# AI Search Assignment

This is the assignment for the sub-module *AI Search* of the module *Artificial Intelligence*. It is to be completed and handed in **via ULTRA** by **2 p.m.** on **2 May 2023**.

***** 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. *****

## Overview

You are to implement ***two different algorithms*** (call them AlgA and AlgB) in ***Python*** to solve the *Travelling Salesman Problem* (*TSP*). Your algorithms can be drawn from those we cover in the lectures but you might also implement other algorithms you have devised or discovered for yourself (though 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 `AlgBenhanced.py`

  (moreover, all implementations need to be precisely formatted and based around skeleton code 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 briefly 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 that follow and which contain more details and clarification; but if they aren't then please email me.

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.

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. The correctness mark is awarded only if *both* of your basic implementations are correct, otherwise the mark is 0.

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 but you will not be awarded a 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, I am happy for you to 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 proforma 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, `AISearchProforma.docx`, how you should do this). If you don't supply these details satisfactorily then both your basic and enhanced implementations will be deemed to be incorrect.

In addition, 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 basic 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*. 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 no enhancement mark. You should not relegate a description of your enhanced algorithm to your code: the algorithm description should be in the proforma with code comments used to show how

the code implements the described algorithm. 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 no enhancement mark.

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 experimentation). 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.

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 but the code actually implements a hill-climbing search then your implementation will be deemed incorrect. Moreover, your overall mark for the assignment will be 0 as I regard this as cheating. (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.)

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. 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 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 of your implementations 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, otherwise you receive a mark of 0
- enhancement: the maximum available mark accounts for around 20% of the total mark (10% per algorithm) with the actual mark dependent upon the degree to which you have tried enhancements
- quality: the maximum basic quality mark accounts for around 80% of the overall quality mark and the maximum enhanced quality mark for around 20%, with the overall quality mark accounting for around 30% of the total mark.

An immediate calculation shows that the stated fractions do not add up to 100%: I will choose the actual fractions after marking. However, they will not diverge significantly from the percentages above and so will not affect the focus you place on the different aspects of the assignment.

Finally, 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.

## 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.

***If you don't follow the rules below then***
***you run the risk of getting no marks!***

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

  - AISearchProforma.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

  - AISearchValidationFeedback.txt

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

It is pointless including anything else in your folder abcd12 (in particular, 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. 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). This will give you information about whether your files are correctly formatted. As I state above, the output from this validation program (run immediately before hand-in) needs to be returned in 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). There are *strict instructions* within skeleton.py that you should read and follow. The definitive list of algorithms and tariffs is supplied to you 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 the file changes and you can download it again from ULTRA).

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. Note also that validation does not check for everything that you should have done and so is not definitive. For example, if you amend bits of `skeleton.py` that I expressly tell you not to then validation might not pick this up (it does not catch absolutely everything); furthermore, it might tell you that you have amended bits of `skeleton.py` but these amendments might have no impact on correctness. 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 will be fine.

Please be aware that ***you are not allowed to import non-standard modules*** 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 and you will receive 0 marks for anything associated with this implementation.

One final 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. You may artificially halt the execution and return the best tour found up until this point. However, take care that your implementation terminates within one minute. If you have a timer that decides to kill execution after a certain amount of time then you should not choose this time limit to be one minute but something less as your code still has work to do to extract the best tour found and prepare the tour file. ***I will simply 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 Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz 1.9 GHz and using Python 3.8 within Windows 10. Please ensure that you respect this in your codes.

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.

