Department of Decision Sciences

HONPR2C
Project II

An analysis of classical heuristics for the *p*-median problem

Assignment 3: Final Report

Assignment Unique Number: 705004

Gemma Dawson

50223909

Study Leader: Ms J L le Roux

29 November 2017

Signed Declaration

I hereby declare that this project is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

Signed:

[illegible]

Gemma Dawson
Student Number: 50223909

Contents

1 Introduction

The network facility location problem of finding a location for facilities or public services within a given space by optimising some predefined objective, is not a simple one. Within the single-objective branch of network location problems, there is a plethora of papers exploring minisum, minimax, and covering problems.

If the number of facilities needed is known, and these facilities need to be located in the manner of minimising the total distance between demand nodes and their assigned facility, then the decision-maker is facing a p-median problem.

The p-median problem is well studied and many different approaches to finding both exact and approximate solutions have been proposed, but determining which method would be best for a particular problem is not clear.

This study will analyse and compare several heuristics applied to the p-median problem in the hope that this decision may be made more clear.

2 Description of the Problem

The uncapacitated p-median problem has been widely studied and many different approaches to finding an acceptable, if not optimal, solution have been presented over the past fifty years.

The focus of this study is to statistically analyse and compare the results obtained from various constructive and local search heuristic algorithms applied to the p-median problem.

Different heuristics applied to p-median problems can produce varying quality of results dependant on the size of the original problem. As algorithms take a considerable amount of time and research to be suitably set-up and run, it may be of benefit to have some prior knowledge of which heuristics may be suitable for the problem on hand.

As such, the main objective of this study would be to provide an indication of which method would be best suited to a given problem.

This study will also investigate the computational time each algorithm requires to generate a solution, taking into account the size of the problem being solved.

Since parametrisation of heuristics play a crucial part of the efficiency of the algorithm and the quality of the results it produces, this will be explained in detail in the study. Given time, different approaches to the same heuristic may be explored as well.

3 Literature Study

While the p-median problem may have been tackled by academics before 1964, it was first formally defined and formulated by Hakimi [1]. It was also in this paper that Hakimi proved that the optimal location of a facility in a connected discrete network would always be found to correspond to the location of a demand node. This considerably reduces the solution search space.

A few years later, ReVelle and Swain [2] formulated the p-median problem as an integer programme which provided the answers to both the question of where the optimal medians were located as well as which demand nodes should be allocated to each median. In this paper, the solution properties of solving this problem with linear programming using the branch-and-bound technique was also discussed.

Later that decade, it was shown that the p-median problem is NP-hard and as such there exists no algorithm that can find the optimal solution within polynomial time. Since most real-world problems are very large in nature, this poses a great problem. Well developed heuristics can provide satisfactory solutions and many such approaches have been proposed. Mladenović et al. [3] produced a comprehensive survey of the heuristics that have been used to solve the p-median problem. This survey divides heuristics into two groups, Classical Heuristics and Metaheuristics. The classical heuristics group consists of constructive heuristics, such as the greedy algorithm, local search, and mathematical programming. The metaheuristics group lists tabu search, variable neighbourhood search, and simulated annealing among others.

There do exist studies that analysis the efficiency of heuristics in solving the p-median. Simulated annealing was analysed by Chiyoshi and Galvão [4] while Alp et al. [5] investigated their proposed genetic algorithm. Hansen and Mladenović [6] applied the variable neighbourhood search algorithm to the p-median problem as well as heuristic concentration, tabu search, and the greedy interchange algorithm. These studies used Beasley's test problem dataset although Hansen and Mladenović's review only used a portion of the available test problems.

Rolland et al. [7] explored the efficiency of tabu search using randomly generated datasets as test problems, and this was compared to results obtained from a two-exchange heuristic and an integer programming algorithm. A heuristic concentration was compared to tabu search by Rosing et al. [8]. Since datasets were independently and randomly generated for each of these studies, it makes it very difficult to compare results.

4 Methodology

All algorithms have been coded in R [9], an open source programming language, due to its ease of use, extensive availability of online resources as well as the functionality provided by downloadable packages that provide useful data handling and visualisation.

As the results of the algorithms under investigation are to be compared, identical problems have been presented to each algorithm to solve. Beasley [10] offers a forty-instance dataset which comprises of problems ranging from 100 to 900 demand nodes that require the locations of between five and 200 medians to be determined. These datasets are given in a text format presenting the pairs of nodes along with the associated cost of traversing this node pair. These text files were converted to cost matrices using the R package igraph [11]. These matrices provide the cost of travel from one node to any other node in each network of the forty instances.

Beasley's library also provides the optimal solution's weighted total cost objective function value for each of the forty instances in the dataset. This allows the results obtained from the algorithms to be objectively assessed against the optimal solution.

As all algorithms require nodes associated with a minimum cost to be identified, a function in R was written to deal with the issue when more than one nodes offer the same minimum cost. R's base package offers a minimum function, but if more than one element in the vector that is passed to this function equals the minimum value, only the first of such elements will be returned and all subsequent elements will be ignored. However, the new function will return a randomly selected element. A direct consequence of having this element of randomness is that an algorithm may produce different results and as such, for every problem in the forty-instance dataset, each algorithm will be run 50 times and the resulting objective function value was recorded, which allowed the deviation of each algorithm's solution from the optimal solution to be analysed. In addition, the computational time required for each of the 50 runs was timed. In the case of the local search heuristics, the number of iterations per run was also recorded.

5 Classical Heuristics and Metaheuristics

5.1 Overview

As previously mentioned, a review of approaches to solving the p-median problem was presented by Mladenović et al. [3] in 2007. This paper divided heuristics into one of two groups, Classical Heuristics and Metaheuristics. This classification has been utilised in this study.

An algorithm can be classified as a Classical Heuristic if it is developed for the purpose of solving the p-median problem, while a heuristic that was developed independent of the p-median problem can be termed a Metaheuristic.

5.2 Pseudocode Notation

For each algorithm examined in this study, a pseudocode will be presented using notation given below. This notation is similar to that used by Whitaker [12].

G :	the problem network
n :	the number of nodes within the network
m :	the number of nodes nodes that are available to be assigned as a median
p :	the number of medians required by the solution
M :	the set of nodes that are available to be assigned as a median
P^* :	the set of medians where $P^* \subset M$
P :	the set of free nodes not in the solution median set where $P \subset M$ and $P \cup P^* = M$
S^* :	objective function value for supplying the network G
k :	an iteration parameter
∞ :	an arbitrarily large number
d_{ij} :	the cost of travelling from node i to potential median j
u_i^k :	the cost of travelling from node i to the closest median j where $j \in P^*$ during iteration k
w_i^k :	the cost of travelling from node i to the second closest median j where $j \in P^*$ during iteration k
c_j^k :	the potential value of S^* if node j is added to the median set P^* during iteration k
S_r :	the value of S^* after adding node r to the median set P^*
S_{rt} :	the potential change to S* resulting from the interchange of node r with median t
S^k :	the change in the value of S^* during iteration k
I, J, K_j :	subsets of the n nodes defined as needed in the pseudocode
a, b, z, q :	indices and parameters defined as needed in the pseudocode
$\emptyset$:	an empty set

6 Classical Heuristics: Constructive

A constructive heuristic is a technique that starts with an empty solution and each iteration extends this solution until a complete solution has been found. Within the context of the unconstrained p-median problem, for each iteration of the algorithm, a median is identified by some deterministic method and added to the current solution. This process continues until the median set contains p elements.

6.1 The Greedy Algorithm

The greedy algorithm, as described by Whitaker [12], is a simple recursive algorithm that starts with an empty median set and, with each iteration of the algorithm, adds a node to the median set. The node that is added to the median set is the one that decreases the objective function value the most when compared to the other non-median nodes.

6.1.1 Pseudocode

Algorithm 1 Greedy Algorithm

```
  INPUT: Cost Matrix

  STEP 0: Initialisation

      Set $P^* = \emptyset$; $k = 1$; and $u_i^1 = \infty$ for all $i = 1,2,\ldots,n$

  STEP 1: For each free node, the objective function value resulting from selecting
  this node as a median is determined. This objective function value is calculated
  by summing the cost of travel for each node in the network to either the closest
  median or the free node under investigation, whichever is smallest.

      $c_j^k = \sum_{i=1}^{n} \min(d_{ij},\, u_i^k) \quad \forall j \in M,\, j \notin P^*$

  STEP 2: The free node that will result in the smallest possible objective function
  value is identified, and the objective function value is updated.

      $S_r = \min_{j \notin P*}(c_j^k) \quad \text{for } r \in M,\, r \notin P^*$
```

STEP 3: Add node r to the set of medians and then determine if the algorithm has completed the number of required iterations.

$$P^* = P^* \cup r$$

```
if $k = p$ then
go to STEP 5 [
else
go to STEP 4 ]
end if
```

STEP 4: Increase the value of iteration parameter and update the cost of travelling from any node to its closet median given that node r has been added to the median set.

$$k = k + 1$$
$$u_i^k = \min(d_{ir},\, u_i^{k-1}) \quad \text{for } i = 1{,}2{,}3\ldots{,}n$$

Go to STEP 1

STEP 5: Once p iterations are complete, set the final objective function value equal to the current iteration's objective function value and stop the algorithm.
$S^* = S_r$ and STOP

6.1.2 Complexity

The greedy algorithm has a complexity of $\mathcal{O}(n.p)$ since for each median that is required, the algorithm will assess the network of n nodes until a set of p medians have been found.

6.1.3 Results

The results from running the greedy algorithm for each of the forty test problems can be seen in table 1.

As can be seen in the table, the computational time is very quick for the test problems that require a small number of medians. The problem that took the longest to run was pmed30 which took on average over 5 seconds to find a solution. This problem requires that two-thirds of the 600 nodes in its network be assigned as medians. It can be seen from figure 1 that the ratio network size to the number of required medians affects the computational time of the algorithm.

Computational time will obviously increase as the network size increases as there are simply more nodes that need to be assessed and as the number of medians required increases in addition to this, it can quickly become a time-consuming algorithm.

