1 Introduction

In many combinatorial optimization problems, greedy strategies do not generally generate an optimal solution. However, a greedy heuristic may yield to a near optimal solution in reasonable time. For example, a greedy strategy for the traveling salesman problem is the following heuristic: "at each stage visit an unvisited city which is nearest to the current city, this heuristic does not return a good solution, but terminates in a reasonable number of steps, finding an optimal solution typically requires unreasonably many steps.

In this chapter, we present the main concepts of a greedy algorithm. An example that illustrates how a greedy heuristic is applied to an instance of the MWDS problem is also given. Then, we review, from the literature, the well-known greedy heuristic for the MWDS problem. Finally, we describe how we can improve the performance of two of the previous greedy heuristics.

2 Greedy algorithms

Greedy algorithms are one of the standard algorithm design techniques such as dynamic programming and divide and conquer paradigm. They are often used to solve optimization problems. In the greedy approach, there is no division of the problem instance into smaller instances.

2.1 Definition

A greedy algorithm arrives at a solution by making a sequence of choices, each of which simply looks the best at the moment. It grabs data items in sequence, each time taking the one that is deemed "best" according to some criterion, without regard for the choices it has made before or will in the future. That is, each choice is locally optimal. The hope is that a globally optimal solution will be obtained, but this is not always the case. For a given algorithm, we must determine whether the solution is always optimal. A greedy algorithm does not always guarantee an optimal solution [24].

2.2 Greedy algorithm components

A greedy algorithm starts with an empty set and adds items to the set in sequence until the set represents a solution to an instance of a problem. Each iteration consists of the following components: [24]

- A ***selection procedure*** chooses the next item to add to the set. The selection is performed according to a greedy criterion that satisfies some locally optimal consideration at the time.
- A ***feasibility check*** determines if the new set is feasible by checking whether it is possible to complete this set in such a way as to give a solution to the instance.
- A ***solution check*** determines whether the new set constitutes a solution to the instance.

2.3 Greedy algorithm properties

The greedy algorithm paradigm is one of the most important in algorithm design,because of its simplicity and efficiency. Greedy algorithms are used in at least three ways [27]:

a) They provide exact algorithms for a variety of problems;
b) They are frequently the best approximation algorithms for hard optimization problems;
c) Due to their simplicity, they are frequently used as heuristics for hard optimization problems even when their approximation ratios are unknown or known to be poor in the worst-case.

2.4 Case of failure

For many other problems greedy algorithms fail to produce the optimal solution, and may even produce the worst possible solution for example:

with the goal of finding the biggest-sum, in the example in the next page, at each step the greedy algorithm will chose 20 in the second step instead of 15 which appears to it the optimal choice , and it will not reach the best solution that contains 97.

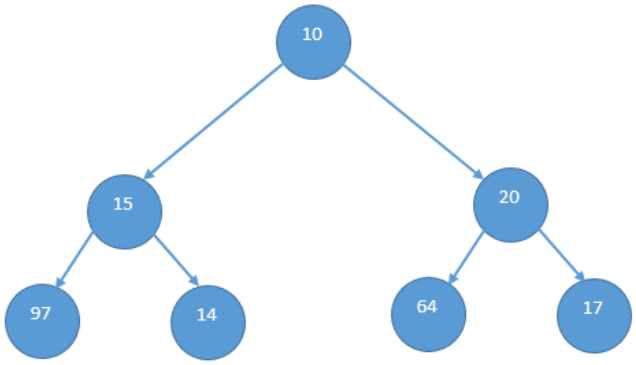

Figure 3.1- case of failure in greedy algorithm

3.5 Pseudo-code for a greedy algorithm

Figure 3.1 shows a pseudo-code for a general greedy approach to optimization.