## FAQs

- Student: *Why do you insist on Python*?
- Iain: Every student has completed Computational Thinking and so can program in Python; this yields a 'level playing field' when it comes to implementation. Also, restricting to Python-only leads to fairer marking.
- Student: *Why do you take the maximum tariff from two 'correct' implementations as the 'sophistication' mark*?
- Iain: On occasion, a more sophisticated algorithm may give worse results than a more elementary algorithm. I want to reward both the quality of the tours that you find and also your accomplishments in coding up a more difficult algorithm. With things set up as above, I encourage you to code up a more sophisticated algorithm without harming your chances of getting good tours.
- *So I guess that's why you take the best overall tour found when you give the 'basic quality' mark*? *So that we don't harm the quality of the tours we find if we choose to implement a more sophisticated algorithm*?
- Iain: Correct!
- Student: *And I suppose you ask us to implement two algorithms and to experiment so that you can assess both our understanding of the algorithms in the course, through our capacity to code them up, along with our ingenuity and deeper understanding in obtaining good tours*?
- Iain: Correct again! If all I asked you to do was to produce basic implementations of two algorithms and produce some half-decent tours then there wouldn't be much of a challenge (nor fun) in that. So, I would like you to experiment. I understand that some of you will have more time and inclination for this than others so I ensure that if all you do is produce basic implementations and some half-decent tours then this will suffice to pass the coursework, which I think is fair. But I'm also giving you the opportunity to show me what you can do and be rewarded for it. In the past, many students have enjoyed the challenge of producing good tours.
- Student: *Why do you only give us a 'correctness' mark if both basic implementations give tours on your secret city files*?
- Iain: I think it is reasonable that your two basic implementations should both be correct. Don't you agree? My definition of 'correctness' is not exactly challenging! (Though having said that, please

ensure that your implementations terminate on my secret city files within the one minute time bound as detailed above.)

- Student: *Hmmm ... I think that we need to be careful when choosing our algorithms as if one of them is incorrect then we lose our 'correctness' mark. Furthermore, if we choose a high-tariff algorithm and a low-tariff algorithm and our high-tariff algorithm turns out to be incorrect then not only do we lose our 'correctness' mark but our 'sophistication' mark will be low too.*

- Iain: You would be wise to bear this in mind when you are choosing the algorithms that you intend to implement. A simple calculation will show you that a high-tariff algorithm being incorrect can decimate the overall mark you receive for this assignment. Please think very carefully about the algorithms you choose to implement (and their termination on my secret city files within one minute). There are more dangers with implementing a high-tariff algorithm, in terms of getting a smooth-running and correct implementation, than a low-tariff one but this needs to be considered not just against the mark rewards but also against the quality of tours overall. A wise course of action might be to implement more than two algorithms as the course progresses and eventually dispense with the additional one(s), but this is up to you.

- Student: *For the 'quality' mark for a tour, is this obtained by a relative comparison of how my tour compares to others obtained from the same algorithm or across all algorithms*?

- Iain: The 'quality' mark for a tour is obtained relative to all other tours for that city file, irrespective of the underlying algorithm.

- Student: *Why do you not just ask us to return the best tour we have found by whatever algorithm for each of the* 10 *city files rather than one tour for each algorithm*?

- Iain: I get to see the profile of tours you have produced for each algorithm. This allows me to have confidence that the tours you have supplied to me are indeed produced by the implementations stated, as I know (roughly) how different algorithms should perform; in fact, I automatically analyse *all* student tour files supplied to me and I can easily see when a claimed implementation is 'out of line' or when two students' tours are identical or very similar. Also, I might run your basic and enhanced implementations on the 10 city files to provide me with a sanity check that your codes and tours are what you say they are, though I'll probably only do this if I am suspicious!

- Student: *Are you often suspicious*?

- Iain: It doesn't really matter whether I am or I'm not as all student submissions are automatically compared against each other and the data I obtain (which is substantial) alerts me to problematic submissions. As I said, I analyse all tours submitted per algorithm and I also compare all student implementations against each other, on a pairwise basis, using plagiarism software that is specifically designed to detect plagiarism in Python programs (the plagiarism software that I use is very good and has been developed by academic researchers). In fact, I also compare student codes with codes from previous years and with other codes that might be publicly available. I should add that each year, the city files are randomly permuted so that tours from previous years are useless!

- Student: *Do you ever obtain discrepancies*?

- Iain: Unfortunately, yes. Last year, my analysis flagged some concerns and 14 students were found guilty of plagiarism, either by colluding with each other or by using programs that were implemented by students in previous years or available on the Web (the previous year it was 7 students). My advice to all students is

  ***DO THE WORK YOURSELF AS OTHERWISE YOU WILL BE FOUND OUT!***

  Do not be tempted to cut-and-paste from codes that are not your own as it is very difficult to avoid plagiarism (I have a personal library of codes that I have found on the Web and these codes are used in my plagiarism checks). Also, even if you cite your source then you will almost surely be deemed guilty of plagiarism as the whole point of this assignment is to assess *your* capability rather than someone else's. There is simply no need for you to consult other people's work: sufficient pseudo-code for all algorithms is supplied in the lectures.