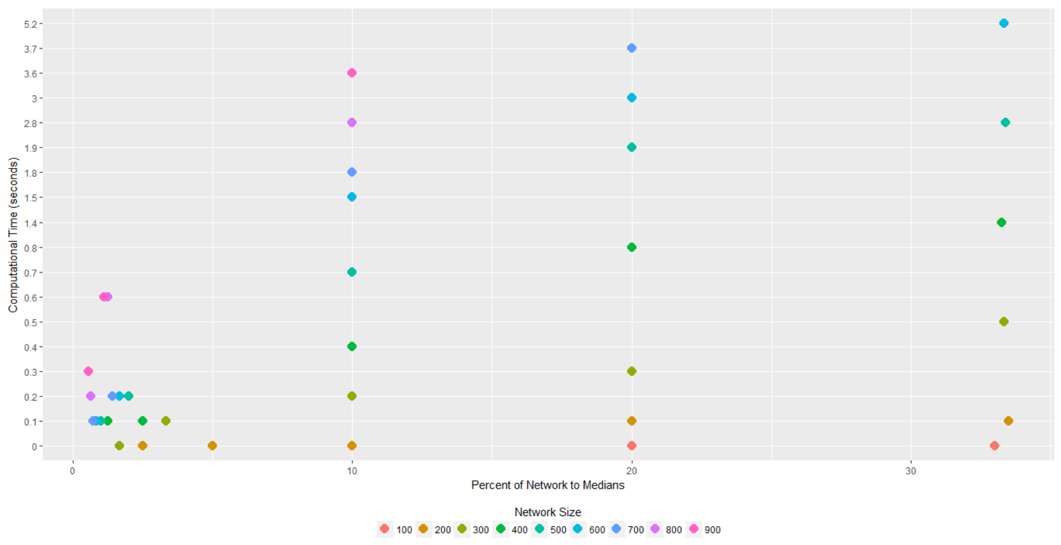

Figure 1: Percentage of Medians required in a Network versus Computational Time

The results obtained from using the greedy algorithm are reasonably good for problems that require a low number of medians. As expected, the accuracy declines as the number of medians required increases. This can be seen in figure 2. This can be attributed to a constructive heuristic never reassessing whether medians found in previous iterations should remain in the median set.

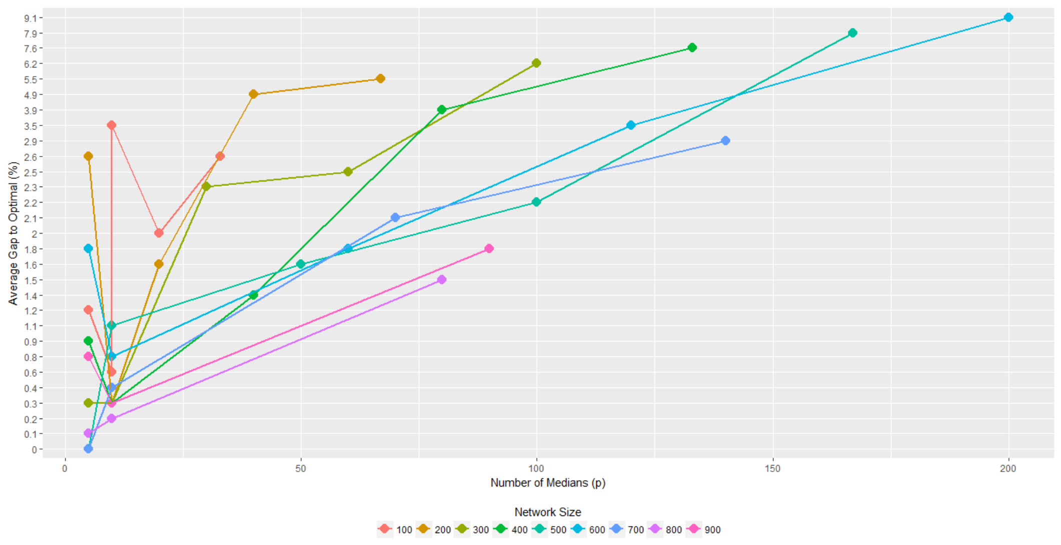

Figure 2: Percentage of Medians required in a Network versus Average Gap to the Optimal Solution

Overall, the greedy algorithm appears to be a good method to obtain a reasonable starting solution which can be improved upon by using a local search algorithm.

Table 1: Results of Greedy Algorithm (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (%)**
1	5	100	198	5819	0 ± 0.01	1.2 ± 0
2	10	100	193	4093	0 ± 0.01	0.6 ± 0
3	10	100	198	4250	0 ± 0.01	3.5 ± 0
4	20	100	196	3034	0 ± 0.01	2 ± 0.2
5	33	100	196	1355	0 ± 0.01	2.6 ± 0.69
6	5	200	786	7824	0 ± 0.02	2.6 ± 0
7	10	200	779	5631	0 ± 0.02	0.3 ± 0
8	20	200	792	4445	0 ± 0.02	1.6 ± 0.16
9	40	200	785	2734	0.1 ± 0.02	4.9 ± 0.48
10	67	200	786	1255	0.1 ± 0.02	5.5 ± 1.6
11	5	300	1772	7696	0 ± 0.02	0.3 ± 0
12	10	300	1758	6634	0.1 ± 0.03	0.3 ± 0
13	30	300	1760	4374	0.2 ± 0.03	2.3 ± 0
14	60	300	1771	2968	0.3 ± 0.03	2.5 ± 0.26
15	100	300	1754	1729	0.5 ± 0.04	6.2 ± 0.83
16	5	400	3153	8162	0.1 ± 0.03	0.9 ± 0
17	10	400	3142	6999	0.1 ± 0.04	0.3 ± 0
18	40	400	3134	4809	0.4 ± 0.07	1.4 ± 0.07
19	80	400	3134	2845	0.8 ± 0.06	3.9 ± 0.19
20	133	400	3144	1789	1.4 ± 0.07	7.6 ± 0.69
21	5	500	4909	9138	0.1 ± 0.04	0 ± 0
22	10	500	4896	8579	0.2 ± 0.04	1.1 ± 0
23	50	500	4903	4619	0.7 ± 0.05	1.6 ± 0.11
24	100	500	4914	2961	1.9 ± 0.36	2.2 ± 0.16
25	167	500	4894	1828	2.8 ± 0.12	7.9 ± 0.65
26	5	600	7069	9917	0.1 ± 0.03	1.8 ± 0
27	10	600	7072	8307	0.2 ± 0.04	0.8 ± 0
28	60	600	7054	4498	1.5 ± 0.1	1.8 ± 0.12
29	120	600	7042	3033	3 ± 0.17	3.5 ± 0.53
30	200	600	7042	1989	5.2 ± 0.22	9.1 ± 0.55
31	5	700	9601	10086	0.1 ± 0.03	0 ± 0
32	10	700	9584	9297	0.2 ± 0.04	0.4 ± 0
33	70	700	9616	4700	1.8 ± 0.1	2.1 ± 0.11
34	140	700	9585	3013	3.7 ± 0.13	2.9 ± 0.56
35	5	800	12548	10400	0.2 ± 0.06	0.1 ± 0
36	10	800	12560	9934	0.6 ± 0.03	0.2 ± 0
37	80	800	12564	5057	2.8 ± 0.25	1.5 ± 0.2
38	5	900	15898	11060	0.3 ± 0.02	0.8 ± 0
39	10	900	15896	9423	0.6 ± 0.04	0.3 ± 0
40	90	900	15879	5128	3.6 ± 0.22	1.8 ± 0.18

6.2 The Fast Greedy Algorithm

Whitaker [12] proposed the fast greedy algorithm. This algorithm differs from the greedy algorithm in that only the nodes that were re-assigned to a new median in the preceding iteration are investigated. From these nodes, the new median is determined to be the one that will maximise the decrease in the objective function value.

6.2.1 Pseudocode

Algorithm 2 Fast Greedy Algorithm

```
INPUT: Cost Matrix

STEP 0: Initialisation

    Set $P^* = \emptyset$; $I = \emptyset$; $k = 1$
        $c_j^0 = c_j^1 = 0$ for $j \in M$
        $u_i^0 = 0$; $u_i^1 = \infty$ for all $i = 1,2,\ldots,n$
        $I = I \cup i$ for $i = 1,2,\ldots,n$

STEP 1: For each free node that was reassigned to a new median in the previous iteration, the objective function value resulting from selecting this node as a median is determined. This objective function value is calculated by summing the cost of travel for each node in the network to either the closest median or the free node under investigation, whichever is smallest. For the first iteration, all nodes in the network are considered part of the reassigned set.

    $c_j^k = \sum_{i \in I} \min(d_{ij}, u_i^k) + c_j^{k-1} - \sum_{i \in I} \min(d_{ij}, u_i^{k-1}) \quad \forall j \in M, j \notin P^*$

STEP 2: The free node that will result in the smallest possible objective function value is identified, and the objective function value is updated.

    $S_r = \min_{j \notin P*}(c_j^k) \quad$ for $r \in M$, $r \notin P^*$
```

STEP 3: Add node r to the set of medians and then determine if the algorithm has completed the number of required iterations.

$$P^* = P^* \cup r$$

```
if $k = p$ then
go to STEP 5 [
else
Set $I = \emptyset$ and go to STEP 4 ]
end if
```

STEP 4: Increase the value of iteration parameter and update the cost of travel from any node to its closet median given that node r has been added to the median set.

$$\begin{aligned} k &= k + 1 \\ u_i^k &= \min(d_{ir},\, u_i^{k-1}) \quad \text{for } i = 1,2,3\ldots,n \end{aligned}$$

```
if $d_{ir} < u_i^{k-1}$ then [
$I = I \cup i$
go to STEP 1]
end if
```

STEP 5: Once p iterations are complete, set the final objective function value equal to the current iteration's objective function value and stop the algorithm.

$S^* = S_r$ and STOP

6.2.2 Complexity

The fast greedy algorithm has a complexity of at most $\mathcal{O}(n.p)$, since it is possible that almost all nodes are assigned to the set I during each iteration. It is expected that the performance of this algorithm would be better than that of the greedy algorithm as the I set from which a new median is selected each iteration would be typically be smaller than the network.

6.2.3 Results

The results obtained from running the fast greedy algorithm 50 times can be seen in table 2 and this algorithm proves true to its name. As seen in figure 3, the fast greedy algorithm produces results, on average, in less than 1 second for all test problems. Even test problems with very large networks, and networks requiring a high number of medians, the algorithm produces a solution set far quicker than the greedy algorithm.

Where the fast greedy fails is in the quality of the solution. At best, the algorithm provides a solution set with an associated objective function value that was 4.3% worse than the optimal objective function value and at worst the solution set's objective function value was over 500% worse.

