Assignment 2

COMP9058 - Metaheuristic Optimisation

November 24, 2025

Due date:
Assignment should be submitted to Canvas in a single pdf by **Sunday, Dec 14th, 23:59**. As per MTU regulations, submitting within 7 days of the deadline will result in a 10% penalty, between 7 and 14 days late will result in a 20% penalty, and later than 14 days after the due date will result in a 100% penalty applied.
The pdf MUST be named *Lastname_StudentNumber_MH2.pdf*. Similarly the folder containing the code to be zipped MUST be named
Lastname_StudentNumber_MH2.py.
You need to write a report with a comprehensive description of the experiments and an evaluation (and analysis) of the results. Use tables and figures where appropriate. As always, the random number generator **must be seeded to your student number** for your experiments.

1 Assignment Description

1.1 SAT - [70 Marks]

The Propositional Satisfiability Problem (SAT) can be represented by a pair F = (V, C) where, V indicates a set of Boolean variables and C a set of clauses in conjunctive normal form (CNF). The following formula shows a SAT example described by means of the CNF. $F = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3)$
where: $\vee$ represents the logical *OR* operator, $\wedge$ represents logical *AND*, and $\neg x_i$ is the negation of x_i.
Given a set of clauses $C_1, C_2, \ldots, C_m$ on the variables $x_1, x_2, \ldots, x_n$, the satisfiability problem is to determine if the formula

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

is satisfiable. That is, is there an assignment of values to the variables so that the above formula evaluates to true? For instance, a potential solution for F would be x_1 =True, x_2=False, x_3=False, x_4=True. x_1 satisfies the first clause, x_4 satisfies the second clause, and $\neg x_3$ satisfies the last clause.

Local Search for SAT

Algorithm 1 describes a traditional local search algorithm for SAT solving. It starts with a random truth assignment for the variables in F, and the the local search algorithm is depicted in lines (3-9) here the algorithm flips the most appropriate variable candidate until a solution is found or a given

Algorithm 1 Local Search For SAT (CNF formula F, Max-Flips, Max-Tries)

```
1: for try := 1 to Max-Tries do
2:    A := initial-configuration(F).
3:    for flip := 1 to Max-Flips do
4:       if  A satisfies F then
5:          return A
6:       end if
7:       x := select-variable(A)
8:       A := A with x flipped
9:    end for
10: end for
11: return 'No solution found'
```

Figure 1: Outline of Local Search SAT approach

number of flips is reached (MaxFlips), after this process the algorithm restarts itself with a new (fresh) random assignment for the variables.
GSAT selects the variable with maximum net gain. GWSAT adds a random walk step, with probability w_p select at random a variable involved in at least one currently unsatisfied clause, otherwise select the variable with maximum net gain.

Implementation - [30 Marks]

The code attached **must** be used as a basis for your implementation. Note that the code contains a number of heuristics, including GSAT, GWSAT, HSAT, and WalkSAT.
You must implement the following variants for testing:

1. *hsatTabu*: Add tabu search to basic hsat, with aspiration criteria of improving on best solution found so far in this search attempt (i.e. not including from previous restarts). Note the data structure *lastFlip* maintains the iteration number of the last iteration each variable was flipped.
2. *grimesWsat*: Heuristic which does the following:
   (a) If zero damage variable, zero damage variable step (randomly select variable from variables with positive gain > 0, and negative gain $= 0$, if such variable exists
   (b) Random walk step with probability wp: choose randomly from variables involved in at least one unsatisfied clause
   (c) Otherwise randomly choose unsat clause, choose variable with maximum net gain, breaking ties randomly
3. *grimesHsat*: Heuristic which does the following:
   (a) If zero damage variable, zero damage variable step (randomly select variable from variables with positive gain > 0, and negative gain $= 0$, if such variable exists
   (b) Age walk step with probability wp: choose variable involved in at least one unsatisfied clause that was flipped the longest ago
   (c) Otherwise randomly choose unsat clause, choose variable with maximum net gain, breaking ties randomly

(Note that heuristics 2 and 3 only differ in their step (b).)
Additionally you must change the code such that the stopping condition is the number of **successive non-improving flips**, instead of total number of flips.