Pseudo-code for a greedy algorithm

```
1: input: C is the set of candidate items
2: S ← ∅ // We construct the solution in the set S
3: while C ≠ ∅ and not solution(S) do
4:     x ← select(C)
5:     C ← C \ {x}
6:     if feasible(S ∪ {x}) then
7:         S ← S ∪ {x}
8: if solution(S) then
9:     return S
10: else
11:     return there are no solutions
```

Figure 3.2 Pseudo-code for a standard greedy algorithm [9].

3 Greedy heuristic for MWDS

Given an undirected graph *G(V;E)*, where *V* is the set of vertices and *E* is the set of edges, and a function $w : V \rightarrow \mathbb{R}+$ that associates a positive weight value *w(v)* to each vertex $v \in V$. Existing greedy heuristics for the MWDS problem build a solution *D* step-by-step, adding one

vertex to *D* at each construction step. Given a partial solution *D*—that is, a set $D \subseteq V$ which is not yet a dominating set—we can differentiate partition V into three disjoint subsets [26]:

1) The vertices in S are called ***black*** vertices.

2) Vertices that are not contained in S but that are adjacent to at least one vertex from S are called ***gray*** vertices.

3) The remaining vertices are referred to as ***white*** vertices (or uncovered vertices).

A vertex ***v*** can be chosen to be included in S according to one of the following greedy functions [29]:

1) In the first heuristic, the dominating set being constructed, ***D***, is initially empty. The ***white degree*** of a node, ***d(u)***, is the number of ***white*** neighbors of a given node ***u***. For computing the minimum weight dominating set, we divide the ***white degree*** of the node, ***d(u),*** by the weight of the node ***w(u)*** and add the node with the maximum ratio to the dominating set ***D***. The next node to include in ***D*** is selected as follows:

$$v \leftarrow argmax \left\{ \frac{d(u)}{w(u)} \middle| u \in (V - D) \right\}.$$

2) In the second heuristic, the sum of the weights of the white neighbors of a node is divided by the weight of the node. The node with the maximum ratio is taken as the node to be included in the dominating set. The weight of the white neighbors of a node ***u*** is represented as **W*(u).***

$$v \leftarrow argmax \left\{ \frac{W(u)}{w(u)} \middle| u \in (V - D) \right\}.$$

3) The third heuristic is calculated as the difference of the weight of the white neighbors of a node and its own weight. The node with the maximum difference is chosen as the node to be included in the dominating set.

$$v \leftarrow argmax \{ W(u) - w(u) \mid u \in (V - D) \}.$$

4) In the fourth heuristic, the product of the white degree of a node and the sum of the weights of the white neighbors of a node is divided by the weight of the node. The node with the maximum ratio is taken as the node to be included in the dominating set.

$$v \leftarrow argmax \left\{ \frac{W(u) \times d(u)}{w(u)} \middle| u \in (V - D) \right\}.$$

The results of the experiment done by Potluri & Singh [potSin2013] showed that the first, the second and the last heuristic function result in a similar performance with no

clear pattern of a particular greedy heuristic being better for general graph instances. In contrast, the third greedy heuristic clearly resulted in the poorest performance.