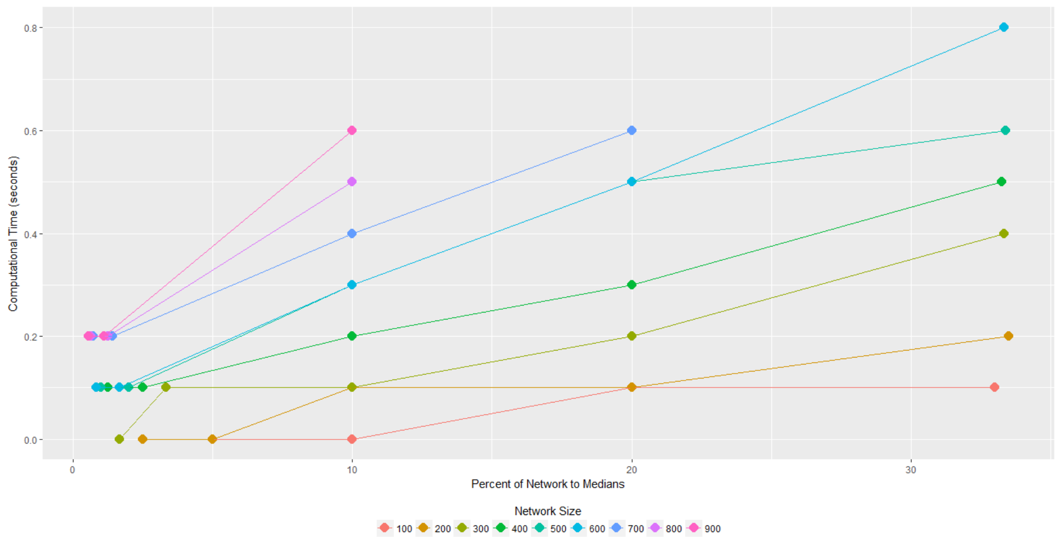

Figure 3: Percentage of Medians required in a Network versus Computational Time

This poor performance may not be an issue if these results are used as a starting solution for a local search heuristic but these solutions are certainly not to be used as a final solution unless there are legitimate reasons for doing so.

Table 2: Results of Fast Greedy Algorithm (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (%)**
1	5	100	198	5819	0 ± 0.01	20.3 ± 0
2	10	100	193	4093	0 ± 0.01	66.3 ± 0
3	10	100	198	4250	0 ± 0.01	64.9 ± 0
4	20	100	196	3034	0.1 ± 0.01	178.3 ± 0
5	33	100	196	1355	0.1 ± 0.01	376.4 ± 0
6	5	200	786	7824	0 ± 0.01	10.1 ± 0
7	10	200	779	5631	0 ± 0.01	41.8 ± 0
8	20	200	792	4445	0.1 ± 0.01	108.4 ± 0
9	40	200	785	2734	0.1 ± 0.01	221.2 ± 0
10	67	200	786	1255	0.2 ± 0.02	457.9 ± 0
11	5	300	1772	7696	0 ± 0.01	11.1 ± 0
12	10	300	1758	6634	0.1 ± 0.02	36.2 ± 0
13	30	300	1760	4374	0.1 ± 0.02	110.1 ± 0
14	60	300	1771	2968	0.2 ± 0.02	262.5 ± 0
15	100	300	1754	1729	0.4 ± 0.03	417.7 ± 0.4
16	5	400	3153	8162	0.1 ± 0.03	6.8 ± 0
17	10	400	3142	6999	0.1 ± 0.03	37.1 ± 0
18	40	400	3134	4809	0.2 ± 0.03	122.7 ± 0
19	80	400	3134	2845	0.3 ± 0.03	262.5 ± 0
20	133	400	3144	1789	0.5 ± 0.05	547 ± 0
21	5	500	4909	9138	0.1 ± 0.04	7.6 ± 0
22	10	500	4896	8579	0.1 ± 0.02	24.9 ± 0.06
23	50	500	4903	4619	0.3 ± 0.03	142.3 ± 0
24	100	500	4914	2961	0.5 ± 0.08	286.1 ± 0.2
25	167	500	4894	1828	0.6 ± 0.04	499 ± 0
26	5	600	7069	9917	0.1 ± 0.04	5.4 ± 0
27	10	600	7072	8307	0.1 ± 0.02	24.1 ± 0
28	60	600	7054	4498	0.3 ± 0.04	132.7 ± 0.55
29	120	600	7042	3033	0.5 ± 0.04	279.7 ± 0
30	200	600	7042	1989	0.8 ± 0.04	542.6 ± 0
31	5	700	9601	10086	0.2 ± 0.03	7.4 ± 0
32	10	700	9584	9297	0.2 ± 0.03	28.5 ± 0
33	70	700	9616	4700	0.4 ± 0.04	167.8 ± 0
34	140	700	9585	3013	0.6 ± 0.05	302.9 ± 0
35	5	800	12548	10400	0.2 ± 0.04	4.3 ± 0
36	10	800	12560	9934	0.2 ± 0.04	26.4 ± 0
37	80	800	12564	5057	0.5 ± 0.03	172.2 ± 0.02
38	5	900	15898	11060	0.2 ± 0.03	5.9 ± 0
39	10	900	15896	9423	0.2 ± 0.03	23.3 ± 0
40	90	900	15879	5128	0.6 ± 0.04	167.6 ± 0

6.3 The Stingy Algorithm

The stingy algorithm was suggested by Feldman, Lehrer, and Ray in their paper [13] published in 1966. Instead of starting with an empty median set and recursively adding a node until all p medians are assigned, the stingy algorithm begins with all nodes in the network being designated as a median. Nodes are removed one at a time until only required p medians remain in the median set.

6.3.1 Pseudocode

Algorithm 3 Stingy Algorithm

```
INPUT: Cost Matrix

STEP 0: Initialisation
    Set $P^* = M$; $P = \emptyset$ and $u_i = 0$ for all $i = 1,2,\ldots,n$

STEP 1: For each node that belongs to the median set, find the value of the objective function if this node were to be removed from the median set.
    $c_j^* = \sum_{i=1}^{n} \min_{j \in P^*, j \neq i} d_{ij}$

STEP 2: Identify which node results in the smallest possible objective function value.
Find node $r$ such that
    $S_r = \min_{j \in P*} (c_j^k) \quad \text{for } r \in P^*,\ r \notin P$

STEP 3: Remove this node from the set of medians.
    $P^* = P^* \backslash \{r\} \qquad P = P \cup \{r\}$

STEP 4: Update the iteration parameter.
    $k = k + 1$

STEP 5: Once $p$ iterations are complete, set the final objective function value equal to the current iteration's objective function value and stop the algorithm.
    $S^* = S_r$ and STOP
```

6.3.2 Complexity

Like the greedy algorithm, there are $\mathcal{O}(n)$ lookups in each iteration of the stingy algorithm. There will be $n - p$ iterations and as such this algorithm will have a complexity of $\mathcal{O}(n.(n-p))$.

For problems with a low number of required medians, this will be much slower than the greedy algorithm but for problems with a high number of required medians, this algorithm may prove useful.

6.3.3 Results

Table 3 gives the results from test problems 1 to 20 over 50 runs using the stingy algorithm. Only the first half of the test problems were able to complete before a timeout of 2 minutes was reached.

The time required by the stingy algorithm to find a feasible solution increased dramatically as the network size of the problem increased. This can be seen in figure 4. The expected decrease in computational time as the number of medians increases is visible, particularly in the test problem with 400 nodes, but does not offer much of a saving to justify the use of this algorithm to solve the p-median problem.

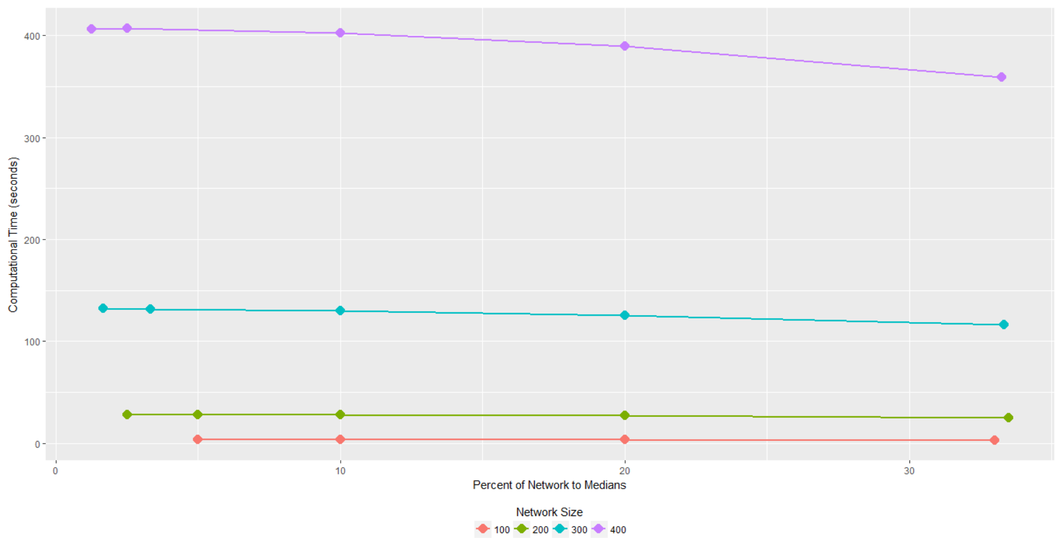

Figure 4: Percentage of Medians required in a Network versus Computational Time

Due to the excessive computational time taken by this algorithm when solving large networks, only the first twenty test problems were run and the results thereof can be seen in table 3.

