Title Page

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

skynetUCI

[**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: MAX-SAT

**Maximum Satisfiability Problem**

**Definition:**

Consider an n-variable [CNF(Conjunctive Normal Form)](https://en.wikipedia.org/wiki/Conjunctive_normal_form) formula F with m-clauses C1, C2,...,Cm. For a truth assignment state σ ∈ {0, 1}n and i ∈ {1, 2, . . . , m}

So [Maximum Satisfiability Problem(maxSAT)](https://en.wikipedia.org/wiki/Maximum_satisfiability_problem) for F can be written as can be written as the following optimization problem over σ ∈ {0, 1}n:

minimize N(σ) =

Where Ui(σ) is defined as:

0 if clause i is satisfied by σ

Ui(σ) = ∀i ∈ {1, 2, . . . , m}

1 if clause i is unsatisfied σ

N(σ) ≥ 0 and only equals 0 if and only if all clauses of F are satisfied.

Here N(σ) is the **objective function**.

**STATE SPACE**: assignment of true/false(1/0) to the variables(literals) involved in the CNF. Here σ defines a particular state.

We explore the state space for a state which results in the minimum value for the Objective function.

**Approach Description:**

To explain our approach we first need to define the flipping operation of a variable and the definition of the break-count of a variable (v) with respect to CNF and state σ.

**Flipping a variable(v):** flip() =

if the current value of the variable is True(1), then flipping it makes it False(0) and if the current value is False(0) the flipping it makes it True(1).

**Break Count**: The Number of currently satisfied Clauses in the CNF which will become unsatisfied by σ if we flipped the value of variable v.

**Approach:**

To begin our search we start with a Initial Truth Assignment and

as this is an Optimization Problem then a **Greedy** Local Search move of flipping a variable in which the **maximum possible number of clauses, or with the least Break Count** is a good approach, but this suffers from Problems of Local Maxima/Minima and large plateaus.

To Counter these we introduce a **Stochastic element** into our solution by selecting a random variable to flip with a probability called noise (**Pnoise**).

To Widen our search of the state space we set an upper threshold limit on the maximum number of flips allowed in a search iteration(**max\_flips**)**,** after which we **reinitiate the initial assign,**  and start all over again.

To track the best truth assignment with the minimum value of the Objective function we maintain a global variable (α), and at each iteration we check whether the current assignment(σ) has a lower value of Objective function, if so then we make the current assignment the best assignment.

If we find an assignment with 0 Value of the objective function, then it means that the current assignment is the solution of the CNF, hence we stop searching and return the current assignment as a solution.

**Algorithm Explanation:**

**PSEUDO-CODE:**

**function** : maxWalkSAT

**Input** : A CNF formula F

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

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

**begin**

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

σ ← a randomly generated truth assignment for F // start fresh attempt

**for** j ← 1 to max-flips **do**

**if** N(σ) == 0 **then** //σ satisfies F then

**return** α // success

C ← an unsatisfied clause of F chosen at random

**if** ∃ variable x ∈ C with breakCount(σ, x) = 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 breakCount

flip(v, σ) // Flip v in σ

**If** N(σ) < N(α) **then** // N(x) is the objective function

α ← σ

**end**

**end**

**return** α

**end**

**function**  : breakCount

**Input**  : A CNF formula F

**Parameters** : a truth assignment σ, a variable x to flip to calculate break count

**Output**  : number of satisfied clauses of σ which becomes unsatisfied on flipping x

**begin**

before\_flip ← σ

flip(x, σ)

after\_flip ← σ

break\_count ← 0

**for each** clause(c) in CNF(F) **do**

**If** U(c, before\_flip) == 1 and U(c, after\_flip) == 0 **do**

break\_count += 1

**end**

**return** break\_count

**end**

**NOTE: Ui** and **flip** are defined in the previous sections

**EXPLANATION PSEUDO-CODE:**

Try to find an assignment α that minimizes Objective function N(x) or makes the Objective function N(x) absolute minimum i.e. 0, till the time runs out or we find a solution. For each attempt with a maximum of max-flips flips allowed, we start with a randomly generated truth assignment for F so that we explore wider state space.

At each iteration we check if the current assignment σ satisfies the CNF if it does then we return it. Else we pick one of the unsatisfied clauses C of F at random. Now from this clause try finding a variable with break-count 0 ie. flipping it does not make any of the previously satisfied clauses unsatisfied. If found then flip it and move to the next iteration.

If not found then from selected clause C with probability Pnoise perform a **random move** and select a random variable to flip and move to next iteration and with 1-Pnoise perform a **greedy move** which selects a variable from C with minimum break-count, flip it and move to next iteration.

After flipping a variable check if there is any improvement in the Objective function by comparing N(σ) < N(α). if it’s true then we’ve found a better assignment σ better than previous α, so we make σ(current assignment) the new α(best assignment)

If we run out of time then we return the α which is the best assignment we’ve found so far with the minimum Objective function value and maximum number of clauses satisfied.