4 Illustrative example of a greedy heuristic for the MWDS problem

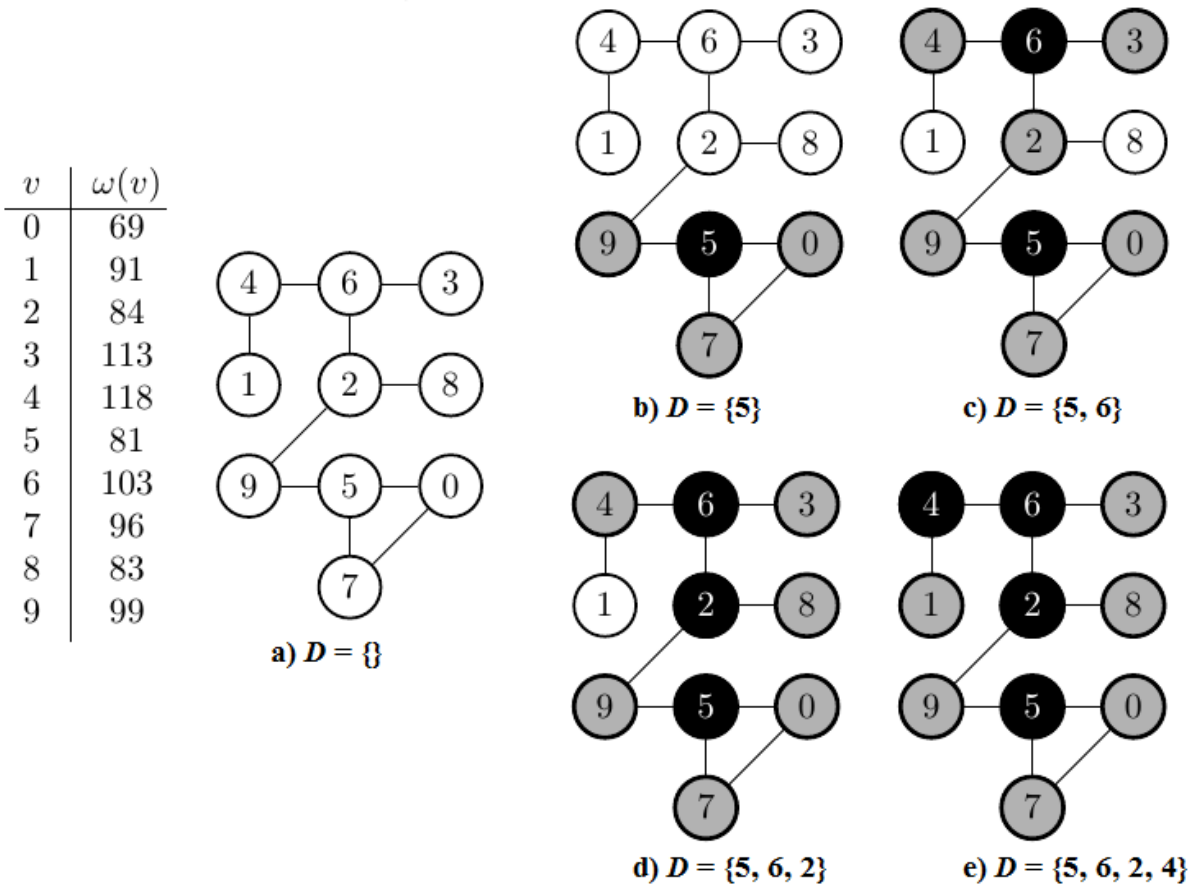

Figure 3.3 An illustrative example of the MWDS problem [10]

Figure 3.2 represents an instance of MWDS problem with 10 vertices (labeled as 0; 1; … ; 9) on which illustrates the basic steps of the first greedy heuristic. At the beginning the partial solution is empty, D = ϕ , and all vertices in V are white. On the other hand, when a complete solution is reached no white node can be found.

4.1 Our proposed greedy heuristics for the MWDS problem

In the previous example, we notice that this heuristic excludes all white vertices with zero current degree to be included in the solution. Consider the two adjacent vertices 1 and 4 as depicted in Figure I.12(d). Here, we have ***w(1) = 91***; ***w(4) = 118***; ***d(1) = 0*** and ***d(4) = 1*** since vertex 4 has just one white neighbor, that is, vertex 1. If this is the case, vertex 4 will be chosen rather than vertex 1 and this, in fact, leads certainly to a solution no much better that if

vertex 1 is considered. To overcome this situation, we propose two modified greedy heuristics:

a) We modify the first greedy heuristic as follows. For each white vertex ***u***, we increment its current degree ***d(u)*** by one, that is ***d^p(u)=d(u)+1*** if u is a white vertex else ***d^p(u)=d(u)*** .

$$v \leftarrow argmax \left\{ \frac{d^p(u)}{w(u)} \middle| u \in (V - D) \right\}$$

b) The second greedy heuristic is modified as follows. For each white vertex ***u***, W^p***(u)*** = ***W(u)*** + ***w(u)*** where ***u*** is a white vertex else W^p***(u)*** = ***W(u)***.

$$v \leftarrow argmax \left\{ \frac{W^p(u)}{w(u)} \middle| u \in (V - D) \right\}.$$

5 Example of greedy heuristic for MWDS

The node is chosen to be in the dominating set according to one of the Perevious explained heuristics:

Before we start with the example demonstration let's point out step by step the greedy algorithm performance using C++ code explanation.