The value of the objective function from the solution set found by the stingy algorithm is at best 99.2% worse than the optimal value and at worse, it is 173.2% worse than the optimal. So, like the fast greedy algorithm, the stingy algorithm is not

Table 3: Results of Stingy Algorithm (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (%)**
1	5	100	198	5819	3.3 ± 0.16	116.1 ± 0
2	10	100	193	4093	3.3 ± 0.11	131.9 ± 3.4
3	10	100	198	4250	3.3 ± 0.11	130.6 ± 0.29
4	20	100	196	3034	3.1 ± 0.12	108.5 ± 1.78
5	33	100	196	1355	2.9 ± 0.15	145 ± 8.07
6	5	200	786	7824	28.2 ± 0.47	99.2 ± 0
7	10	200	779	5631	28.1 ± 0.36	125.3 ± 0.81
8	20	200	792	4445	27.8 ± 0.34	131.3 ± 2.1
9	40	200	785	2734	26.9 ± 0.37	135.3 ± 2.34
10	67	200	786	1255	25 ± 0.77	173.2 ± 0
11	5	300	1772	7696	131.8 ± 0.38	154.2 ± 12.12
12	10	300	1758	6634	131.4 ± 0.23	163 ± 0
13	30	300	1760	4374	130.1 ± 0.44	149.6 ± 0
14	60	300	1771	2968	125.5 ± 0.26	138.7 ± 1.97
15	100	300	1754	1729	116.4 ± 0.38	125.8 ± 1.03
16	5	400	3153	8162	406.3 ± 0.17	149.7 ± 0
17	10	400	3142	6999	406.8 ± 1.41	136.8 ± 2.79
18	40	400	3134	4809	402.4 ± 3.86	135.7 ± 1.26
19	80	400	3134	2845	389.6 ± 0.69	124.3 ± 0.33
20	133	400	3144	1789	359 ± 0.19	128.2 ± 0

useful in any situation other than a problem with a small network size that requires a large number of medians.

6.4 Comparison of Results from Constructive Heuristics

Figure 5 shows the average gap between the solution's objective function value and the optimal solutions objective function for each of the forty test problems. It can be seen that the greedy algorithm was able to produce results that were very close to the optimal value with very slight decreases as the number of medians required increased.

The fast greedy algorithm produced adequate results for the test problems that demanded five medians. The massive difference in accuracy for the fast greedy algorithm is clearly visible in figure 5.

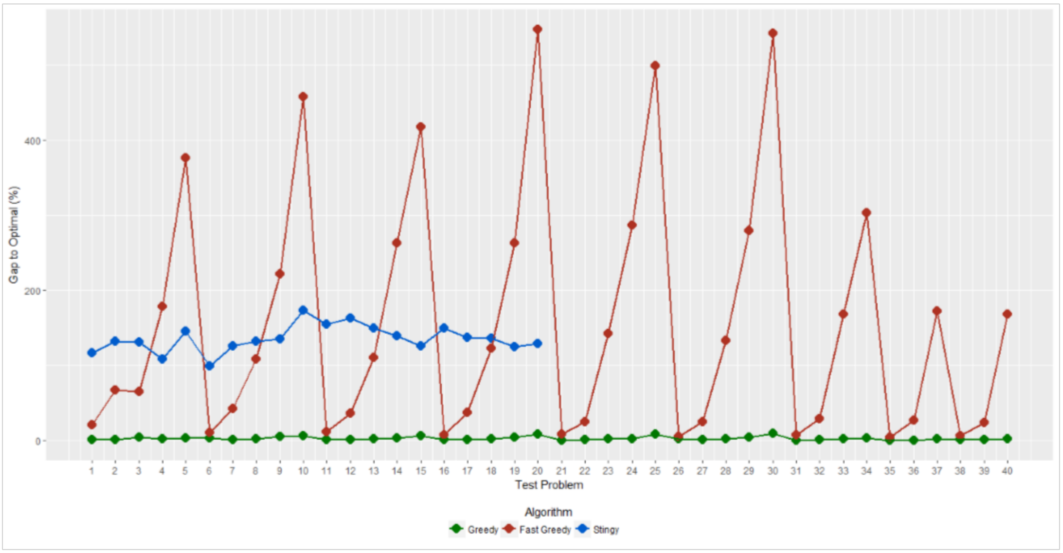

Figure 5: Average Gap to Optimal per Test Problem

In figure 6, the average computational time for the greedy algorithm, the fast greedy algorithm and the stingy algorithm is shown. It can clearly be seen that the stingy algorithm takes far longer than the other two algorithm and increases dramatically as the network size increases. It can be seen that the computational time for the stingy algorithm does decrease as the value of p increases.

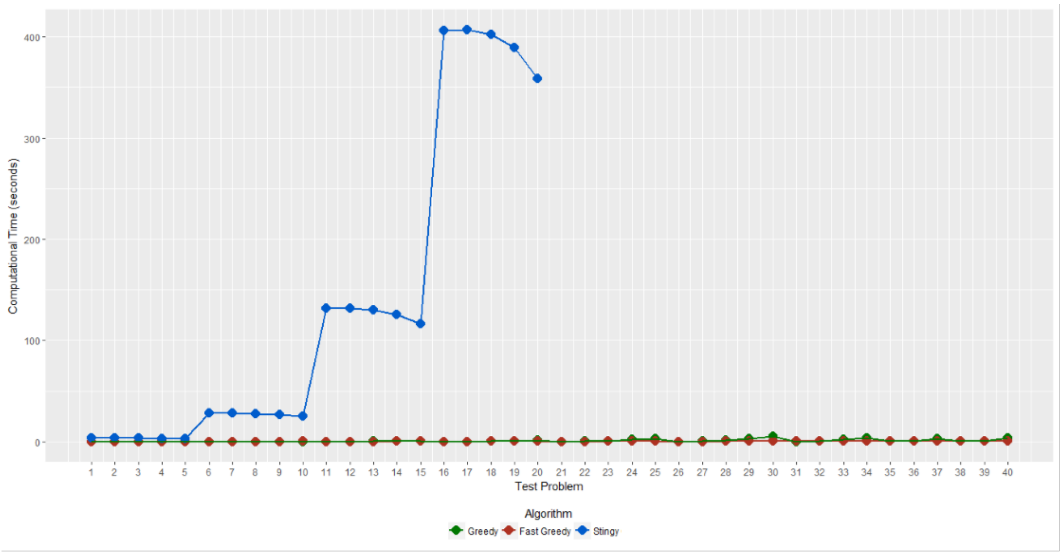

Figure 6: Average Computational Time per Test Problem for Greedy, Fast Greedy and Stingy Algorithms5

The computational times for the greedy algorithm and the fast greedy algorithm can be better compared in figure 7. In this graph, it can be seen that the fast greedy algorithm usually solves faster than the greedy algorithm especially as the network size and value of the required number of medians, p, increases.

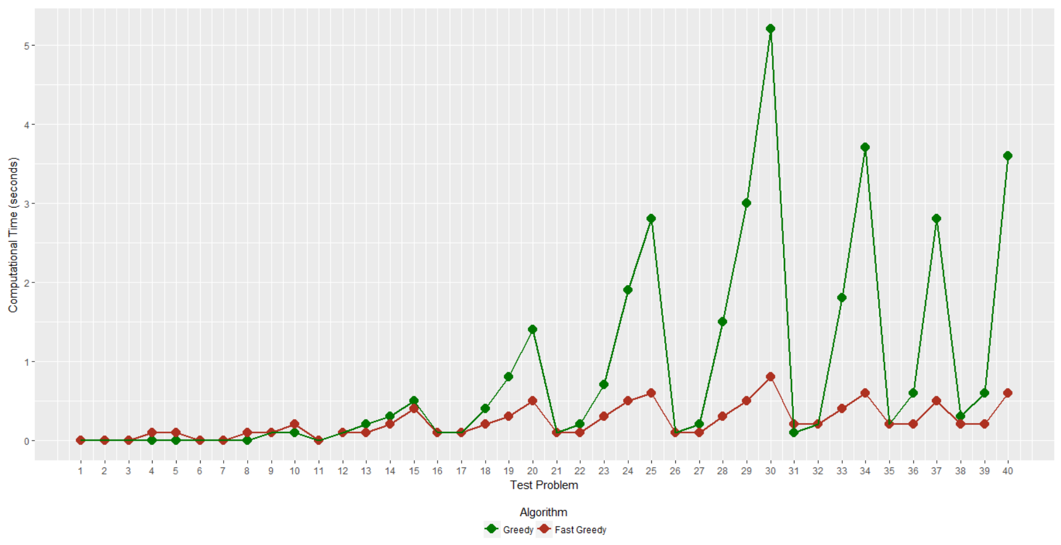

Figure 7: Average Computational Time per Test Problem for Greedy and Fast Greedy Algorithms

Figure 8 compares the average computational time for each test problem against the average accuracy of the solution produced. Figure 9 removes the results from the stingy algorithm to allow for the performance of the greedy and the fast greedy algorithms to be assessed easily.

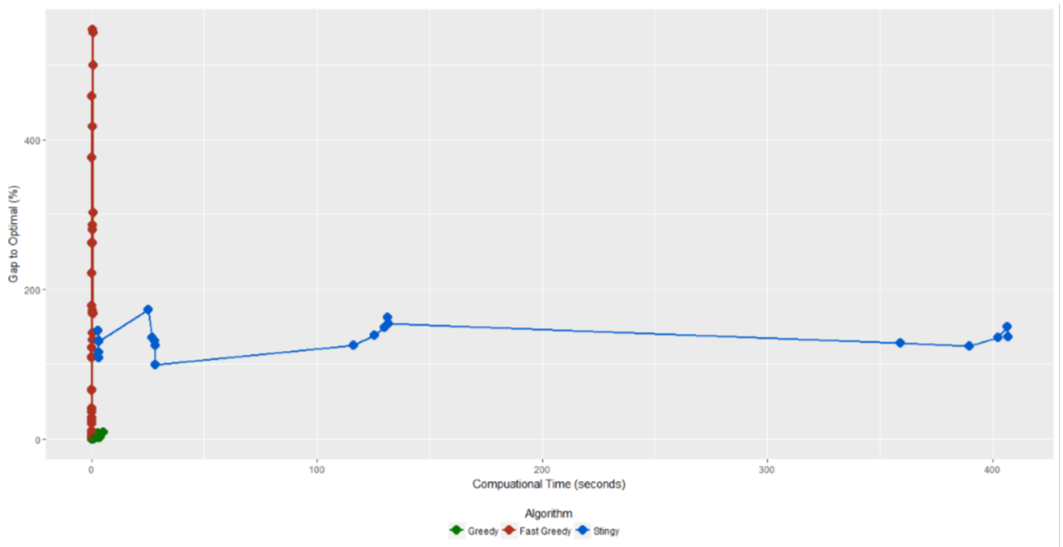

Figure 8: Average Computational Time against Average Gap to the Optimal Solution

It can be seen that the greedy algorithm provides a very accurate solution but does require more time to complete than the faster greedy algorithm when solving problems with large networks. The faster greedy algorithm completes the required number of iterations far quicker but at the loss of the accuracy of the solution.