**TIME and SPACE COMPLEXITY:**

As for n variables we have two values, true and false. And as maxSAT is a variation of k-SAT problem which has a complexity of O(2\*(k-1)/k)n [[1](https://homepages.cwi.nl/~rdewolf/schoning99.pdf), [2](https://www.researchgate.net/publication/228780993_A_Probabilistic_Algorithm_for_k_-SAT_Based_on_Limited_Local_Search_and_Restart)], hence this algorithm has **Exponential time complexity.**

As for the space complexity as this algorithm explores the state space in depth first search manner without any other data structure hence this has **linear space complexity.**

**Properties Analysis and Experiment Observations:**

Maximum number of flips allowed in an iteration with respect to an initial truth assignment (**Max-Flips)** is inversely proportional to the number of times we start fresh with a new initial assignment, which results in smaller state space explored.

But keeping max-flips extremely small to allow large state space search will also keep us from finding the best local solution.

So we need an optimum value for the max-flips, hence we performed the following experiments to get to an optimum value.

1. **Satisfiable CNF verification**
   1. **Custom All true Solution assignment:**

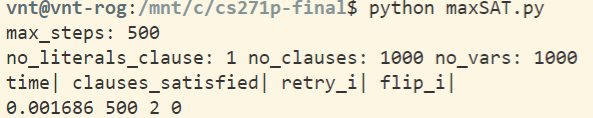
an all true n variable, n clause cnf with 1 variable in each clause like cnf = [(1,), (2,), (3,) .... (n, )]

* 1. **Custom Alternate true, false solution assignment:**

alternate true, false n variable, n clause with 1 variable in each clause assignment like: cnf = [(1,), (-2,), (3,).....(-(n-1), ) (n,)]

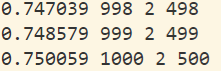
**Observations:**

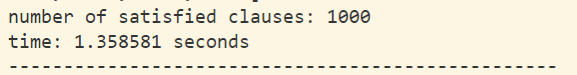
**Run # 1:**

****

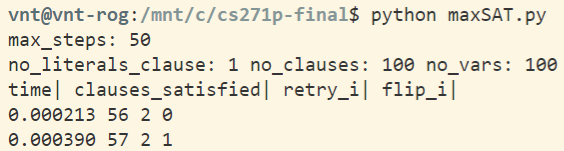
**………..**

**………..**

****

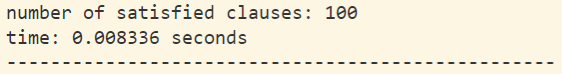
****

**Run # 2**

****

**………..**

**………..**

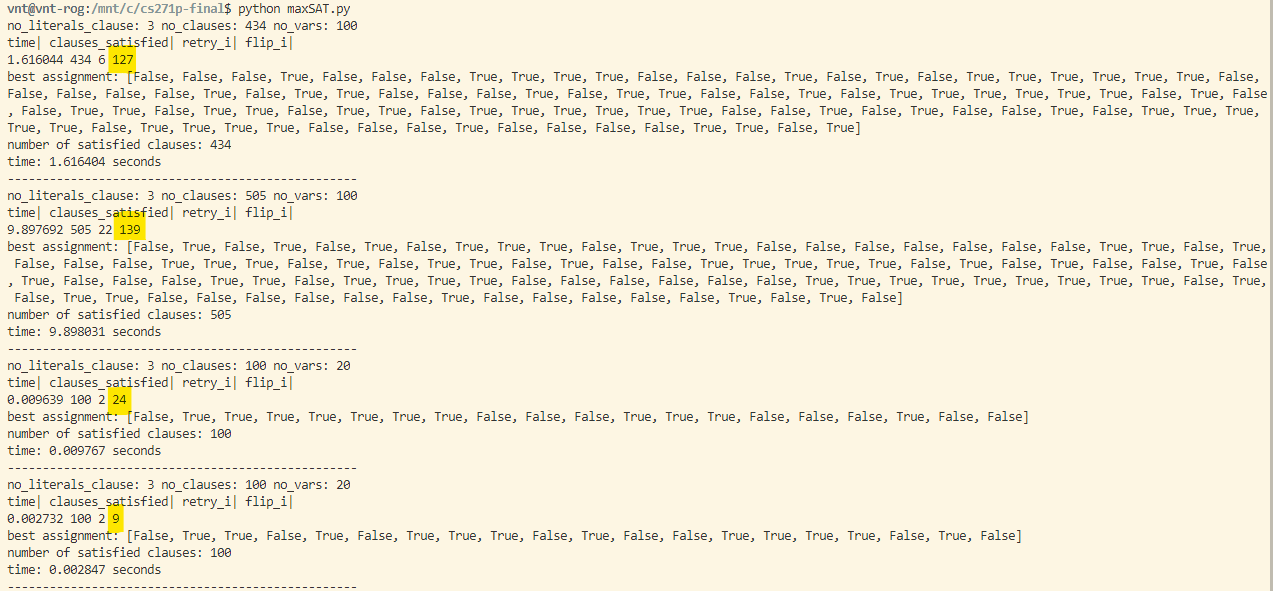
****

Here our experiments observed that our initial assignment always results in around half of the clauses satisfied, which always results in a solution being n/2 steps away from the solution, as there is always a free move available.

