serious risk of losing marks.*

VALIDATE BEFORE HAND-IN!

Read the instructions in the comments in `validate_before_handin.py` to see how to use it. Note that the program alerts you as to what is wrong with your submission (in a validation feedback file) so that you can fix it. *Validation does not check for everything that you should have done and so is not definitive.* So, you might have programs that validate but which will still be deemed incorrect and also programs that do not validate yet run perfectly well within my assessment framework. What’s the solution? Follow my simple instructions and you’ll be fine. If you are struggling to validate and don’t know what’s going wrong then email me.

Please be aware that *you are not allowed to import non-standard modules* (including `numpy`) into your Python codes. You can see the modules that you can import if you look in `validate_before_handin.py`. If you submit an implementation that uses a non-standard module then the implementation will be deemed to be incorrect.

One final (but important) thing: I will be executing your implementations automatically and it is possible that you choose parameters so that your implementations (on my secret city sets) take a long time to execute. When you hand your implementations in, you should ensure that your implementations will take no more than *one minute* on my secret city sets (of sizes around 50 and 100). Typical parameters that you will need to adjust are, for example, the number of iterations in a genetic algorithm, the temperature function in simulated annealing, and so on. *If an execution of one of your codes takes too long then it will be killed and deemed to be incorrect.*

You may force termination within one minute. For example, suppose you have implemented an A* search and the implementation has not terminated within a minute (which is likely to be the case). You may artificially halt the execution and return the best tour found up until this point (or, more precisely, some sort of completion of one of the partial tours found; you’ll have to decide how you do this). However, take care that your whole implementation terminates within one minute, including reading the city set data, setting things up, running your code and writing your tour to the output file. If you have a timer that decides to kill execution after t seconds then you should not choose t to be 60 but something less. *I will simply automatically kill any execution after one minute.*

Note that the time taken by an implementation of some algorithm may vary from execution to execution and be affected by data that you write or print during execution (so, in particular, you should ensure that the code you supply does not write or print during execution). For example, take a genetic algorithm. If you choose to terminate after 100 iterations then the time taken per iteration will be influenced by the number of cities, the size of the population, the randomized choices, the complexity of the cross-over, whether you print data during the execution, the data structures you use and so on. Also, the speed of the processor will have an effect. I will be executing your codes on my laptop where the processor is an AMD Ryzen 5 Pro 7535U and using Python 3.13 within Windows 11. Please ensure that you respect this in your codes. In particular, if you try to import the module `signal` so as to kill an execution after a certain amount of time then the variable `signal.SIGALRM` is only available on Unix and not on Windows; so your code will crash for me. You'll have to terminate your code in another way.

Finally, note that I only require your submitted implementations to terminate within one minute on my secret city sets. There are *no limitations* as to how long you run your implementations for on the other 10 city sets in order to derive the tours that you hand in (you can run them for a week, for all I care). Similarly, the parameter settings for your implementations can be anything you like when you run them on the 10 city sets; they only needed to be appropriately set on the submitted implementations (as I explained above). Also, the tours you hand in can be obtained from the basic or the enhanced implementations of your algorithms.

Plagiarism

I expect you to *write your own codes* and I explain more about this in the FAQs. All of your codes should be **heavily commented** so as to explain what you are doing and you should retain the structure of the pseudocode supplied in lectures in the vanilla versions of your codes. **Insufficiently commented code will be subject to fines**. You should not use generative AI: you can never be sure what you are getting. Besides, there is no point as I give you full pseudocode for all the algorithms and it is straightforward to code any of them by using this pseudocode. Code deemed to have been supplied by generative AI will be treated as plagiarism.