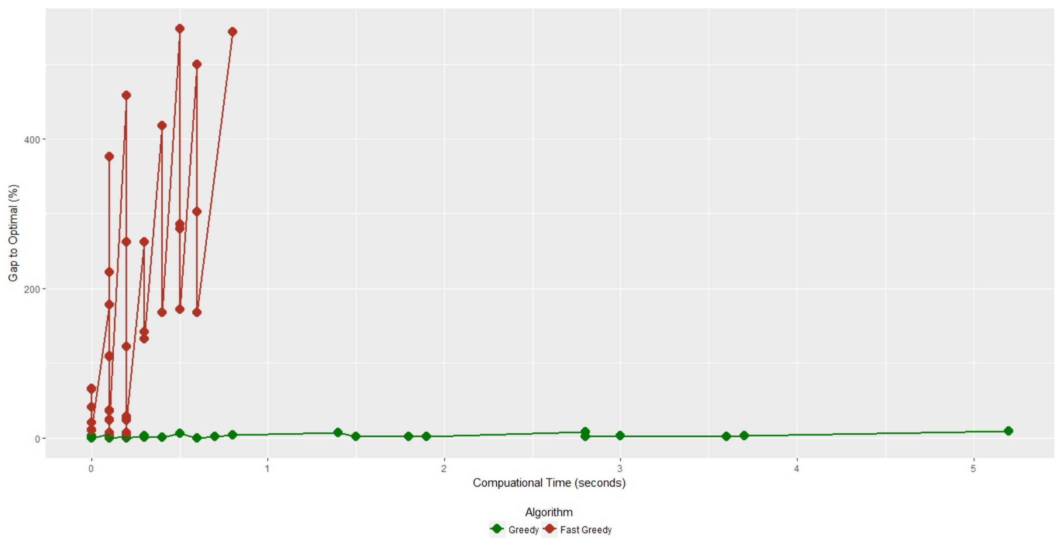

Figure 9: Average Computational Time against Average Gap to the Optimal Solution

Based on the work done by Whitaker [12], it was expected that the fast greedy algorithm would produce results quicker than the greedy algorithm. The trade-off in finding a solution faster would be an objective function value that is less accurate when compared to the optimal objective function value. This trade-off may be acceptable in certain situations.

7 Classical Heuristics: Local Search

A local search heuristic differs from a constructive heuristic in that it starts with a feasible solution and each iteration attempts to improve this solution by some deterministic method. This process continues until there is no better solution within reach of the current solution or some predetermined time-out is reached.

The algorithms analysed in this section were run once with a solution median set that was generated using the greedy algorithm described in subsection 6.1 and then a second time using a median set that was generated randomly from all the nodes in the network.

7.1 The Interchange Algorithm of Teitz and Bart

The following interchange algorithm was proposed by Teitz and Bart [14] and the amended algorithm was presented by Whitaker [12] fifteen years later. It is Whitaker's algorithm that was used in this study.

Using the input solution, the algorithm assesses what the change in objective function would be if one of the free nodes was changed with one of the median nodes. If this change results in a decreased objective function value, then the switch is made, and the algorithm then starts assessing whether there exist any free nodes that should be interchanged with the new median set. The process continues until no improvement in the objective function can be found.

7.1.1 Pseudocode

Algorithm 4 Interchange Algorithm of Teitz and Bart

```
INPUT: Cost Matrix; $P^*$; $S^*$

STEP 0: Initialisation

        Set $q = a = 0$; $k = 1$; $b = m - p$; $S = S^*$ and $P = M - P$

STEP 1: For all nodes in the network, find the closest and second medians.

        $u_i^k = d_{ix} = \min_{j \in P^*}$
        $w_i^k = d_{iy} = \min_{j \in P^*, j \neq x}$

STEP 2: The value of $q$ is updated until it is equal to the number of nodes in the
network minus the number of nodes required for the median.
if $q = b$ then
go to STEP 5 [
else
Set $q = q + 1$ and go to STEP 3 ]
end if

STEP 3: Determine which node currently in the median set, $P^*$, should possibly
be interchanged with the $q^{\text{th}}$ node in the set of free nodes. The change in the
objective function value if this interchange were to proceed is calculated.

        $r = P_q$
        $S_{rt} = \min_{j \in P^*} [\sum_{i \in I} [\min(d_{ir}, u_i^k) - u_i^k] + \sum_{i \in J} \in I[\min(d_{ir}, w_i^k) - u_i^k]]$

        where $I = \{\text{all } i \in G : d_{ij} > u_i^k\}$ and $J = \{\text{all } i \in G : d_{ij} = u_i^k\}$
```

```
STEP 4: If the proposed node switch from STEP 3 results in a decrease of the
objective function value, then proceed with the interchange and update the ob-
jective function value as well as the iteration parameter. If the interchange would
result in an increase in the objective function value, then go back to STEP 2 and
update q.
if $S_{rt} \geq 0$ then
go to STEP 2 [
else if $S_{rt} < 0$ then
    $k = k + 1$
    $S^* = S^* + S_{rt}$
    $P_q = t$
    $P^* = P^* \backslash \{t\} \cup \{r\}$
go to STEP 1 ]
end if

STEP 5: If, after all nodes in the free node set have been investigated (i.e. $q = b$),
increase the value of $a$ by one and then assess whether the objective function
improved in the prior iteration. If there was an improvement, then repeat the
algorithm again but if there was no improvement stop.
    $a = a + 1$
if $S > S^*$ then
Set $q = 0$ and go to STEP 2 [
else if $S \leq S^*$ then
Set $S = S^*$ and STOP ]
end if
```

7.1.2 Results

The results from running the interchange algorithm of Teitz and Bart for 50 times with an initial starting solution generated from the greedy algorithm is given in table 4 while the results from using a random feasible solution can be seen in table 5.

Staring with the computational time, the figure 10 shows the average total time taken by each run for both starting solutions. The random solution generally takes longer to get to a final solution for the test problems that have a high p-value. This result is unsurprising as the value of p increases, along the the network size, there are far more options to be considered when deciding which free node and median to swap. It would be interesting to see how many unique swaps have been made compared to how many repetitive swaps but this has not been covered in this study.

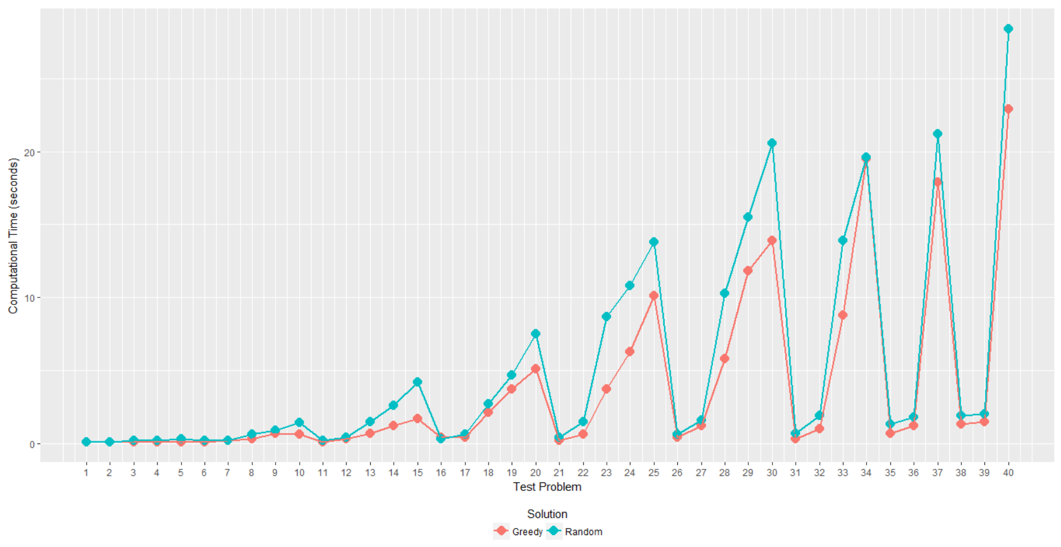

Figure 10: Average Computational Time of each Test Problem

When looking at the accuracy of the solution in figure 11, there is no better starting solution. In some cases, the greedy starting solution does well and in other cases, the random starting solution is best. The worst case for both the greedy solution and the random is still better than results from either the fast greedy algorithm and the stingy algorithm.

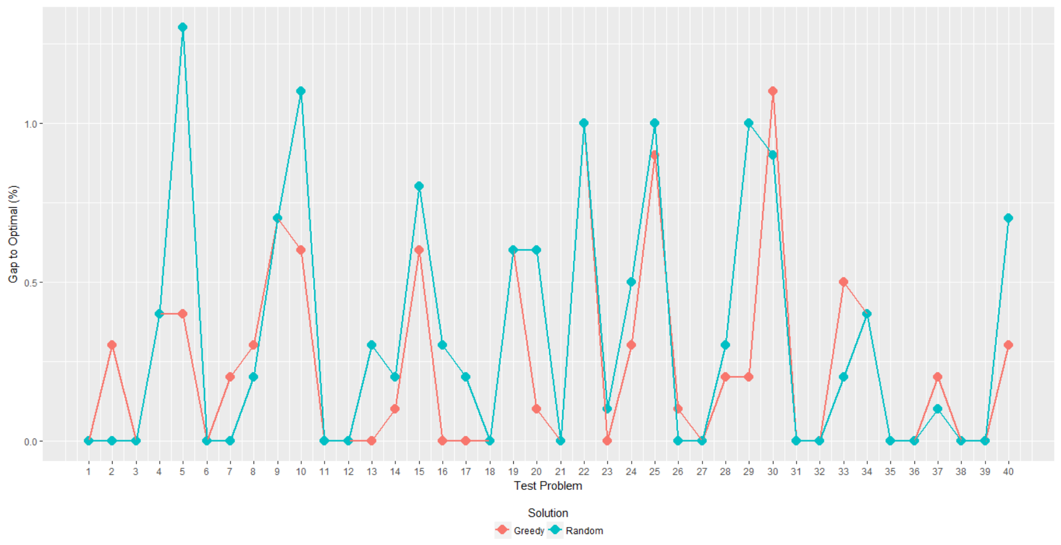

Figure 11: Average Gap to Optimal Value of each Test Problem

Test problem pmed28 ($n = 600$ and $p = 60$) resulted in the highest number of iterations for both starting solutions with a greedy solution requiring 38 iterations and the random solution requiring 224 iterations. This is a remarkable difference and illustrates the importance of acquiring a good starting solution prior to using the Teitz and Bart interchange algorithm.