* 1. **Randomly generated satisfiable cnf:**

We have verified the validity of our solution by running on two 20 variables, 100 clauses 3-MAXSAT problem, 3rd benchmark of 100 variables, 505 clauses 3-MAXSAT problem and an 100 clauses 434 clauses Problem.

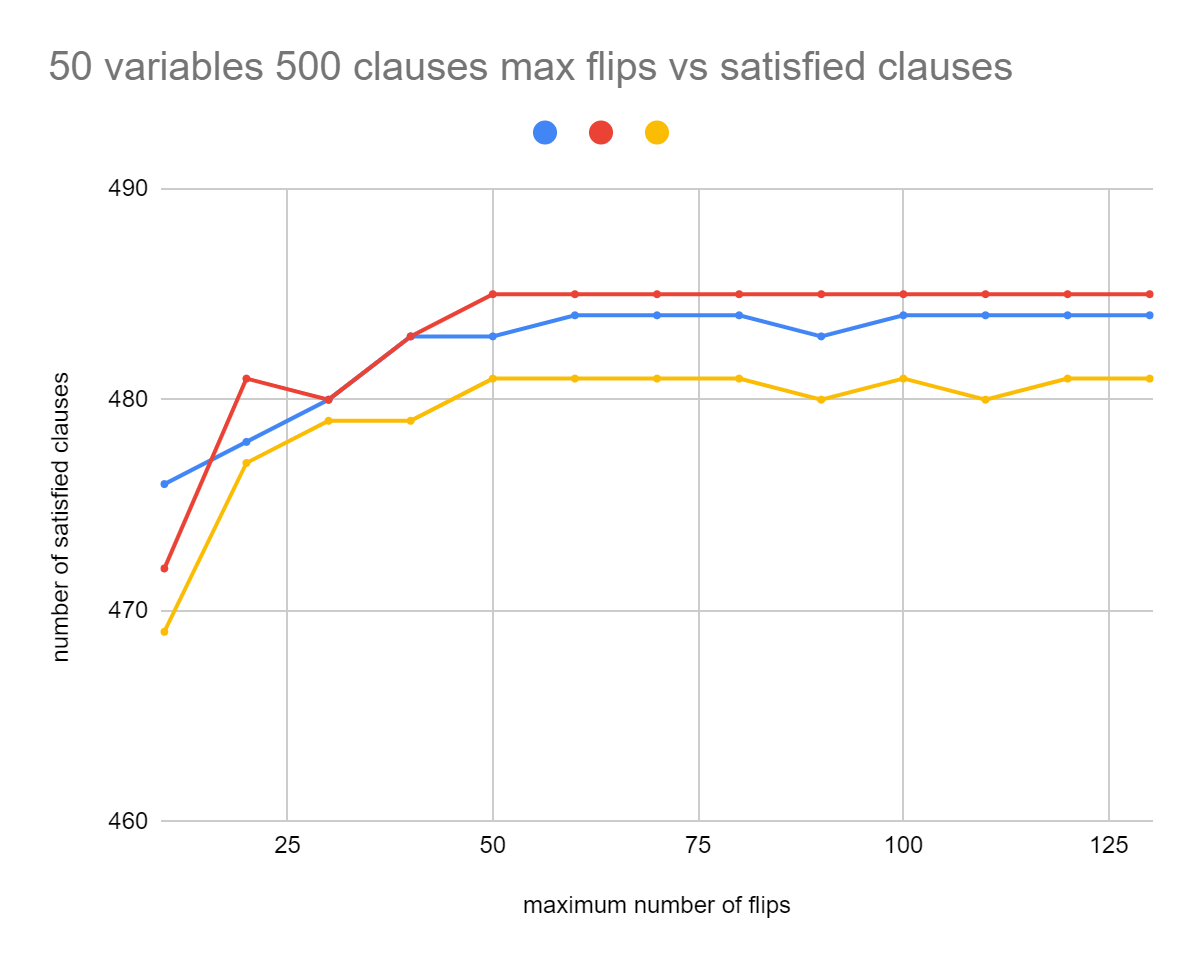
As shown below, Our solution is able to reach the answer pretty fast with less that half the number of clauses flips performed (highlighted in yellow the value of flip\_i)



1. **Unsatisfiable CNF maximum satisfiable value convergence:**

We have measured the number of flips required to reach the maximum value against different values of the max\_flips and charted the following graph. We cross verified the max number of clauses satisfiable by using [pysat RC2Stratified solver](https://pysathq.github.io/docs/html/api/examples/rc2.html).