- Student: *Why do you ask us to return our validation feedback*?

- Iain: This is new for this year. In the past, some students didn't bother validating and as a consequence, because they had tampered with something they shouldn't have or not supplied information they should have, their codes wouldn't run. Kind person that I am, I (painstakingly) fixed their submissions when I could and fined them small numbers of marks. In fact (and unbelievably), 48 students were fined last year for one thing or another. I will not be so kind in future. Henceforth, if a student's codes do not run, no matter what the reason, these codes will simply be deemed incorrect. Returning the validation feedback focuses students' minds on ensuring that their

codes will run. If a student's codes run but they haven't submitted their validation feedback then they will be fined!

- Student: *Can I change the subject and ask about enhancement?*
- Iain: Of course. Shoot!
- Student: *First, the basic implementation. Is it the case that you are not too fussy as to the nature of a basic implementation and that so long as the implementation is of a standard version of the algorithm in question, this will be fine*?
- Iain: Yes. Any implementation that follows a well-understood high-level structure of the algorithm in question will suffice (so, for a brute-force search, for example, any implementation that checks all possible tours is deemed to be a brute-force search). Any bad design decisions you make in your implementation will only harm: (a) whether your code terminates; or (b) the quality of your tours. Again, I will code-check implementations if I feel I have to. The easiest thing to do is to use the pseudocode supplied in lectures. If you diverge from the pseudocode in the lectures then you should explain how your algorithm works using pseudocode, natural language, etc., in the proforma and the reason for divergence (I say a bit more about this below).
- Student: *Can we take an algorithm from the course that enhances another algorithm from the course as our enhancement*? *For example, can we take Elitist Ant System as an enhancement of Ant System*?
- Iain: No. You should regard any algorithm from the course as a 'basic' algorithm and you need to enhance any 'basic' algorithm in an original way. Your two chosen algorithms must have different codes from the algorithm codes and tariffs file.
- Student: *What about if I implement, say, a genetic algorithm but its structure is different to the one in lectures*? *In fact, what about if I cut-and-paste some genetic algorithm Python code I found on the Web*?
- Iain: It goes without saying that I want you to develop your own code (though I have just said it to emphasise the fact!). If you choose to implement a genetic algorithm, say, with a structure different to the one presented in lectures then, as stated above, you should explain the divergence in the proforma and also the reason why you have chosen to diverge. This reason should be for experimental reasons, e.g., 'I implemented a genetic algorithm of slightly different structure to the one in lectures as this allowed me to introduce experimental variations that I couldn't do otherwise.' An illegal reason is 'I found some

genetic algorithm code on the Web and the structure was different to the algorithm outlined in lectures.' ***Do not cut-and-paste code from the Web, even if you cite your source! As I explain above, I use plagiarism detection software and you will be found out!*** I will regard cut-and-paste code as plagiarism even if the source is cited. I don't want to stop students starting to implement algorithms before they have been covered in lectures but if you do this then you should bear what I have said above in mind.

- Student: *Where do you draw the line between enhancing a basic algorithm and writing an entirely different algorithm*?

- Iain: Drawing the line is easy: if your enhanced implementation is not building on your basic implementation in an obvious sense (look at the codes) or does not follow the general structure of the underlying algorithm then it is an implementation of a different algorithm (which is not allowed). If I feel that I am being deliberately misled or lied to in the proforma then I will regard this as cheating.

  Also, on a related note, I would prefer that enhancements are algorithmic enhancements rather than implementation ones. For example, you might claim that your enhancement is to speed up the implementation in some way (such as using multiprocessing). This is not really what I am looking for: this is a course about algorithms, not (Python) implementations and you should focus on algorithmic enhancements. This does not mean to say that you should not speed up your implementations; for one thing, doing so will generally mean a more extensive search and (hopefully) better results. It's just that you won't obtain enhancement marks for doing this.