On average, the random starting solution required more iterations per run than the greedy starting solution. This can be seen in figure 12.

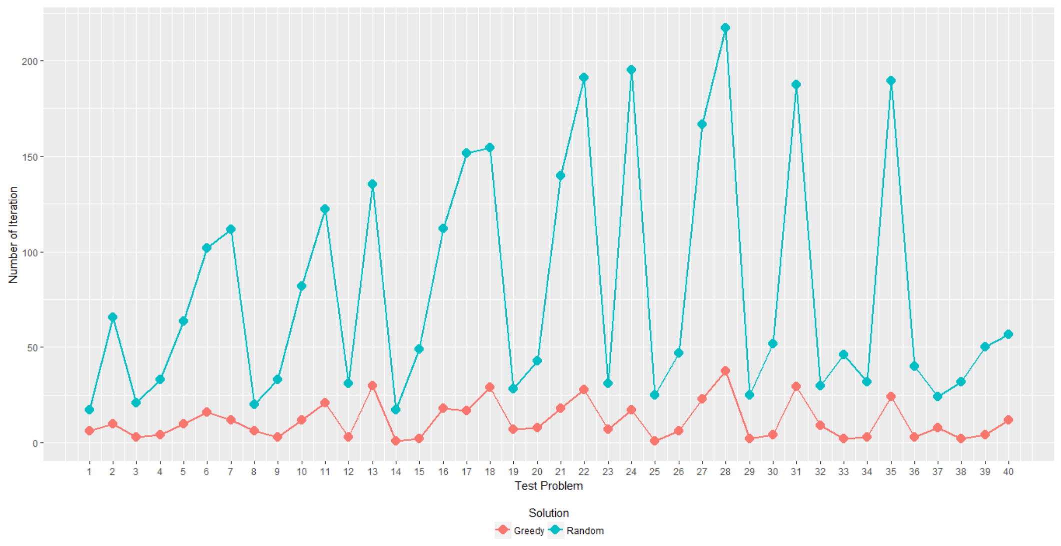

Figure 12: Average Number Iterations of each Test Problem

Table 4: Results of the Interchange Algorithm of Teitz and Bart - Greedy (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (percent)**	**Num. of Iterations**
1	5	100	198	5819	0.1 ± 0.02	0 ± 0	6 ± 0
2	10	100	193	4093	0.1 ± 0.01	0.3 ± 0	10 ± 0
3	10	100	198	4250	0.1 ± 0.02	0 ± 0	3 ± 0
4	20	100	196	3034	0.1 ± 0.02	0.4 ± 0	4 ± 0
5	33	100	196	1355	0.1 ± 0.02	0.4 ± 0	10 ± 0
6	5	200	786	7824	0.1 ± 0.02	0 ± 0	16 ± 0
7	10	200	779	5631	0.2 ± 0.02	0.2 ± 0	12 ± 0
8	20	200	792	4445	0.3 ± 0.02	0.3 ± 0	6 ± 0
9	40	200	785	2734	0.7 ± 0.04	0.7 ± 0	3 ± 0
10	67	200	786	1255	0.6 ± 0.02	0.6 ± 0	12 ± 0
11	5	300	1772	7696	0.1 ± 0.02	0 ± 0	21 ± 0
12	10	300	1758	6634	0.3 ± 0.03	0 ± 0	3 ± 0
13	30	300	1760	4374	0.7 ± 0.04	0 ± 0	30 ± 0
14	60	300	1771	2968	1.2 ± 0.07	0.1 ± 0	1 ± 0
15	100	300	1754	1729	1.7 ± 0.07	0.6 ± 0	2 ± 0
16	5	400	3153	8162	0.4 ± 0.03	0 ± 0	18 ± 0
17	10	400	3142	6999	0.4 ± 0.02	0 ± 0	16.8 ± 1.03
18	40	400	3134	4809	2.1 ± 0.08	0 ± 0	28.9 ± 1.66
19	80	400	3134	2845	3.7 ± 0.12	0.6 ± 0	7 ± 0
20	133	400	3144	1789	5.1 ± 0.19	0.1 ± 0	8 ± 0
21	5	500	4909	9138	0.2 ± 0.02	0 ± 0	17.8 ± 0.42
22	10	500	4896	8579	0.6 ± 0.04	1 ± 0	27.9 ± 0.32
23	50	500	4903	4619	3.7 ± 0.1	0 ± 0	7 ± 0
24	100	500	4914	2961	6.3 ± 0.19	0.3 ± 0	17 ± 0
25	167	500	4894	1828	10.1 ± 1.43	0.9 ± 0.07	1 ± 0
26	5	600	7069	9917	0.4 ± 0.03	0.1 ± 0	6 ± 0
27	10	600	7072	8307	1.2 ± 0.06	0 ± 0	23 ± 0
28	60	600	7054	4498	5.8 ± 0.2	0.2 ± 0	37.7 ± 0.48
29	120	600	7042	3033	11.8 ± 1.65	0.2 ± 0	2 ± 0
30	200	600	7042	1989	13.9 ± 0.44	1.1 ± 0	4 ± 0
31	5	700	9601	10086	0.3 ± 0.02	0 ± 0	29.3 ± 0.95
32	10	700	9584	9297	1 ± 0.05	0 ± 0	9 ± 0
33	70	700	9616	4700	8.8 ± 0.21	0.5 ± 0	2 ± 0
34	140	700	9585	3013	19.5 ± 1.88	0.4 ± 0.01	3 ± 0
35	5	800	12548	10400	0.7 ± 0.03	0 ± 0	24 ± 0
36	10	800	12560	9934	1.2 ± 0.05	0 ± 0	3 ± 0
37	80	800	12564	5057	17.9 ± 3.65	0.2 ± 0.01	8 ± 0
38	5	900	15898	11060	1.3 ± 0.06	0 ± 0	2 ± 0
39	10	900	15896	9423	1.5 ± 0.05	0 ± 0	4 ± 0
40	90	900	15879	5128	22.9 ± 0.3	0.3 ± 0	12 ± 0

Table 5: Results of the Interchange Algorithm of Teitz and Bart - Random (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (percent)**	**Num. of Iterations**
1	5	100	198	5819	0.1 ± 0.01	0 ± 0	17 ± 0
2	10	100	193	4093	0.1 ± 0.02	0 ± 0	65.5 ± 0.53
3	10	100	198	4250	0.2 ± 0.01	0 ± 0	21 ± 0
4	20	100	196	3034	0.2 ± 0.02	0.4 ± 0	33 ± 0
5	33	100	196	1355	0.3 ± 0.02	1.3 ± 0	63.6 ± 0.52
6	5	200	786	7824	0.2 ± 0.03	0 ± 0	101.8 ± 1.23
7	10	200	779	5631	0.2 ± 0.02	0 ± 0	111.5 ± 3.14
8	20	200	792	4445	0.6 ± 0.03	0.2 ± 0	20 ± 0
9	40	200	785	2734	0.9 ± 0.06	0.7 ± 0.02	33 ± 0
10	67	200	786	1255	1.4 ± 0.19	1.1 ± 0.16	81.8 ± 0.42
11	5	300	1772	7696	0.2 ± 0.02	0 ± 0	122.2 ± 3.19
12	10	300	1758	6634	0.4 ± 0.02	0 ± 0	31 ± 0
13	30	300	1760	4374	1.5 ± 0.08	0.3 ± 0	135.1 ± 1.45
14	60	300	1771	2968	2.6 ± 0.27	0.2 ± 0.05	17 ± 0
15	100	300	1754	1729	4.2 ± 0.78	0.8 ± 0.17	49 ± 0
16	5	400	3153	8162	0.3 ± 0.02	0.3 ± 0	112.1 ± 1.45
17	10	400	3142	6999	0.6 ± 0.02	0.2 ± 0	151.4 ± 1.65
18	40	400	3134	4809	2.7 ± 0.1	0 ± 0	154.6 ± 1.84
19	80	400	3134	2845	4.7 ± 0.25	0.6 ± 0.09	28 ± 0
20	133	400	3144	1789	7.5 ± 0.83	0.6 ± 0.13	43 ± 0
21	5	500	4909	9138	0.4 ± 0.02	0 ± 0	139.6 ± 1.26
22	10	500	4896	8579	1.5 ± 0.06	1 ± 0	190.9 ± 1.6
23	50	500	4903	4619	8.7 ± 0.21	0.1 ± 0	31.2 ± 1.55
24	100	500	4914	2961	10.8 ± 0.65	0.5 ± 0.13	195.2 ± 1.81
25	167	500	4894	1828	13.8 ± 1.62	1 ± 0.2	25 ± 0
26	5	600	7069	9917	0.6 ± 0.03	0 ± 0	47 ± 0
27	10	600	7072	8307	1.6 ± 0.05	0 ± 0	166.7 ± 2.26
28	60	600	7054	4498	10.3 ± 0.8	0.3 ± 0.08	217 ± 3.65
29	120	600	7042	3033	15.5 ± 1.9	1 ± 0.07	25 ± 0
30	200	600	7042	1989	20.6 ± 2.98	0.9 ± 0.2	52 ± 0
31	5	700	9601	10086	0.7 ± 0.03	0 ± 0	187.5 ± 2.95
32	10	700	9584	9297	1.9 ± 0.09	0 ± 0	30 ± 0
33	70	700	9616	4700	13.9 ± 0.74	0.2 ± 0.08	46 ± 0
34	140	700	9585	3013	19.6 ± 1.63	0.4 ± 0.07	32 ± 0
35	5	800	12548	10400	1.3 ± 0.06	0 ± 0	189.5 ± 9.01
36	10	800	12560	9934	1.8 ± 0.08	0 ± 0	40 ± 0
37	80	800	12564	5057	21.2 ± 3.31	0.1 ± 0.04	24 ± 0
38	5	900	15898	11060	1.9 ± 0.06	0 ± 0	32 ± 0
39	10	900	15896	9423	2 ± 0.09	0 ± 0	50 ± 0
40	90	900	15879	5128	28.4 ± 7.87	0.7 ± 0.24	56.6 ± 0.52

7.2 The Fast Interchange Algorithm

The fast interchange algorithm, proposed by Whitaker [12], is based on the Teitz and Bart algorithm from subsection 7.1. When searching for a free node that should be swapped with a node in the median set, only free nodes that have yet to be examined are considered.