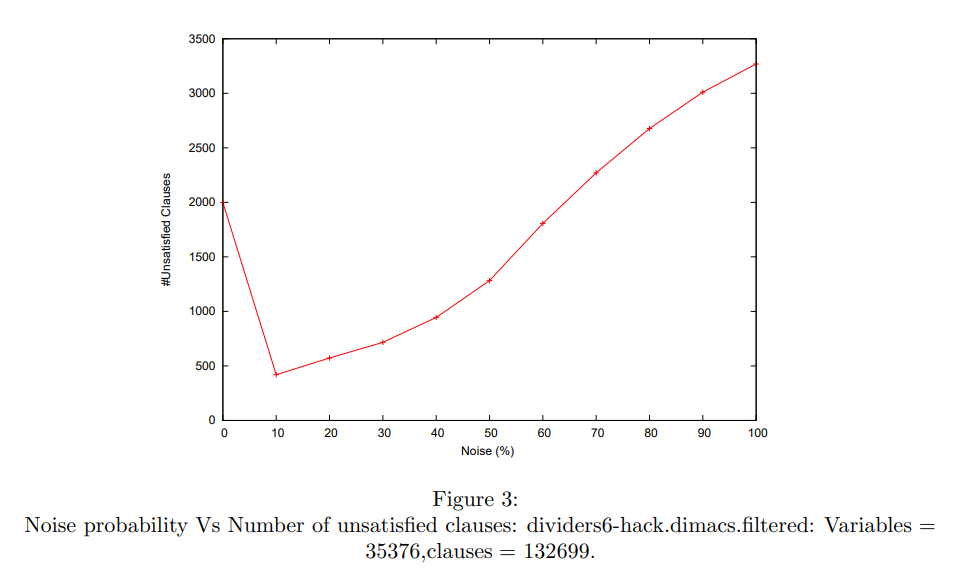
**Observations:**



Here we observed that the maximum value of satisfied clauses is achieved around the 50 max flips, which is no\_clauses/10.

Similar pattern is observed in other tests.

**Conclusion:** from the above two observations we’ve concluded that we need to set the max\_flips to at least half the number of clauses in the CNF.

We then focused on **Pnoise**, the random move probability.

After researching we’ve found a [Research Paper](https://1library.net/document/q05xp6gy-enhanced-walksat-finite-learning-automata-max-sat.html) which stated that “*Peak performance with respect to the lowest number of unsatisfied clauses is achieved when the walking probability was set to* ***10****.*” in the sections 6.1 on page 27.

We’ve also conducted our experiment on a much smaller data set and a similar trend is observed.

**Conclusion:** we’ve found that the best value of noise is 10% or 0.1

After that we focused on the **initial random assignment.**

Here we had 3 choices while setting the boolean value for a variable

Equilikely assignment of True and False for each variable Or Higher probability of True Or Higher Probability of False.

Logically thinking the number of flips required will be higher if the solutions has the higher concentration of True and initial assignment sets False with higher probability, and vice versa. So logically the equi likely assignment of True and False is Optimum.

The Same inference is also verified experimentally

**Conclusion:** we’ve found that equilikey assignment of true and false value for each variable in initial assignment is the best optimal way.

**Improvements:**

During our experiments we’ve found following improvements that boosted the performance of our solution.

1. We found that checking the first initial unaltered(unflipped)assignment for Optimality gave better results than not doing so, as not doing so will skip on the truth assignment always.
2. As the performance of the solutions depends largely on the initial assignment, hence we decided to hash each initial assignment and during re-run we would pick the already occurring assignment with lower probability.

This optimization worked best when the number of variables were lower as the probability of initializing with the same initial assignment is inversely proportional to the number of variables.

Branch-and-Bound Depth-First-Search (BnB) - Topic: TSP

**Problem Definition:**

Given a list of cities and the distances between any two cities, find the shortest route that visits each city exactly once and returns to the origin city.

**State Space:**

State space (partial assignment of values to variables) is basically all the possible routes that have visited some cities (not visiting every city and returning to the original city) just once.

(For example, assume there are 5 cities, state space can be a route that has visited city A, city B, city C, which has a current path cost of 75, or a route that has visited city B, city D, which has a current path cost of 10, and so on.)

**Approach Description:**

Before describing the procedure of Branch-and-Bound Depth-First-Search algorithm, we first need to define the definition of the upper bound, current cost function, heuristic function, and evaluation function.

* **Upper Bound U:** a estimate of the maximum value for the optimal solution
* **Current Cost Function g(n):** the current costs from starting state to the current state
* **Heuristic Function h(n):** the estimated cost from the current state to the goal state
* **Evaluation Function f(n):** the estimated optimal path cost for the possible solution of the current state

**Approach Procedure:**

For Branch-and-Bound Depth-First-Search algorithm, we first randomly select one city as the starting node and expand the state space tree as deep as possible.

