CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Metaheuristic Optimization (SAT)

Due date:

Assignment should be submitted to Canvas before 10AM **Tuesday Dec 11th**

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.

$$F = (x_1 \vee x_3 \vee -x_4) \wedge (x_4) \wedge (x_2 \vee -x_3)$$

Where $\vee$ represents the ***or*** Boolean connective, $\wedge$ represents **and,** and $-x_i$ is the negation of x_i. Given a set of clauses, C_1, C_2, ... , C_m on the variables x_1, x_2, …, 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 $-x_3$ satisfies the last clause.

Local Search

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 number of flips is reached (MaxFlips), after this

process the algorithm restarts itself with a new (fresh) random assignment for the variables.

As one may expect, a critical part of the algorithm is the variable selection function (select-variable) which indicates the next variable to be flipped in the current iteration of the algorithm. The WalkSAT algorithm introduces a diversification strategy in order to avoid local minimums. The algorithm starts by selecting, uniformly at random, an unsatisfied clause c and then picks a variable from c. The variable that is generally picked will result in the fewest previously satisfied clauses become unsatisfied, with some probability of picking a random variable from c.

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'
```

I/O Specification

In this assignment you will use the same I/O format as defined in the lab work of Week 8.

GSAT with Tabu Search (20 Marks)

In this part of the assignment you will extend the popular GSAT algorithm with Tabu search (without aspiration criteria). In this context, after a variable *x* has been flipped, it cannot be flipped back within the next *tl* steps (the tabu tenure tl is a parameter of your algorithm). For each step of the algorithm, the variable to be flipped is selected like in the basic GSAT, except that the choice is restricted to variables that are currently not Tabu.

Use the following parameters:

tl = 5 steps
1000 iterations per restart
10 restarts

Novelty+ (30 Marks)

Your Novelty+ implementation will work as follows:

1. Select an unsatisfied clause c (uniformly at random)
2. With some probability wp select a random variable from c, while in the remaining cases use Novelty for your variable selection process.

    Novelty: Identify the best V_{best} and second best variables V_{2best} in c (w.r.t. the minimization of the number of unsatisfied clauses). If V_{best} is not the most recently flipped variable in the clause, novelty will select this variable. Otherwise, V_{2best} will be flipped with some probability p and V_{best} with some probability 1-p.

Use:
wp=0.4
p (for Novelty+) = 0.3
100000 iterations limit or 1 minute
No restarts

TSP (35 Marks)

The iterative local search framework depicted in Algorithm 2 comprises two phases. First, in a local search phase, the algorithm improves the current solution until reaching a local minimum. Second, in the perturbation phase, the algorithm perturbs the current incumbent solution (s*) in order to escape from difficult regions of the search (e.g., a local minimum). Finally, the acceptance criterion decides whether to update s* or not. To this end, with a probability 5% s'* will be chosen, and the better one otherwise.

Algorithm 2 Iterated Local Search

1: s_0 := Initial Solution
2: s^* := LocalSearch(s_0)
3: **repeat**
4: s' := Perturbation(s^*)
5: s'^* := LocalSearch(s')
6: s^* := AcceptanceCriterion(s^*, s'^*)
7: **until** No stopping criterion is met

For this part of the assignment, you will implement the following local search algorithm for your iterative local search implementation (in lines 2 and 5).

Local Search algorithm

1. Select randomly an edge (*e*) from the current tour.
2. Make a 3-opt movement with all other edges (*e* + 2 other edges) of the tour and select the best tour therefore generated, If it is better than the current tour then make it the current tour and go to 1
3. Else. Stop

Perturbation phase:

Repeat 5 times

1. Select randomly 2 edges (i, j) from the current tour.
2. Perform a 2-opt move with (i, j) and update the current tour

Evaluation (15 Marks)

- RTDs: Analyse and compare the RTDs (with at least 100 executions) of GSAT/Tabu and Novelty+ for uf20-020.cnf and uf20-021.cnf.
- Analyse and compare the impact of two heuristic approaches for building the starting solution for the 3-opt algorithm (i.e., random tours vs. nearest-neighbor starting tours).
    - Stopping condition: 5 minutes per execution (at least 5 executions per instance)
    - Instances: inst-0.tsp and inst-13.tsp

Submission:

You must submit the following files (in a single zip file):

- All source codes.
- A Readme file, which briefly describes all submitted files. In the Readme file, you should also provide information about compiling environment, compiling steps, execution instructions, etc.
- A Sample file, which describes the test you have run on your program. Also describes any cases for which your program is known not to work correctly.
- Report document (pdf) with the evaluation section

Rubric:

	The algorithm is logically well designed and efficient without inappropriate design choices (e.g., unnecessary loops). Code is well-commented	The algorithm always works properly and meets the specification. Code is clean, understandable, and well-organized	The algorithm works properly in limited cases.	The Algorithm is incorrectly implemented.
GSAT & Tabu Search	(16 - 20 Marks)	(11 - 15 Marks)	(6-10 Marks)	(0-5 Marks)
Novely+	(24 - 30 Marks)	(13 - 23 Marks)	(6-12 Marks)	(0-5 Marks)
TSP	(25 - 35 Marks)	(15 - 24 Marks)	(7-14 Marks)	(0-6 Marks)

Evaluation:

You need to write a report with a comprehensive description of the experiments and an extensive evaluation (and analysis) of the results. Use tables and figures where appropriate.

	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
RTDs (SAT)	(7-10 Marks)	(5- 6 Marks)	(2-4 Marks)	(0-2) Marks
TSP	(4-5 Marks)	(2- 3 Marks)	(1-2 Marks)	(0-0) Marks