- Student: *Is it the case that 'enhancement' marks will be awarded using what we say in the proforma*?

- Iain: Yes. I will award 'enhancement' marks according to what you say in the proforma. You should describe the enhancements you made and any other relevant information within the context of these enhancements. Do not expect me to delve into your code to work out what you have done; on the other hand, do expect me to look at your code to verify that you have actually done what you say you have done. Clearly commented and readable code will be in your own best interests. If I cannot *quickly* verify that your enhancements are what you say they are then I'll just award a mark of 0. It is your job to be as clear as possible. The 'enhancement' mark will be determined by the depth, intricacy and novelty of your enhancements, irrespective of the quality of the tours produced, which will be rewarded through the 'enhanced quality' mark. A high number of enhancements does not

necessarily lead to a high 'enhancement' mark. If you use a research paper as the basis for your enhancement then please ensure that you reference the paper *in full*; that is, so that I can use your reference to find and consult the paper. Otherwise, your enhancement might be regarded as plagiarism.

- Student: *Suppose that I do lots of experimentation and settle on the best variation to hand in. Will I receive credit for the experimentation that I did but didn't hand in as my enhanced algorithm*?

- Iain: It is up to you to convince me in the pdf and in your codes that you did indeed undertake the experiments you say you did. Apart from proper descriptions in the pdf, one way of doing this would be to include other experimentation code in your enhanced algorithm but commented out (even better if you tell me in the comments how to comment it back in and run it for myself).

- Student: *Is it the case that in order to get 'enhanced quality' marks, the tours produced by my enhanced implementation must be better than those produced by my basic implementation*?

- Iain: Yes, this is the case. I will define a minimum amount by which an enhanced tour has to be better than a basic tour, for each of my secret city files (these amounts will be decided when all assignments have been submitted and will be internal to the marking process but will be fair and reasonable).

- Student: *What if we have worked hard to fine-tune our implementations to perform really well on the* 10 *city files but when you run them on your secret city files, they don't do so well*?

- Iain: There is a small chance that this might happen but if your enhanced implementation improves things across many of the 10 city files, it is likely that it will improve things on my secret city files too. Also, the 'enhanced quality' mark is relatively small. Similarly, if your algorithm is randomized then it might not do so well on the solitary run of it that I undertake. Such is life! However, a decent randomized algorithm should perform well almost all of the time and for a good randomized algorithm to go wrong on one run, you'll have to be extremely unlucky.

- Student: *If I restrict the run-time of my implemented algorithms but the tour produced by the enhanced implementation is worse than that produced by the basic implementation then should I return the basic implementation as the enhanced implementation*? *That is, should the enhanced implementation always produce the tour with the shortest length*?

- Iain: Your enhanced algorithm should be the basic algorithm with additional enhancements. If your enhanced implementation produces a worse tour than the basic implementation (possibly when forced to terminate within one minute) then that creates a problem for you (as you will receive no enhanced quality mark). In particular, this doesn't say much about the quality of your enhancements!

- Student: *I don't fully understand the timeline for the assignment. The high-tariff algorithms are not covered until the end of the term and so what am I supposed to be doing in (and outside) AI Search practicals if I have to wait to see the high-tariff algorithms, which are the ones I want to implement*?

- Iain: Of the two algorithms you implement, only the one with the highest tariff will contribute to the 'sophistication' mark. As I explain above, just because an algorithm is high-tariff does not mean to say that it will provide the best tours. It is up to you as to how you manage your time as regards the assignment but personally I would be developing basic implementions of some of the algorithms as I go, looking for algorithms that give good tours (and forgetting about implementations that don't provide good tours). Of course, doing this gives you a deeper understanding of the algorithms and so better scope to enhance them and obtain better tours. Alternatively, you might wait until you have seen all the algorithms before starting to implement. If you are managing your time in this way (though, as I have said, this would not be my choice) then clearly you would be using the practical sessions to do other things, possibly on other modules (but you would lose any potential help that the demonstrators could give you in practicals). If you wait until the final lecture before starting to implement then that still gives you plenty of time to undertake the assignment (but this would definitely not be my approach). The take-home message is: you should manage your time properly and plan in advance how you intend to work on all of the modules that you are undertaking this term. As I have repeatedly said to my children (so much so that they know it is coming and simply roll their eyes): failing to prepare is preparing to fail!