When expanding one node, we consider *the current path cost* and a *predefined heuristic function* together as an *evaluation function*, which is used to compare with a *predefined upper bound* so that we could determine whether we should visit deeper states or not. If we decide not to expand deeper nodes, then the state space tree is *pruned*. Otherwise, we will keep on expanding this possible route.

During the BnB-DFS, when one possible route is found, we compare the cost of this route with the predefined upper bound, if the cost of the current solution is lower than the upper bound, we update the upper bound with the cost of the current solution; otherwise, we discard the current solution.

Once the BnB-DFS algorithm is done, we are guaranteed to get an optimal solution, which has the minimal path cost.

**Algorithm Explanation:**

In our experiment, we considered 5 approaches for the TSP problem, which are the *brute-force DFS*, *naive BnB DFS*, *BnB DFS with Greedy*, *BnB DFS with an optimized lower bound*, and *BnB DFS with an optimized lower bound and Greedy method*.

**Brute Force DFS Approach (without Branch-and-Bound no heuristic):**

* explanation: the normal Depth-First search algorithm, which explores the whole state space search tree without pruning.

**Naive Branch-and-Bound DFS:**

* upper bound: the current optimal path cost found so far
* heuristic function h(n): the current path cost from the start city to the current city
* current cost function g(n): the path cost from the start city to the current city
* evaluation function f(n) = g(n) + h(n): the estimated optimal path cost from the current city to the goal city
* explanation:

**Branch-and-Bound DFS with Greedy**

**Branch-and-Bound DFS with An Optimized Lower Bound:**

* heuristic function h(n): 1/2\* Sum of cost of two edges adjacent to u
* current cost function g(n): the path cost from the start city to the current city
* evaluation function f(n) = g(n) + h(n): the estimated optimal path cost from the current city to the goal city
* explanation

**Branch-and-Bound DFS with An Optimized Lower Bound and Greedy**

* heuristic function h(n): 1/2\* Sum of cost of two edges adjacent to u
* current cost function g(n): the path cost from the start city to the current city
* evaluation function f(n) = g(n) + h(n): the estimated optimal path cost from the current city to the goal city
* explanation

1. (optional) Branch-and-Bound Depth-First Search with Minimum Spanning Tree(MST)

* heuristic function h(n): the MST of the unvisited cities and the visited cities set
* current cost function g(n): the path cost from the start city to the current city
* evaluation function f(n) = g(n) + h(n): the estimated optimal path cost from the current city to the goal city
* explanation:

**Properties Analysis:**

1. Brute Force Approach (without Branch-and-Bound no heuristic):

time complexity:

space complexity:

1. Naive Branch-and-Bound Depth-First Search:

time complexity:

space complexity:

1. Branch-and-Bound Depth-First Search with Greedy

time complexity:

space complexity:

1. Branch-and-Bound Depth-First Search with An Optimized Lower Bound:

time complexity:

space complexity:

1. Branch-and-Bound Depth-First Search with An Optimized Lower Bound and Greedy

time complexity:

space complexity:

**Experiment Observations:**

Experiment Parameters

* N: # of cities
* K: # of distinct distance values
* U: mean of normal distribution
* V: variance of normal distribution

Experiment #1 (with **different N: # of cities**)

N=5, K=0.4\*5\*5=10, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 0 | 65 | 482.889 |
| Naive BnB DFS | 0 | 65 | 482.889 |
| BnB DFS with Greedy | 0 | 65 | 482.889 |
| optimized BnB DFS | 0 | 19 | 482.889 |
| optimized BnB DFS with Greedy | 0 | 12 | 482.889 |

N=8, K=0.4\*8\*8=25.6**≈**25, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 48 | 13700 | 777.943 |
| Naive BnB DFS | 46 | 13700 | 777.943 |
| BnB DFS with Greedy | 56 | 13700 | 777.943 |
| optimized BnB DFS | 0 | 300 | 777.943 |
| optimized BnB DFS with Greedy | 0 | 327 | 777.943 |

N=10, K=0.4\*10\*10=40, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4170 | 986410 | 954.516 |
| Naive BnB DFS | 4576 | 986410 | 954.516 |
| BnB DFS with Greedy | 4827 | 986410 | 954.516 |
| optimized BnB DFS | 37 | 8704 | 954.516 |
| optimized BnB DFS with Greedy | 17 | 3412 | 954.516 |