The main file must follow the same naming convention (Lastname_StudentNumber_MH2.py) and be created such that it can be run from the command line with parameters passed in for each parameter as follows:

`python Grimes_R1234_MH2.py inst_dir alg nRuns nRestarts nFlips` w_p `tl`

where

- inst_dir: the full path to the directory containing the uf150* instances (from zip file on Canvas)
- alg: one of { gwsat, hsat, walksat, hsatTabu, grimesHsat, grimesWsat}
- nRuns: the number of runs of the algorithm to perform
- nRestarts: is the number of restarts
- nFlips: is the number of successive iterations without an improving move before restarting
- w_p: is the random walk probability
- tl: is the tabu length

The default settings are given as follows (using the command line arguments listed above in that order):

```
python Grimes_R1234_MH2.py myfolder/ gsat 10 50 500 0.1 10
```

Note, here gsat does not use w_p so it will just ignore this parameter value in the code, similarly algorithms that don't use tabu will ignore the tabu parameter. **If your code doesn't run in this exact format, marks will be lost. You MUST use the template code provided and only make changes where needed, it is your responsibility to ensure that the file you submit runs in the manner specified.**

Evaluation - [40 Marks]

You must evaluate your implementation of the methods (hsatTabu, grimesHsat, grimesWsat) and three of the given approaches in the code (gwsat, hsat, walksat) on ten SAT instances from the uf_150 lib and discuss the results (this is setup in the code to use ten instances based on your student number). Further tests involving the parameter settings of (nRestarts nFlips) combined in similar method of varying generations and population in GA (i.e. vary such that their product is constant) , and of w_p (for gwsat), and tl (for hsatTabu) should be performed. There should be at least 10 runs per experimental setup.
Overall experiments:

- gwsat (implemented)
- hsat (implemented)
- walksat (implemented)
- hsatTabu
- grimesHsat
- grimesWsat
- (nRestarts, nFlips) on hsat
- wp parameter on gwsat
- tl parameter on hsatTabu

Instances

You will test your algorithms on ten instances of the *uf150-645* SAT set from Canvas (note this contains 100 instances). The id of instances you choose must end in the last number of your student id. **Note the code you are given is already setup for this.**

```
for filename in listdir(directory):
    if filename.endswith("n.cnf"):
        satInst=directory+"/"+filename
```

Figure 2: The n in "n.cnf" is replaced with the last value of your student number

1.2 Runtime Distribution - [30 Marks]

You should compare the **runlength** distributions of the gwsat algorithm and walksat algorithm with 100 runs each on two randomly selected instances of the 50-variable problem set (there are 1000 instances in this set), generating the run**length** distribution plots. Note both heuristics are included in the code provided.

2 Submission, marking and academic integrity

2.1 Submission:

This assignment is due on Sunday, Dec 14th. You must submit the pdf and py files separately via the assignment and code submission portals on Canvas Your pdf **MUST** be named Lastname_StudentNumber_MH2.pdf (e.g. Grimes_Diarmuid_R001234567_MH2.pdf). Similarly the the python file MUST be named *Lastname_StudentNumber_MH2.py*.

2.2 Rubics:

Coding

	The algorithms are logically well designed and efficient without inappropriate design choices (e.g., unnecessary loops). Code is well commented.	The algorithms always works properly and meets the specification. Code is clean, understandable and well organized.	The algorithms works properly in limited cases.	The algorithms are incorrectly imple-mented.
SAT	(22-30 Marks)	(15-21 Marks)	(7-14 Marks)	(0-6 Marks)

Evaluation

	Excellent presentation, depth and insight analysis of the empirical results.	Good presentation of the results (e.g., describing the results with well structured tables).	Incomplete and/or unclear presentation of the results.	The results are inconsistent with the logic of the configuration/ operators.
SAT	(31-40 Marks)	(21-30 Marks)	(11-20 Marks)	(0-10 Marks)
RTD	(22-30 Marks)	(15-21 Marks)	(7-14 Marks)	(0-6 Marks)

2.3 Academic Integrity

This is an **individual assignment**. The work you submit must be your own. In no way, shape or form should you submit work as if it were your own when some or all of it is not. Any online source that is used must be cited, and a full citation given, e.g. do NOT give "stackoverflow", but the full citation, e.g. https://meta.stackoverflow.com/questions/339152/plagiarism-and-using-copying-code-from-stack-overflow-and-submitting-it-in-an-as. If you are unsure on whether something should be cited, general rule of thumb is to err on the side of caution and include the citation. You can also ask me via email
Collusion: Given how much freedom there is in the assignment, everybody's work will be different. It will be obvious if there is collusion. ***All parties to collusion will be penalized.***
Deliberate plagiarism: You must not plagiarise the programs, results, writings or other efforts of another student or any other third-party. Plagiarism will meet with severe penalties, which can include exclusion from the Institute.
Your report and code will be checked for signs of collusion, plagiarism, falsification and fabrication. You may be called to discuss your submission and implementation with me and this will inform the grading, any penalties and any disciplinary actions.