- Student: *Can you give me some illustrations of the different mark awards depending upon what sort of a student I am*?

- Iain: OK; here are some illustrations. Let's suppose that the maximum: sophistication mark is 10; correctness mark is 4; enhancement mark is 6; basic quality mark is 8; and enhanced quality mark is 2 (though, as I said earlier, these marks may have different values).

- Student A: Maybe you are hard-pushed and will not have the time nor inclination for experimenting but correctly implement a reasonably sophisticated algorithm at tariff 8 and a less complicated algorithm at tariff 6; so, your sophistication mark will be 8 and your correctness mark will be 4. The lack of experimentation means: that you get an enhancement mark of 0; and that you only obtained moderately good tours overall. Perhaps you get a basic quality mark of 5 and so an overall quality mark of 5. You would score 17/30 = 57% (a very solid lower-second mark).
- Student B: Maybe you are struggling and cannot correctly implement two algorithms but get a basic tariff-6 algorithm working, though the tours produced are not very good, resulting in a quality mark of 4. You would receive a sophistication mark of 6, a correctness mark of 0 and an enhancement mark of 0. Your total mark would be 10/30 = 34% (a fail mark).
- Student C: maybe you are like Student A but you are inclined to experiment. Your enhanced implementations are correct and reasonably innovative; so you obtain an enhancement mark of 4. The tours of the secret city files produced by your enhanced implementations produce a modest improvement over those produced by your basic implementations and you obtain an enhanced quality mark of 1. So, you would score 22/30 = 73% (a first-class mark).

The moral of the story? Show ambition with your implementations and experiment!

- Student: *I'm pretty ambitious and want to implement an algorithm not covered in the lectures. Can I do this*?
- Iain: Yes, ***but you must obtain explicit permission from me!*** First, take a look and see if your algorithm already has a tariff allocated in the text file `alg_codes_and_tariffs.txt`. If so then make sure that the algorithm you are thinking of really is the algorithm mentioned in the file. You need to email me for clarification so that you don't misinterpret things. If your algorithm does not appear in `alg_codes_and_tariffs.txt`, email me with a reference as to the algorithm you wish to implement and, after consideration, I'll give you a tariff and add it to the file. Either way, I will explicitly tell you in an email whether or not you can implement the algorithm under discussion. ***You cannot implement any non-standard algorithm without this consent!*** In the past I have refused consent to students because the proposed non-standard algorithm was too complicated,

because I did not feel that they had the knowledge or capability to implement the algorithm in question, because they clearly did not have a full understanding of the algorithm in question or because the algorithm was outside the spirit of this sub-module. Remember: if you receive consent to implement a non-standard algorithm then you need to provide both a description of it in the proforma (using pseudocode and natural language) and also a full reference of your source. There is additional space in the proforma that can be used when a non-standard algorithm is implemented.

- Student: *By the way, how do we know when we have found an optimal tour for some city set*?

- Iain: Almost always you don't. I don't know what the shortest tour is for some of the city sets and in the real world, we almost always don't know this either. The point of using heuristic algorithms such as the ones we develop is that we don't have efficient exact algorithms but we need a solution that is as good as we can make it and so that this solution is produced relatively quickly. That's why we turn to heuristic methods. It will be up to you to decide when you stop looking for a better solution to some city set.

- Student: *Why do you not allow us to import non-standard modules into our Python codes*?

- Iain: For three reasons. First, many of you will have only been programming with Python for just over a year and so the practice of coding up basic data structures will do you good. Second, you simply don't need additional modules to cope with the algorithms that you will be implementing. Third, there may exist some wacky modules out there with ready-made implementations of the algorithms we are working with, which defeats the object of the assignment!

- Student: *How fussy are you that we follow the guidelines stated here and in the programs that you supply to us*?