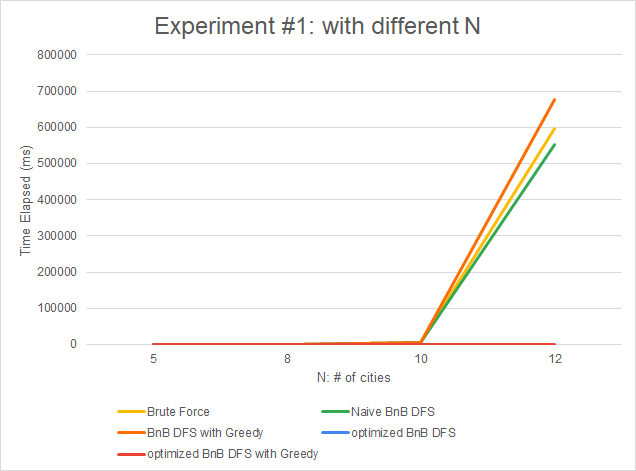
N=12, K=0.4\*12\*12=57.6≈57, U=100, V=5

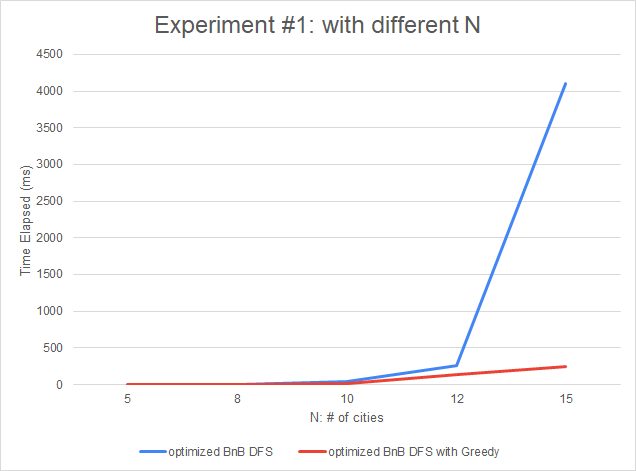
|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 597376 | 108505112 | 1129.63 |
| Naive BnB DFS | 552032 | 107292894 | 1129.63 |
| BnB DFS with Greedy | 677013 | 107221835 | 1129.63 |
| optimized BnB DFS | 253 | 51891 | 1129.63 |
| optimized BnB DFS with Greedy | 134 | 20396 | 1129.63 |

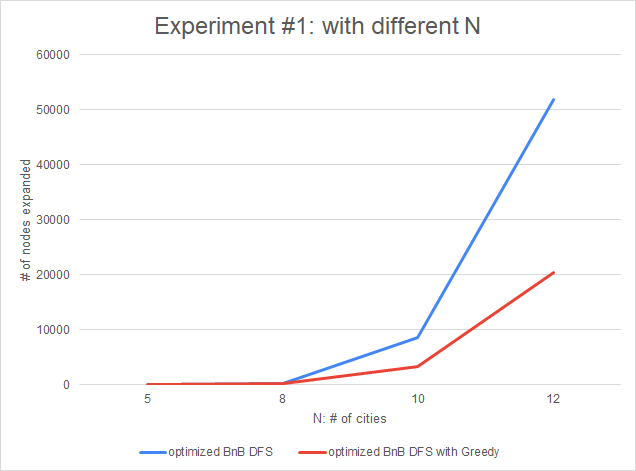
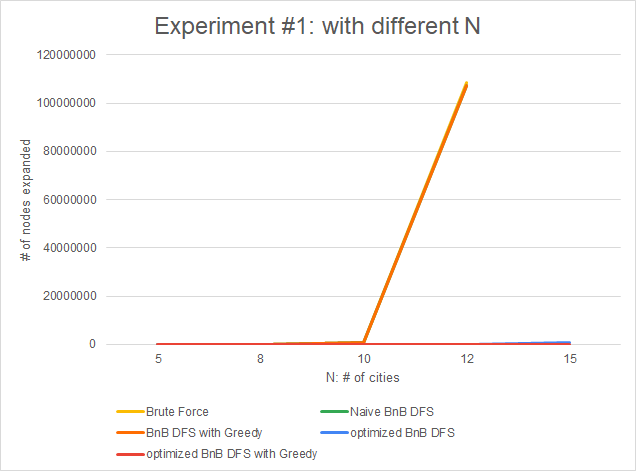
N=15, K=0.4\*15\*15=90, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | -- | -- | -- |
| Naive BnB DFS | -- | -- | -- |
| BnB DFS with Greedy | -- | -- | -- |
| optimized BnB DFS | 4110 | 859203 | 1408.85 |
| optimized BnB DFS with Greedy | 251 | 36464 | 1408.85 |

(A figure for experiment #1 to show 5 different results)







Experiment #2: (with different **K: # of distinct distance values**)

N=10, K=0.01\*10\*10=1, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4057 | 986410 | 908.398 |
| Naive BnB DFS | 4193 | 986410 | 908.398 |
| BnB DFS with Greedy | 5174 | 986410 | 908.398 |
| optimized BnB DFS | 4517 | 986410 | 908.398 |
| optimized BnB DFS with Greedy | 4653 | 986410 | 908.398 |

