# AI Search Assignment

This is the assignment for the sub-module *AI Search* of the module *Artificial Intelligence*. The hand-out date is 19th October 2020 and it is to be completed and handed in **via DUO** by **2 p.m.** on **15 January 2021**.

******** 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. 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 set 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 ***one-page proforma*** (a pdf document) that briefly describes the enhancements you have made in your enhanced implementations, in comparison to the basic implementations.

## 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. A list of FAQs follows.

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 of your basic implementations (`AlgAenhanced.py` and `AlgBenhanced.py`)

($d$) 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 marks 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, and the highest tariff is 10. However, ... your sophistication mark will be the maximum tariff from your correct basic implementations (the definition of 'correctness' follows and I'll say why in the FAQs).

- 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 set will have 48 cities and the other city set will have 90 cities. Your basic implementations need to be correct: when I run your basic implementations on my secret city sets, a legal tour is produced for both city sets. The correctness mark is 4 if both of your basic implementations are correct and 0 otherwise (I'll say why in the FAQs).

So, for example, if you have only one correct basic implementation then your sophistication mark will be 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).

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 you experiment, the more marks you'll receive).

- The maximum mark you can be awarded for the enhancement of a basic implementation is 3; so, the maximum enhancement mark in total is 6 (of course, if you only supply one enhanced implementation then the maximum enhancement mark you can be awarded is 3). 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 marks.

Your enhancement mark will be derived by the commentary and explanation you submit in the one-page proforma together with a code inspection. You should ensure that you adhere to the guidelines as regards the one-page proforma, explained in the template I supply to you, as if you fail to respect these guidelines then I will consider the one-page proforma to be invalid and you will 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 distinct 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!

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 sets, 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 sets (regardless of whether this is via an implementation of AlgA or of AlgB; I'll say why in the FAQs).

- The maximum basic quality mark available is 8. This mark will consist of 10 components, one for each set of cities, and reflect how good your tours are relative to tours produced by others.

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 not more than 1 mark. So, the maximum overall enhanced quality mark is 2. Of course, to be awarded an enhanced quality mark for some algorithm, you need that both of your implementations are correct (I'll say why the enhanced quality mark is relatively small in the FAQs).

When I compare a basic implementation with an enhanced implementation, please ensure that the parameters are identically set, e.g., if your AlgA is a genetic algorithm then you should ensure that the number of iterations, mutation probability, size of population, and so on, are identically set in `AlgAbasic.py` and `AlgAenhanced.py`; 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 will consider your enhanced implementation to be incorrect and so it will not be awarded an enhancement mark or an enhanced quality mark.

In summary, the possible marks for each category follow:

- sophistication: the maximum obtainable is 10 but this 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 mark of 4 is awarded, otherwise a mark of 0
- enhancement: there is a maximum of 3 marks available for each correct enhanced implementation; so, the overall 'enhancement' mark is at most 6
- quality: the maximum basic quality mark available is 8 and the maximum enhanced quality mark available is 2; so, the overall quality mark is at most 10.

Consequently, the maximum mark achievable is 30.

## 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 be:

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

  which are the basic versions 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).

It is pointless including anything else in your folder abcd12 (in particular, the folder of city files). On receiving your folder, the first thing I do is to automatically delete *everything* with a name different from the list of names above.

## 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. There are 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 place the new version on DUO).

You need to ensure that ***all of your Python implementations can be executed in command-line mode by supplying the city file, e.g., via the command***

- `python AlgAbasic.py AISearchfile012.txt`

If I cannot execute your implementations as above then I will not be able to run your codes and you will lose marks. Using `skeleton.py` will enable such a command-line execution (take a look at the instructions within the comments). You should ensure that your own code does not use the list of arguments available via `sys.arg` at all.

Note that you are not allowed to import any non-standard modules into your Python codes (I explain why in the FAQs). You can see the modules that you can import if you look in `skeleton.py`. If you submit an implementation that uses a non-standard module then the implementation will be deemed to be incorrect.

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.*** 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 feedback file) so that you can fix it.

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 48 and 90). 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. 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 randomized choices, the complexity of the cross-over, whether you print data during the execution 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. By the way, the version of Python that I have installed is Python 3.8 and I will be executing your programs under the Windows 10 operating system. Please ensure that you respect this in your codes.

## 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 sets*?
- 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!
- 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 sets 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 analyse all student tour files supplied to me and I can easily see when a claimed implementation is 'out of line'). Also, I might run your basic and enhanced implementations on the 10 city sets to provide me with a sanity check that your codes are what you say they are (though I'll probably only do this if I am suspicious!).

- Student: *What if we have worked hard to fine-tune our implementations to perform really well on the* 10 *city sets but when you run them on your secret city sets, 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 sets, it is likely that it will improve things on my secret city sets too. Also, the 'enhanced quality' mark is relatively small.

- 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.

  - 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 TSP 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 moderately innovative; so you obtain an enhancement mark of 3. The tours of the secret city sets 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 21/30 = 70% (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 course. Can I do this?*

- Iain: Yes. 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, make sure that the algorithm you are thinking of really is the algorithm mentioned in the file. It is best 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 and I'll give you a tariff and add it to the file.

- 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.

- 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.