7.2.1 Pseudocode

Algorithm 5 Fast Interchange Algorithm

```
INPUT: Cost Matrix; $P^*$; $S^*$

STEP 0: Initialisation

    Set $k = z = q = 1$ and $b = m - p$

    Define $P = M - P^*$

For each node in the network, determine the distance to both the closest node in the median set as well as the second closest node in the median set.
For $i = 1,2,\ldots,n$, find $x$ and $y$ where $x$, $y \in P^*$ such that

    $u_i^1 = d_{ix} = \min_{j \in P^*} d_{ij}$
    $w_i^1 = d_{iy} = \min_{j \in P^*, j \neq x} d_{ij}$

STEP 1: Calculate the change in the objective function if one node from the median set were to be interchanged with a free node.
Set $r = P_q$ and find some node $t$, $t \in P^*$ such that

    $S_{rt} = \left[\sum_{i \in I} d_{ir} - u_i^k\right] + \min_{j \in P^*}\left[\sum_{i \in K_j} [\min(d_{ir},w_i^k) - u_i^k]\right]$
    where $I = \{\text{all } i \in G : d_{ir} < u_i^k\}$
    and $K_j = \{\text{all } i \in G : d_{ir} \geq u_i^k \text{ and } d_{ij} - u_i^k\}$

STEP 2: If the objective function value were to increase by the proposed interchange of nodes, then go to STEP 4. If the proposed change of nodes results in a decrease of the objective function value, then make the change, update iterative parameters and the objective function and go to STEP 3.
```

```
if $S_{rt} \geq 0$ then
go to STEP 4 [
else if $S_{rt} < 0$ then

    $k = k + 1 \quad S^* = S * + S_{rt} \quad P_q = \{t\} \quad P^* = P^* \backslash \{t\} \cup \{r\}$ and go to STEP 3

]
end if

STEP 3: Update the cost of travel for each node in the network to the closest and
second closest medians.
if $d_{it} > u_i^{k-1}$ then

    $u_i^k = \min(d_{ir}, u_k^{k-1})$

let $s \in P^*$ be a median for node $i$ such that $d_{is} = u_i^k$

        set $w_i^k = u_i^{k-1}$ if $d_{ir} \leq u_i^{k-1}$
    or set $w_i^k = \min(d_{ir}, w_i^{k-1})$ if $d_{ir} > u_i^{k-1}$ and $d_{it} > w_i^{k-1}$
    or set $w_i^k = \min_{j \in P^*, j \neq s} d_{ij}$ if $d_{ir} > u_i^{k-1}$ and $d_{it} = w_i^{k-1}$

[
else if $d_{it} = u_i^{k-1}$ then

    $u_i^k = \min(d_{ir}, w_k^{k-1})$

let $s \in P^*$ be a median for node $i$ such that $d_{is} = u_i^k$

    set $w_i^k = w_i^{k-1}$ if $d_{ir} \leq w_i^{k-1}$ or set $w_i^k = \min_{j \in P^*, j \neq s} d_{ij}$ if $d_{ir} > w_i^{k-1}$

]
end if

STEP 4: Once the algorithm has assessed all free nodes, go to STEP 5. Otherwise,
increase the iterative parameters and go back to STEP 1.
if $z = b$ then
go to STEP 5 [
else if $z < b$ then

    $z = z + 1$, $q = q + 1$ and go to STEP 1

]
end if
```

STEP 5: If the algorithm finds that the current median set cannot be improved upon, then find the percent of free nodes that were assessed for a possible exchange from the free node set to the median set and stop.

$a = \frac{q}{b}$ and STOP

7.2.2 Results

The results obtained from running the fast interchange algorithm 50 times starting with a solution generated via the greedy algorithm can be seen in table 6 and the results obtained from the fast interchange starting with a random feasible solution are given in table 7.

As with all algorithms thus far, the average computational time increases as the size of the network increases. This can be seen in figure 13. Unlike the Teitz and Bart interchange algorithm the random starting solution is only marginally slower than the greedy starting solution.

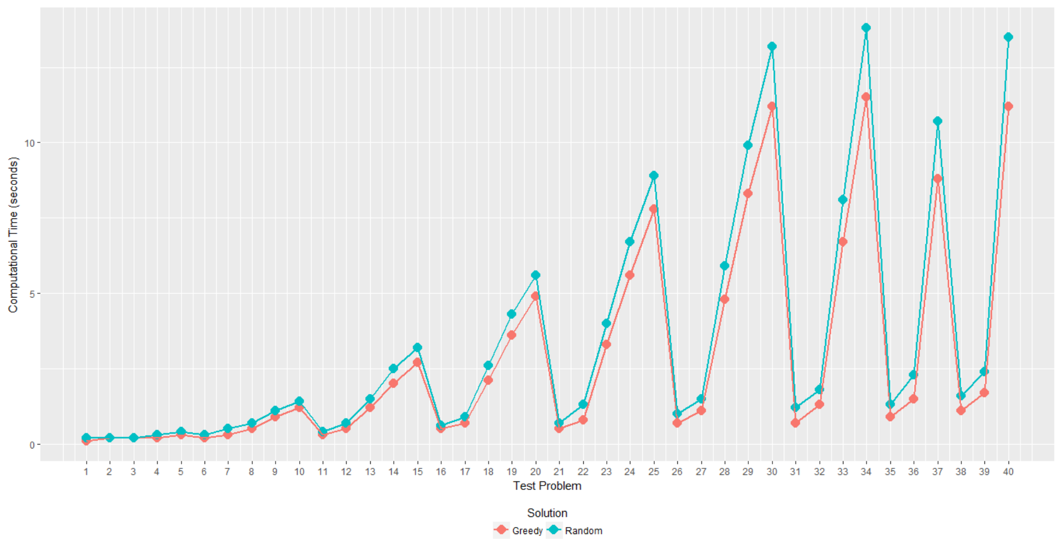

Figure 13: Percentage of Medians required in a Network versus Computational Time

In figure 14 it can be seen that like the Teitz and Bart interchange algorithm, the quality of the final solution varied considerably. For some test problems, the random starting solution was able to reach a near-optimal final solution when the greedy starting solution was not, but for other test problems the random starting solution was far worse than the final solution from a greedy starting solution.

Overall, the greedy starting solution provides a final solution than the random starting solution.

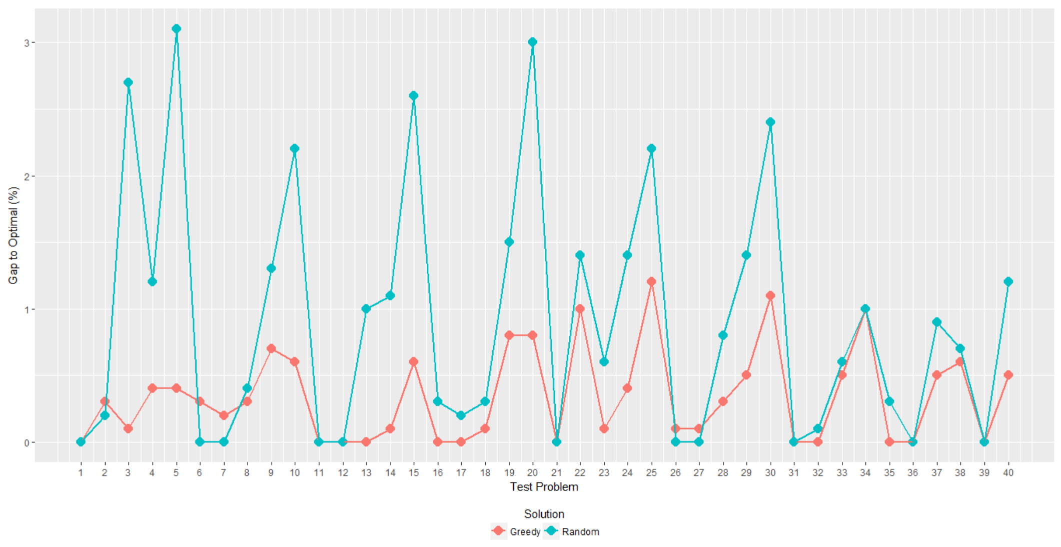

Figure 14: Average Gap to Optimal Value of each Test Problem

The average number of iterations required for each of the starting solutions over the 50 runs can be seen in figure 15. For this algorithm, the test problem that resulted in the maximum average number of iterations was pmed34 ($n = 140$ and $p = 700$). The greedy starting solution required on average 28.3 iterations until a final solution was found while the random starting solution required more than 7 times the number of iterations than the greedy starting solution. Neither starting solution resulted in finding an optimal solution in any of the 50 runs.