N=10, K=0.05\*10\*10=5, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4769 | 986410 | 1005.3 |
| Naive BnB DFS | 4404 | 986144 | 1005.3 |
| BnB DFS with Greedy | 5883 | 986096 | 1005.3 |
| optimized BnB DFS | 12 | 2500 | 1005.3 |
| optimized BnB DFS with Greedy | 7 | 935 | 1005.3 |

N=10, K=0.1\*10\*10=10, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4598 | 986410 | 977.367 |
| Naive BnB DFS | 4526 | 986410 | 977.367 |
| BnB DFS with Greedy | 4909 | 986410 | 977.367 |
| optimized BnB DFS | 20 | 4668 | 977.367 |
| optimized BnB DFS with Greedy | 24 | 4145 | 977.367 |

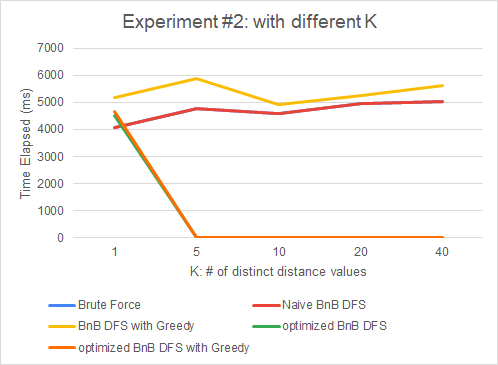
N=10, K=0.2\*10\*10=20, U=100, V=5

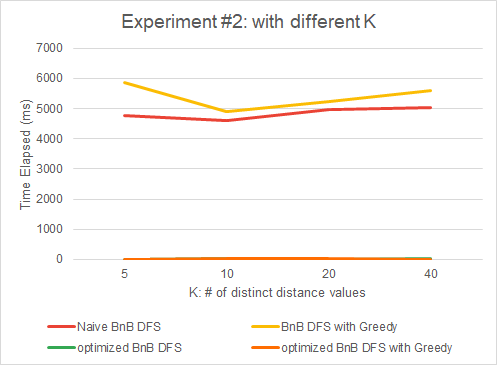
|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4962 | 986410 | 960.012 |
| Naive BnB DFS | 4256 | 986410 | 960.012 |
| BnB DFS with Greedy | 5238 | 986410 | 960.012 |
| optimized BnB DFS | 9 | 6757 | 960.012 |
| optimized BnB DFS with Greedy | 25 | 4212 | 960.012 |

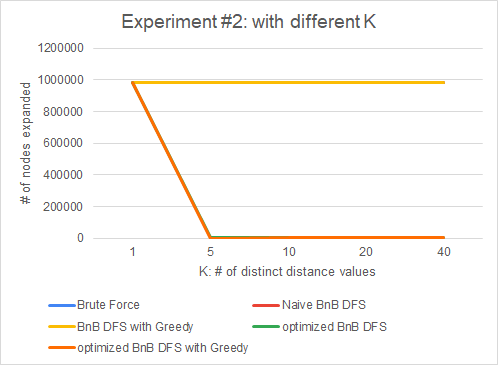
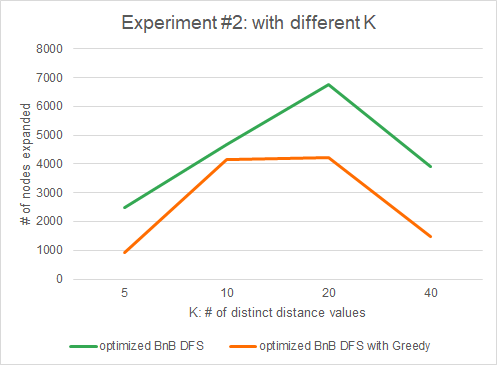
N=10, K=0.4\*10\*10=40, U=100, V=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 5026 | 986410 | 939.733 |
| Naive BnB DFS | 4661 | 985246 | 939.733 |
| BnB DFS with Greedy | 5615 | 985241 | 939.733 |
| optimized BnB DFS | 16 | 3913 | 939.733 |
| optimized BnB DFS with Greedy | 8 | 1463 | 939.733 |

(A figure to show 5 different results)







Experiment #3: (with different **V: variance of normal distribution for costs**)

benchmark:

5%\*U, (small)

25%\*U (big)

N=10, K=40, U=100, V=5%\*100=5

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 3844 | 986410 | 956.743 |
| Naive BnB DFS | 3853 | 986392 | 956.743 |
| BnB DFS with Greedy | 5208 | 986392 | 956.743 |
| optimized BnB DFS | 15 | 2351 | 956.743 |
| optimized BnB DFS with Greedy | 10 | 685 | 956.743 |

N=10, K=40, U=100, V=10%\*100=10

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4092 | 986410 | 922.05 |
| Naive BnB DFS | 3802 | 932187 | 922.05 |
| BnB DFS with Greedy | 5143 | 924947 | 922.05 |
| optimized BnB DFS | 10 | 2416 | 922.05 |
| optimized BnB DFS with Greedy | 2 | 434 | 922.05 |

