universidade de aveiro

Minimum Node Weight Dominant Set of an Undirected Graph Brute Force and Greedy Heuristic Approach

Advanced Algorithms

Bruno Silva 98374

***Abstract* –This report provides insightful results into the advantages and disadvantages of both *Brute Force* algorithms and *Greedy Heuristics*. These approaches were compared by computing a graph problem : *Computing the Minimum Node Weight Dominant Set of an Undirected Graph*. It was shown clearly how unfeasible the *Brute Force* approach is for bigger data inputs, and how much quicker the heuristics is, even though at the cost of computing only a local optimum.**

***Keywords* –Graphs, Dominant Set, Optimization, Brute Force, Greedy Heuristic**

I. INTRODUCTION

Graph theory is a branch of computer science that studies the relationships between nodes and edges in networks. It's widely used to model real life systems involving networks, such as social media, transport systems, etc...
One of the many properties of a graph is its dominant sets. A dominant set can be described as a subset of nodes of graph such that every other node not included in this subset is in the neighborhood of at least one node from the dominant subset. If a weight is assigned to each node of the graph, this problem can turn into an optimization problem, by aiming to find either the heaviest or lightest dominant set.
Various approaches are valid for solving this sort of problems, the more straight forward approach is often *Brute Force*. A brute force algorithm blindly evaluates every possible solution, without ever stopping to reconsider if the optimal solution was already found. This algorithm design's biggest downside is obviously that the time and space complexity is often undesirably high, very frequently hoping into the combinatorial explosion realm, leading to unfeasible computing times very fast with bigger input sizes. On the other hand, it always guarantees that if an optimal solution exists, it will be found.
Another very usual choice for solving this problem is with *Greedy Heuristics*. A Heuristic can be defined as intuitive approach/set of guidelines that helps the computer make educated guesses much like humans do, although it may not guarantee an optimal solution.
A greedy heuristic is particular has a few rules:

- Begins from a partial constructed solution and expands into a complete solution[1]
- The best possible choice is taken each step.
- Each choice has to be feasible
- Each choice is irrevocable

This design by not being able to make a sacrifice and choose a worse local choice that would yield better choices in the future, shows its lack of intelligence, and this is why it often does not lead in fact to the global optimal solution.

II. FORMAL COMPUTATIONAL ANALYSIS

This section aims to provide a formal analysis of the problem, which will later on be backed up by the experimental results.

A. General Considerations

For this problem, it's meaningless to analyze disconnect graphs as the solution would always be trivial: *Find the subset for each component of the graph and add the subsets*. For this reason, all graphs were generated with only one component.
Also, for similar reasons, isolated nodes are trivial to analyze as well, as every isolated node would be necessarily be part of any dominant set, otherwise isolated nodes would be nodes without any dominant set's nodes in its adjacency, thus breaking every possible solution.

B. Brute Force Algorithm

The procedure computing the solutions using *Brute force* is as follows:

1. Compute every possible graph subset
2. Iterate over all the subsets and test if they are dominant:
   (a) Iterate over every outsider node[2]
   (b) Get each outsider node's neighborhood
   (c) Check if the neighborhood intersects the dominant set
   (d) If the intersection is always different from 0,it's a dominant set.
3. Find minimum weights

The time complexity of each step is:[3]

1. For a graph with n nodes, computing every possible subset has exponential time complexity $O(2^n)$
2. Iterating over every subset is also $O(2^n)$

[1] A complete solution does not mean the global optimum solution

[3] Each step matches the list above

(a) Iterate over every outsider node: $O(n_{outsider})$, where $n_{outsider}$ is the number of nodes.
(b) Get outsider node neighborhood : $O(d)$, where d is the average degree of the outsider nodes
(c) Check intersection : $O(s)$, where s is the size of the dominant set

3. Finding minimum subset weights includes summing all the weights of the subset's nodes and the finding the minimum. The former is hard to analyze as the number of dominant sets is graph dependent, but in the worst case where every subset is also dominant would be $O(2^n)$ and in the best would be $\Omega(1)$ if there would be only 1 dominant set. The latter is easier to analyze, its just $O(L)$, where L would be the number of dominant subsets

Overall, in the worst case scenario the time complexity would be:

$$O(2^n + 2^n + n_{outsider} + d + s + 2^n + L) \quad (1)$$

However, it can be simplified since in practice all the cases where all the subsets are dominant are usually not meaningful, such as 100% dense graphs, because if every subset is dominant, or almost every one is, like in very dense graphs, it's not an interesting problem anymore.

$$O(2^n + 2^n + n_{outsider} + d + s + L_{average} + L) \quad (2)$$

However only the worst term matters, so the overall time complexity of the Brute Force algorithm is:

$$O(2^n) \quad (3)$$

C. Greedy Heuristic

This approach is much simpler, it comes from human intuition, is somehow similar to how a human with limited "computing power" would solve it:

1. Order nodes by $\frac{weight}{degree}$
2. Chose the node with smaller ratio and remove it from the list of non chosen nodes
3. Check if it's a dominant set
4. If not, repeat steps from 2

The intuition for this heuristic is :

- If a node has less weight, it's obviously better
- If a node has more neighbors, it has more potential to be a crucial node in a dominant set
- Consequently, the lowest $\frac{weight}{degree}$ the better the node will be

The time complexity is much simpler too, in the worst case scenario it's necessary to order the nodes, which since *Quicksort* [4] is being used, on average is $\Theta(n * log(n))$. If it's assumed $L_{average}$ to be the average number of nodes in a dominant set we would have to check on average $L_{average}$ times if the subset was already dominant. And considering the time complexity of checking if a subset is dominant and finding the minimum weight was already seen in the *Brute Force* approach as $O(n_{outsider} + d + s + L_{average} + L)$, we can deduct that the overall time complexity of the *Greedy Heuristic* is :

$$O(n * log(n))^5 \quad (4)$$

However, while this approach gains in time and space efficiency, lacks in accuracy. Take the example of this simple graph :

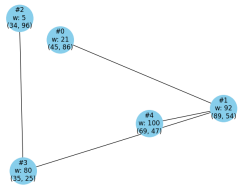

Fig. 1
GREEDY HEURISTIC FAILING TO OUTPUT GLOBAL BEST

The global lightest dominant set of this graph is $[1, 2]$ with $w = 97.0$

Node	Weight	Neighbors	Weight-Neighbor Ratio
0	21	1	21.0
1	92	3	30.66
2	5	1	5.0
3	80	2	40.0
4	100	1	100.0

TABLE I
HEURISTIC'S ANALYSIS

From this table, the order of nodes that the greedy heuristic will chose will be $[2, 0, 1, 3, 4]$. However, neither $[2]$ nor $[2, 0]$, which will be the 2 first partial constructed solutions from the heuristic, are dominant. Thus, from continuing the execution the heuristic gets $[2, 0, 1]$, which is in fact dominant, and is better than a lot of other dominant sets. Nevertheless, it's not the lightest dominant set as $w = 118$.

III. EXPERIMENTAL RESULTS

For replicability of the experiments to remain true, the seed *98374* was used for every random operation.

A. Number of Operations

B. Largest problem feasible

C. Comparison of formal and experimental results

[3] Outsider nodes is the complementary subset of the dominant set
[4] np.argsort() is being used, which uses *Quicksort by default*
[5] Reminder that n is the number of graph nodes