As we already mentioned we have, the W(D) contains all the white vertices and the D solution partial is empty:

- Count the heuristic and select a chosen vertex

```
cd = degree[*it];
nval = (double) cd / (double) weight[*it];
…

best_vertex = *it;
```

- The chosen vertex is colored in BLACK and putted in the solution sub-set.

```
color.at(best_vertex) = 'b';
blist.push_back(best_vertex);
```

- The neighbors of the chosen vertex are colored GRAY, and deleted from the W(D).

```
color.at(*it) = 'g';
wlist.remove(*it);
```

- If a GRAY vertex has no WHITE neighbors, do not take it in consideration in the next iteration.
- Keep repeating the previous steps until the W(D) is empty.

```
while(! wlist.empty())
```

This example will illustrate the first greedy algorithms performance $v \leftarrow argmax \left\{ \frac{d^p(u)}{w(u)} \middle| u \in (V - D) \right\}$ on an undirected weighted graph, with 15 vertices that are represented in circles with a number inside which is the ID of the vertex, and the number outside the circle represents the vertex's weight.

Initially all the vertices are colored white W(S) = V.and the sub-set of solution S is empty S =∅

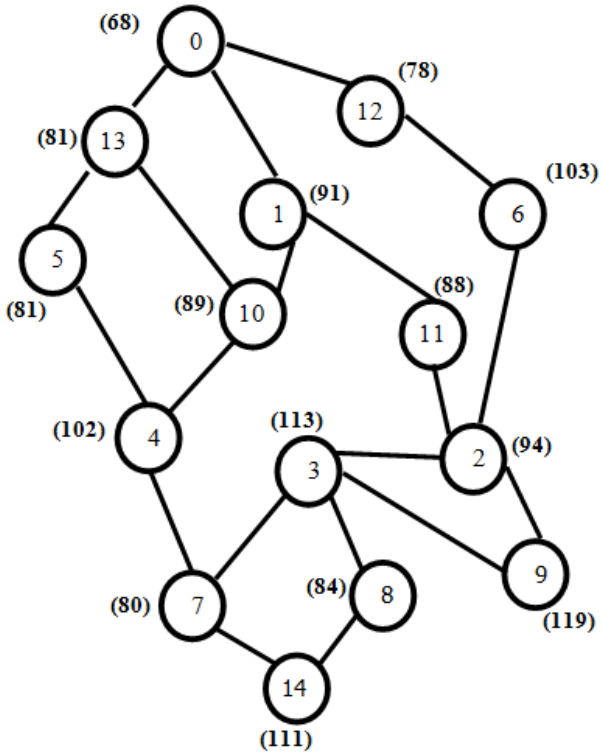

Figure 3.4- input undirected weighted graph with white nodes

This heuristics select vertex 0 during the first iteration for inclusion into the partially constructed dominating set. Vertex 0 is colored BLACK now and after that all the neighbors of vertex 0, vertices 1, 12, 13 and 6 are colored GREY, this step is shown in the figure below.