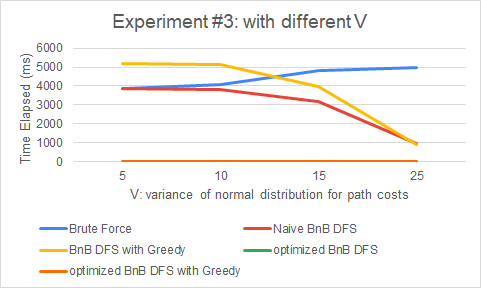
N=10, K=40, U=100, V=15%\*100=15

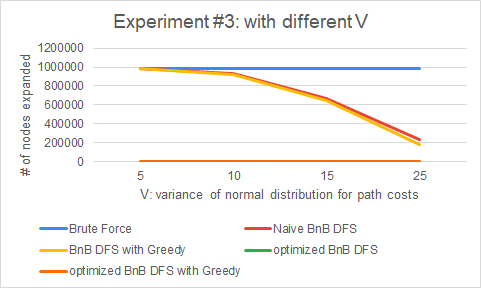
|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4802 | 986410 | 831.049 |
| Naive BnB DFS | 3199 | 663332 | 831.049 |
| BnB DFS with Greedy | 3995 | 640948 | 831.049 |
| optimized BnB DFS | 9 | 1735 | 831.049 |
| optimized BnB DFS with Greedy | 5 | 718 | 831.049 |

N=10, K=40, U=100, V=25%\*100=25

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | Time Elapsed (in millisec) | # of nodes expanded | Minimum cost |
| Brute Force | 4997 | 986410 | 677.714 |
| Naive BnB DFS | 930 | 229457 | 677.714 |
| BnB DFS with Greedy | 898 | 177815 | 677.714 |
| optimized BnB DFS | 1 | 766 | 677.714 |
| optimized BnB DFS with Greedy | 0 | 117 | 677.714 |

(A figure to show 4 different results)





Appendix

References

---DFS B&B on TSP---

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

<https://www.coursera.org/lecture/advanced-algorithms-and-complexity/tsp-branch-and-bound-RkoEK>

<https://en.wikipedia.org/wiki/Branch_and_bound#Pseudocode>

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

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

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

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

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

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

--- SLS for MAX-SAT ---

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

[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)

[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/)

[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/>

Project Report Guidelines

deadline: Dec. 17, THU, 11:59 p.m.

Guidelines:

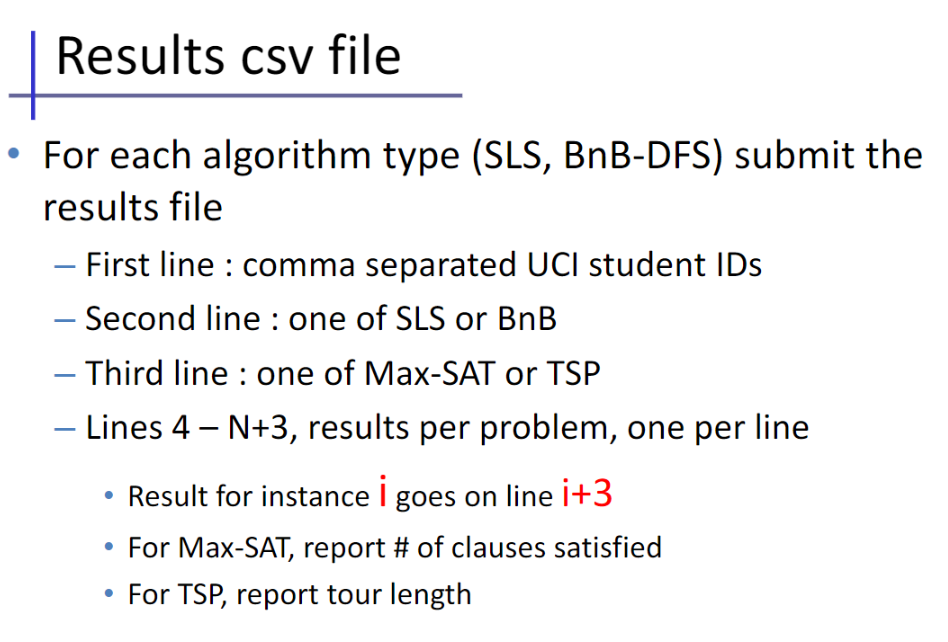
* 6-15 pages, not including: Title page, References, and Appendix
* Break the report into 2 parts
* Do NOT print supporting materials, e.g. source codes
* Total 3 files to submit:

1. project report - pdf
2. source codes with README build instructions - zip
3. competition benchmarking results files (for SLS and BnB-DFS) - zip

Submission:

* create a group on Canvas and add team members

<https://gist.github.com/jessiwu/c1a696508794dbfac4485a7e4c62a495>

****