- Iain: I am extremely fussy! ***I simply refuse to go through your submissions so as to correct mistakes you have made by not following the guidelines, no matter how trivial these mistakes might be. This is non-negotiable.*** If you submit material without bothering to validate it with the (painstakingly-prepared) program `validate_before_handin.py` then you only have yourself to blame. Last year, about 30% of students submitted codes that either had not been properly validated or were such that the skeleton code that I expressly asked them not to change had been changed. The marks these students were awarded were *severely* affected by

their errors with some being awarded 0 marks. I have to admit that I was shocked that so many students simply did not follow these clear instructions.

***VALIDATE BEFORE HAND-IN!***

If you don't validate before hand-in then you will be fined!

- Student: *What should I do if when I validate my files before handing them in, I get a fatal error and can't understand why*?
- Iain: There are one or two common pitfalls. Make sure that when you download the city files from ULTRA and put them in a folder, you don't inadvertently also include garbage files in the folder (this can happen especially if you are using a Mac). Also, ensure that you download the tariff file `alg_codes_and_tariffs.txt` too and that all files are in their correct locations (see the comments at the start of the program `validate_before_handin.py`). As a last resort, email me.
- Student: *Will you answer questions we might have about the assignment*?
- Iain: Yes, of course I will. However, if you ask me a question whose answer is available in this document or the programs I supply to you then do not expect more than a very brief answer alerting you to this fact; indeed, if I think that you haven't even bothered to read the material I have supplied to you then this might irritate me so much that I don't even bother replying to you! In any case, not getting an answer is an answer in itself and you'll know what to do!
- Student: *I have some final questions related to the TSP. Is it alright if I quickly ask these*?
- Iain: (Sigh.) Go on then.
- Student: *For the TSP, is there a distance given for every pair of cities*?
- Iain: Yes. You should regard an instance of the TSP assignment as a weighted complete digraph where the vertices are the 'cities' and where there is a directed edge from every 'city' $x$ to every distinct 'city' $y$ with the weight being the given 'distance'. Note that it might be the case in general that the distance/weight from city/vertex $x$ to city/vertex $y$ is different from the distance/weight from city/vertex $y$ to city/vertex $x$. Also, the instances might not correspond to real sets of cities with real (Euclidean) distances; they might originate from

an entirely different application or even just be randomly generated. Finally, we are always looking for a shortest length tour where any tour should contain each city/vertex exactly once.

As a further point of clarification, the city files that I supply to you for the assignment are, in fact, symmetric in that the distance from city $x$ to city $y$ is equal to the distance from city $y$ to city $x$ (as it happens, the secret city files are symmetric too). You will see from `skeleton.py` that I (very helpfully!) unpack a city file on $n$ cities into a full $n \times n$ distance matrix for you to use called $dist\_matrix$. You should not assume that any city file is symmetric as this restricts the general applicability of your implementations: if you need to know the distance from city $x$ to city $y$ then look up $dist\_matrix[x][y]$; and if you need to know the distance from city $y$ to city $x$ then look up $dist\_matrix[y][x]$. The take-home message is that your codes should assume nothing about the city files' distances.

Another point of clarification is that, for us, any city-to-city distance is always a non-negative integer but might be equal to 0. Of course, this means that it is possible (and perhaps preferable) to move between the two cities in question at a cost of 0.

- Student: *Some TSP algorithms, like Christofides' Algorithm, for example, are specifically designed for TSP instances that satisfy the triangle inequality. Can we assume the triangle inequality*?

- Iain: No. You should not assume anything at all about the instances of the TSP given to you in the assignment. If you want to know whether an instance of the TSP satisfies the triangle equality then you can easily write some code to check this. You might still consider implementing an algorithm like Christofides' Algorithm even if your TSP instance does not satisfy the triangle inequality: the algorithm will still provide a tour but any performance result requiring that the instance should satisfy the triangle inequality will not necessarily hold.

- Student: *So that I might be able to get a rough idea of how well my implementations will perform when run on your secret city files, can you supply a city file of size about* 100?

- Iain: I don't have city files readily to hand. However, you can easily randomly generate your own to undertake your tests with. I see things like this as wrapped in with all the experimentation I request of you. A word of caution: even if your implementation terminates within a minute for your randomly-generated 100-city city file, it might not do likewise on my secret city files (for the reasons I highlighted above).