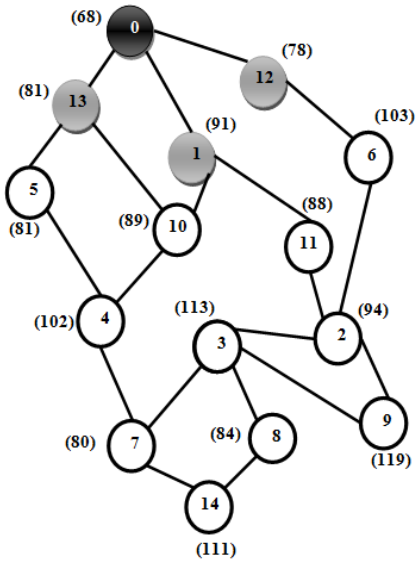

Figure 3.5- Vertex 0 is selected by the heuristic in the first iteration

In the next iteration, the vertex 2 is selected to be added to the solution subset S colored in BLACK and its neighbors in GREY

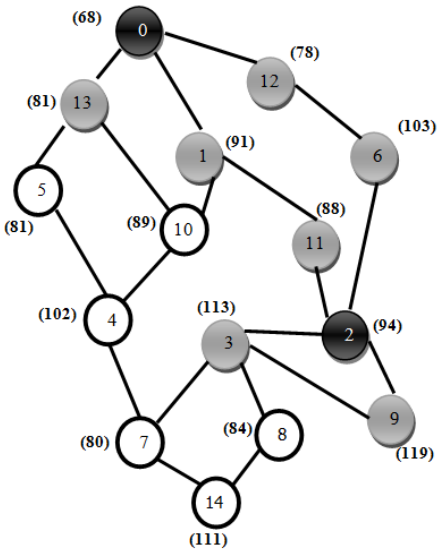

Figure 3.6- Vertex 2 is selected by the heuristic in the second iteration

While there is remaining WHITE vertices the process is still running in this third iteration, the vertex 7 is colored in BLACK and the vertices 4, 14 are colored in GRAY

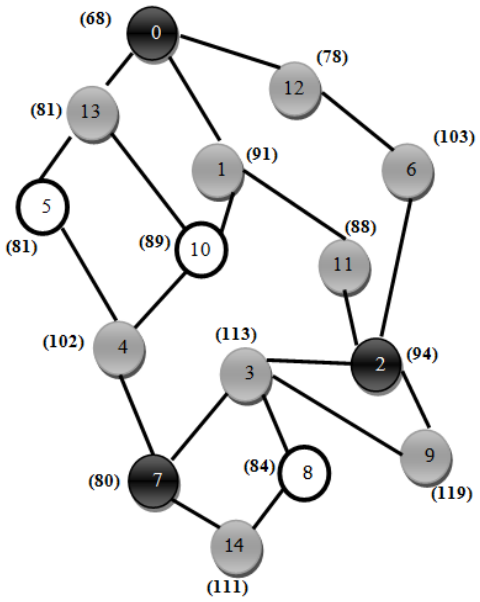

Figure 3.7- Vertex 7 is selected by the heuristic in the third iteration

Finally, and keep working with the same process, in the last iteration the vertex 8 is colored in BLACK and chosen by this heuristic to be added to solution partial

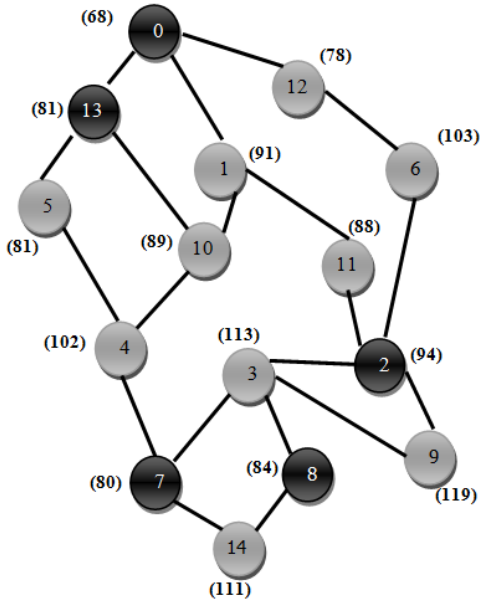

Figure 3.8- Vertex 8 is selected by the heuristic in the fifth iteration

The results that we have from this greedy heuristic algorithm is W(S) =∅ , S = {0, 2, 7, 13, 8} with a minimum weight equals to 407.