[271P Project Draft Design](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[**Team Name**](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[skynetUCI](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[**Team Members**](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[Wu, Yu-Jie, yujiew19, 12646408](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[Parvez, Mohammed Farhaan, parvezm, 48768989](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[Bharot, Vineet, vbharot, 88649968](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

[(Stochastic) Local Search (SLS) - Topic: 3-SAT](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/3-sat-local-search-Oa7yq)

1. Design the algorithm(s) and data structures

2. Present the algorithms using pseudo-code

3. Explain/describe the (pseudo-code of the) algorithms

4. Provide assessment/evaluation of the time/space complexity of your algorithms

[MAX-SAT Project, Additional Experiments (uchicago.edu)](http://people.cs.uchicago.edu/~pankratov/addl_exp)

[A Variable Neighborhood Walksat-Based Algorithm for MAX-SAT Problems (nih.gov)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4142167/)

[Algorithms for Maximum Satisfiability - with Applications to AI (helsinki.fi)](https://www.cs.helsinki.fi/group/coreo/aaai16-tutorial/aaai16-maxsat-tutorial.pdf)

https://www.hindawi.com/journals/tswj/2014/798323/

<https://www.hindawi.com/journals/tswj/2014/798323/alg1/>

https://www.hindawi.com/journals/tswj/2014/798323/alg2/

Simulated Annealing Max SAT

Schedule : maxTemp = 0.3 minTemp = 0.01 decay rate = 1/n(number of variables)

[[2003.02981] Learning Complexity of Simulated Annealing (arxiv.org)](https://arxiv.org/abs/2003.02981)

O(m^½) samples suffice to find an approximately optimal cooling schedule of length m.

function SimulatedAnneallingMAXSAT(clauses, initial, maxTemp, minTemp, decayRate) returns a maximum satisfying model:

Curr←initial

For stepNo=1 to ∞ do

temp←maxTemp\*e^(-stepNo\*decayRate)

If temp < minTemp then break

For i = 0 to n do

prob ← 1/(1+e^(-Δ(curr, i)/temp)

// random() procedure returns a real number (0,1) uniformly at random

If random() < prob then

Curr[j] ← 1 - curr[j]

If Value(curr) > Value(X) then X ← curr

From book

function SimulatedAnneallingMAXSAT(clauses, initial, maxTemp, minTemp, decayRate) returns a maximum satisfying model:

Curr←initial

For stepNo=1 to ∞ do

temp←maxTemp\*e^(-stepNo\*decayRate)

If temp < minTemp then break

For i = 0 to n do

prob ← 1/(1+e^(-Δ(curr, i)/temp)

// random() procedure returns a real number (0,1) uniformly at random

If Value(curr) > Value(X) then X ← curr

Else If random() < prob then

Curr[j] ← 1 - curr[j]

STATE SPACE: assignment of true/false to the variables(literals) involved in the CNF.

OBJECTIVE FUNCTION: maximization of the number of satisfied clause

NEIGHBORHOOD RELATION: using simulated annealing on random based on temperature.

Algorithm 3 provides pseudocode for simulated annealing for the unweighted MAX-k-SAT problem. Simulated annealing is a randomized algorithm that mimics to some extent the physical process of the same name. The algorithm chooses to flip a truth value of variable $ i$ with some probability related to $ \Delta(X,i)$ via the logistic function shown on line 9. The temperature parameter $ temp$ can be used to control how important $ \Delta(X,i)$ is in computing this probability. If the temperature is high, then the algorithm will be flipping variables almost at random with probability close to $ \frac{1}{2}$, no matter what $ \Delta$ is for that variable. If the temperature is low, then the algorithm will almost certainly be flipping variables with positive $ \Delta$ values and leaving variables with negative $ \Delta$ values untouched.

During the runtime of the algorithm the temperature is gradually decreased from high to low. The rule, by which the temperature changes, is called a temperature schedule. The reported good temperature schedule to use is $ maxTemp=0.3$, $ minTemp=0.01$, $ decayRate=\frac{1}{n}$, using the formula on line 4 to compute the current temperature. Intuitively, this allows the algorithm to explore the landscape of the objective function extensively at the beginning, giving slight preference to valleys with larger values of the objective function. When the temperature drops down, simulated annealing starts searching the landscape more thoroughly in some promising part of the landscape. This method keeps track of the best ovserved truth assignment and outputs it at the end of the procedure. The weighted MAX-k-SAT problem can be solved by exactly the same method with just a small modification to the computation of the $ \Delta$ function

**Input** : A CNF formula F

**Parameters** : Integers max\_duration in seconds, max-tries; noise parameter pnoise ∈ [0, 1]

**Output** : A maximum satisfying assignment α for F

**begin**

max\_number\_clauses\_satisfied = 0

**for** time ← current\_time to current\_time + max\_duration **do**

σ ← a randomly generated truth assignment for F

for j ← 1 to max-flips **do**

**if** σ satisfies F **then**

α ← σ

**return** α // success

C ← an unsatisfied clause of F chosen at random

**if** ∃ variable x ∈ C with break-count = 0 **then**

v ← x // freebie move

**else if** random(0, 1) < pnoise **then** // random walk move

v ← a variable in C chosen at random

**else**: // greedy move

v ← a variable in C with the smallest break-count

Flip v in σ

number\_clauses\_satisfied ← count number of clauses satisfied (σ)

**If** number\_clauses\_satisfied > max\_number\_clauses\_satisfied **then**

α ← σ

max\_number\_clauses\_satisfied ← number\_clauses\_satisfied

**end**

**return** α

STATE SPACE: assignment of true/false to the variables(literals) involved in the CNF.

OBJECTIVE FUNCTION: maximization of the number of satisfied clause

NEIGHBORHOOD RELATION: using simulated annealing on random based on temperature.

[IncompleteAlg-SAT-Handbook-prelim.pdf (cornell.edu)](http://www.cs.cornell.edu/~sabhar/chapters/IncompleteAlg-SAT-Handbook-prelim.pdf)

[Local Search for Hard SAT Formulas: The Strength of the Polynomial Law (princeton.edu)](https://www.cs.princeton.edu/~sixuel/others/aaai16_liu.pdf)

[Branch-and-Bound Depth-First-Search (BnB) - Topic: TSP](https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK)

1. Design the algorithm(s) and data structures

Input:

a complete weighted undirected graph representing the cities and path cost between them (if it is a non-complete graph, assigning an infinity large number to the non-existent path)

Explanation:

use an adjacency matrix to represent the graph since it is a complete graph. If there exists five cities, there will be a 5x5 matrix(or vector<vector<int>> graph) representing the graph. For each i, j, graph[i][j] means the cost of the path from i to j.

Output: the minimum cost of a possible route that visits each city exactly once and returns to the origin city

2. Present the algorithms using pseudo-code

<https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/>

Exact code in c++

<https://mathcourses.nfshost.com/archived-courses/mat-375-001-2015-fall/lectures/lec-28-travelling-salesman-problem.pdf>

<https://artint.info/html/ArtInt_63.html>

1: **Procedure** DFBranchAndBound(*G,s,goal,h,bound0*)

2: **Inputs**

3: *G*: graph with nodes *N* and arcs *A*

4: *s*: start node

5: *goal*: Boolean function on nodes

6: *h*: heuristic function on nodes

7: *bound0*: initial depth bound (can be *∞* if not specified)

8: **Output**

9: a least-cost path from *s* to a goal node if there is a solution with cost less than *bound0*

10: or *⊥* if there is no solution with cost less than *bound0*

11: **Local**

12: *best\_path*: path or *⊥*

13: *bound*: non-negative real

14: **Procedure** cbsearch(*⟨n0,...,nk⟩*)

15: **if** (*cost(⟨n0,...,nk⟩)+h(nk) < bound*) **then**

16: **if** (*goal(nk)*) **then**

17: *best\_path ←⟨n0,...,nk⟩*

18: *bound ←cost(⟨n0,...,nk⟩)*

19: **else**

20: **for each** arc *⟨nk,n⟩∈A* **do**

21: *cbsearch(⟨n0,...,nk,n⟩)*

22: *best\_path ←⊥*

23: *bound ←bound0*

24: *cbsearch(⟨s⟩)*

25: **return** *best\_path*

3. Explain/describe the (pseudo-code of the) algorithms

4. Provide assessment/evaluation of the time/space complexity of your algorithms

Reference:

<https://www.coursera.org/learn/advanced-algorithms-and-complexity>

<https://artint.info/html/ArtInt_63.html>

<https://apps.dtic.mil/dtic/tr/fulltext/u2/a314598.pdf>

<https://www.quantamagazine.org/computer-scientists-break-traveling-salesperson-record-20201008/>

[MAX-SAT Project, Additional Experiments (uchicago.edu)](http://people.cs.uchicago.edu/~pankratov/addl_exp)

[A Variable Neighborhood Walksat-Based Algorithm for MAX-SAT Problems (nih.gov)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4142167/)