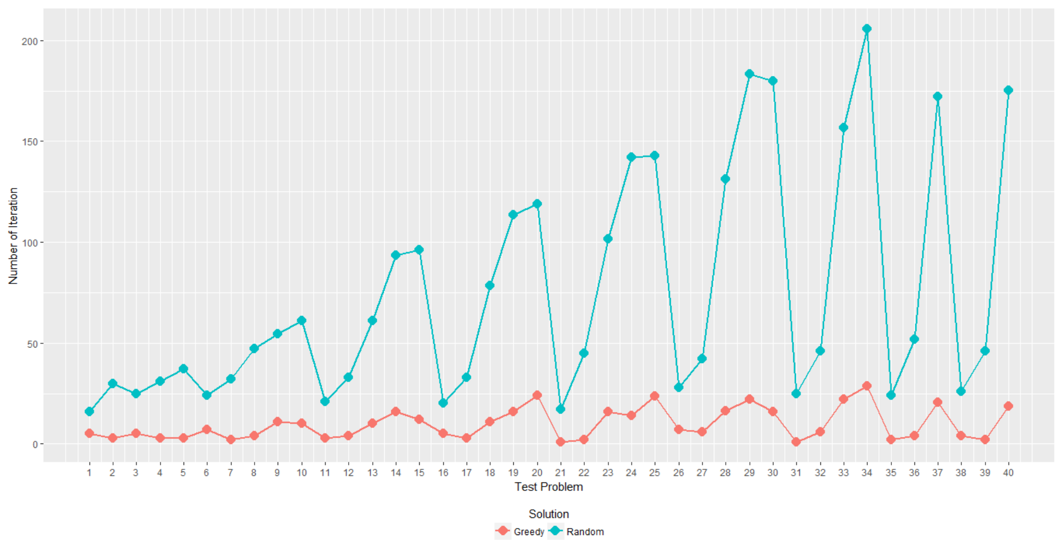

Figure 15: Average Number Iterations for each Test Problem

Table 6: Results of the Fast Interchange Algorithm - Greedy (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (percent)**	**Num. of Iterations**
1	5	100	198	5819	0.1 ± 0.02	0 ± 0	5 ± 0
2	10	100	193	4093	0.2 ± 0.02	0.3 ± 0	3 ± 0
3	10	100	198	4250	0.2 ± 0.02	0.1 ± 0	5 ± 0
4	20	100	196	3034	0.2 ± 0.02	0.4 ± 0	3 ± 0
5	33	100	196	1355	0.3 ± 0.03	0.4 ± 0	3 ± 0
6	5	200	786	7824	0.2 ± 0.02	0.3 ± 0	7 ± 0
7	10	200	779	5631	0.3 ± 0.02	0.2 ± 0	2 ± 0
8	20	200	792	4445	0.5 ± 0.03	0.3 ± 0	4 ± 0
9	40	200	785	2734	0.9 ± 0.05	0.7 ± 0	11 ± 0
10	67	200	786	1255	1.2 ± 0.05	0.6 ± 0	10 ± 0
11	5	300	1772	7696	0.3 ± 0.02	0 ± 0	3 ± 0
12	10	300	1758	6634	0.5 ± 0.04	0 ± 0	4 ± 0
13	30	300	1760	4374	1.2 ± 0.06	0 ± 0	10 ± 0
14	60	300	1771	2968	2 ± 0.08	0.1 ± 0	16 ± 0
15	100	300	1754	1729	2.7 ± 0.11	0.6 ± 0	12 ± 0
16	5	400	3153	8162	0.5 ± 0.03	0 ± 0	5 ± 0
17	10	400	3142	6999	0.7 ± 0.02	0 ± 0	3 ± 0
18	40	400	3134	4809	2.1 ± 0.09	0.1 ± 0	11 ± 0
19	80	400	3134	2845	3.6 ± 0.11	0.8 ± 0	16 ± 0
20	133	400	3144	1789	4.9 ± 0.15	0.8 ± 0	24 ± 0
21	5	500	4909	9138	0.5 ± 0.04	0 ± 0	1 ± 0
22	10	500	4896	8579	0.8 ± 0.02	1 ± 0	2 ± 0
23	50	500	4903	4619	3.3 ± 0.14	0.1 ± 0	16 ± 0
24	100	500	4914	2961	5.6 ± 0.2	0.4 ± 0.03	14 ± 0
25	167	500	4894	1828	7.8 ± 0.22	1.2 ± 0.08	23.8 ± 0.48
26	5	600	7069	9917	0.7 ± 0.04	0.1 ± 0	7 ± 0
27	10	600	7072	8307	1.1 ± 0.07	0.1 ± 0	6 ± 0
28	60	600	7054	4498	4.8 ± 0.22	0.3 ± 0	16.5 ± 0.51
29	120	600	7042	3033	8.3 ± 0.22	0.5 ± 0	21.9 ± 0.35
30	200	600	7042	1989	11.2 ± 0.25	1.1 ± 0	16 ± 0
31	5	700	9601	10086	0.7 ± 0.04	0 ± 0	1 ± 0
32	10	700	9584	9297	1.3 ± 0.06	0 ± 0	6 ± 0
33	70	700	9616	4700	6.7 ± 0.18	0.5 ± 0	22 ± 0
34	140	700	9585	3013	11.5 ± 0.29	1 ± 0.03	28.3 ± 0.47
35	5	800	12548	10400	0.9 ± 0.03	0 ± 0	2 ± 0
36	10	800	12560	9934	1.5 ± 0.08	0 ± 0	4 ± 0
37	80	800	12564	5057	8.8 ± 0.24	0.5 ± 0.07	20.8 ± 1.33
38	5	900	15898	11060	1.1 ± 0.05	0.6 ± 0	4 ± 0
39	10	900	15896	9423	1.7 ± 0.07	0 ± 0	2 ± 0
40	90	900	15879	5128	11.2 ± 0.21	0.5 ± 0.02	18.5 ± 0.5

Table 7: Results of the Fast Interchange Algorithm - Random (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (percent)**	**Num. of Iterations**
1	5	100	198	5819	0.2 ± 0.02	0 ± 0	16 ± 0
2	10	100	193	4093	0.2 ± 0.02	0.2 ± 0	30 ± 0
3	10	100	198	4250	0.2 ± 0.02	2.7 ± 0.38	26 ± 1
4	20	100	196	3034	0.3 ± 0.02	1.2 ± 0	31 ± 0
5	33	100	196	1355	0.4 ± 0.02	3.1 ± 0	37 ± 0
6	5	200	786	7824	0.3 ± 0.02	0 ± 0	24 ± 0
7	10	200	779	5631	0.5 ± 0.03	0 ± 0	32 ± 0
8	20	200	792	4445	0.7 ± 0.03	0.4 ± 0	47 ± 0
9	40	200	785	2734	1.1 ± 0.06	1.3 ± 0.02	55 ± 0.5
10	67	200	786	1255	1.4 ± 0.06	2.2 ± 0	61 ± 0
11	5	300	1772	7696	0.4 ± 0.04	0 ± 0	21 ± 0
12	10	300	1758	6634	0.7 ± 0.02	0 ± 0	33 ± 0
13	30	300	1760	4374	1.5 ± 0.06	1 ± 0.07	62 ± 0.98
14	60	300	1771	2968	2.5 ± 0.12	1.1 ± 0.04	95 ± 1.3
15	100	300	1754	1729	3.2 ± 0.13	2.6 ± 0.33	98 ± 1.02
16	5	400	3153	8162	0.6 ± 0.02	0.3 ± 0	20 ± 0
17	10	400	3142	6999	0.9 ± 0.04	0.2 ± 0	33 ± 0
18	40	400	3134	4809	2.6 ± 0.16	0.3 ± 0.01	79 ± 0.46
19	80	400	3134	2845	4.3 ± 0.13	1.5 ± 0.12	116 ± 1.5
20	133	400	3144	1789	5.6 ± 0.17	3 ± 0.16	122 ± 1.13
21	5	500	4909	9138	0.7 ± 0.04	0 ± 0	17 ± 0
22	10	500	4896	8579	1.3 ± 0.06	1.4 ± 0	45 ± 0
23	50	500	4903	4619	4 ± 0.14	0.6 ± 0.1	102 ± 0.93
24	100	500	4914	2961	6.7 ± 0.2	1.4 ± 0.07	144 ± 1.84
25	167	500	4894	1828	8.9 ± 0.25	2.2 ± 0.24	144 ± 1.24
26	5	600	7069	9917	1 ± 0.05	0 ± 0	28 ± 0
27	10	600	7072	8307	1.5 ± 0.08	0 ± 0	42 ± 0
28	60	600	7054	4498	5.9 ± 0.2	0.8 ± 0.07	133 ± 1.25
29	120	600	7042	3033	9.9 ± 0.24	1.4 ± 0.1	187 ± 1.95
30	200	600	7042	1989	13.2 ± 0.22	2.4 ± 0.22	184 ± 2
31	5	700	9601	10086	1.2 ± 0.05	0 ± 0	25 ± 0
32	10	700	9584	9297	1.8 ± 0.07	0.1 ± 0	46 ± 0
33	70	700	9616	4700	8.1 ± 0.15	0.6 ± 0.12	161 ± 0.99
34	140	700	9585	3013	13.8 ± 0.25	1 ± 0.17	211 ± 2.8
35	5	800	12548	10400	1.3 ± 0.05	0.3 ± 0	24 ± 0
36	10	800	12560	9934	2.3 ± 0.1	0 ± 0	52 ± 0
37	80	800	12564	5057	10.7 ± 0.25	0.9 ± 0.1	178 ± 2.69
38	5	900	15898	11060	1.6 ± 0.06	0.7 ± 0	26 ± 0
39	10	900	15896	9423	2.4 ± 0.1	0 ± 0	46 ± 0
40	90	900	15879	5128	13.5 ± 0.25	1.2 ± 0.07	178 ± 1.7

7.3 The Alternate Algorithm

The alternate algorithm was presented in a paper by Maranzana in 1964 [15] and begins with creating a random median set and assigning all nodes in the network to one of these median nodes. The group of free nodes associated with each median node is evaluated and the median for this group is determined and replaces the node in the median set if different from the current median. All nodes in the network are then reassigned to the closest median in the new median set and once again each group's true median is determined and replaces the current node in the median set. This process continues until there is an iteration with no changes to the median set.

7.3.1 Pseudocode

Algorithm 6 Alternate Algorithm

```
INPUT: Cost Matrix, $P^*$, and $S^*$

STEP 0: Initialisation
                    Set $k = 1$

STEP 1: Assign each node in the network to its closest median.
        $K_j = \{\text{all } i \in G \text{ such that } \min(d_{ij})\}$ where $j \in P^*$

STEP 2: Find the median node for each current median's set of associated nodes.
        Find $i \in K_j$ where $\min \left[ \sum_{i \in K_j} d_{ij} \right]$

STEP 3: Update $P^*$ with the new set of medians.
        $P^* = \{\text{all } k_j\}$ for $j \in P^*$

STEP 4: The algorithm must stop if either the updated $P^*$ is identical to the
new $P^*$ or there has been no improvement in th value of $S^*$ during the past three
iterations.

if $P^*(k) = P^*(k-1)$ OR $S^{k-2} = S^{k-1} = S^k = 0$ then
STOP [
else
                    $S^k = S^*$ and $k = k + 1$
go to STEP 1 ]
end if
```

7.3.2 Results

In table 8, the results obtained from repeating the alternate algorithm 50 times with a starting solution generated from the greedy algorithm are given and table 9 shows the results from starting the alternate algorithm with a random feasible solution.

Figure 16 gives the average computational time for all forty test problems for each of the starting solutions. For low values of p, both the random starting solution and the greedy starting solution are near identical while the greatest difference can be seen when the ratio of required medians to network size is high like in pmed25 ($n = 500$ and $p = 167$) and pmed30 ($n = 600$ and $p = 200$)

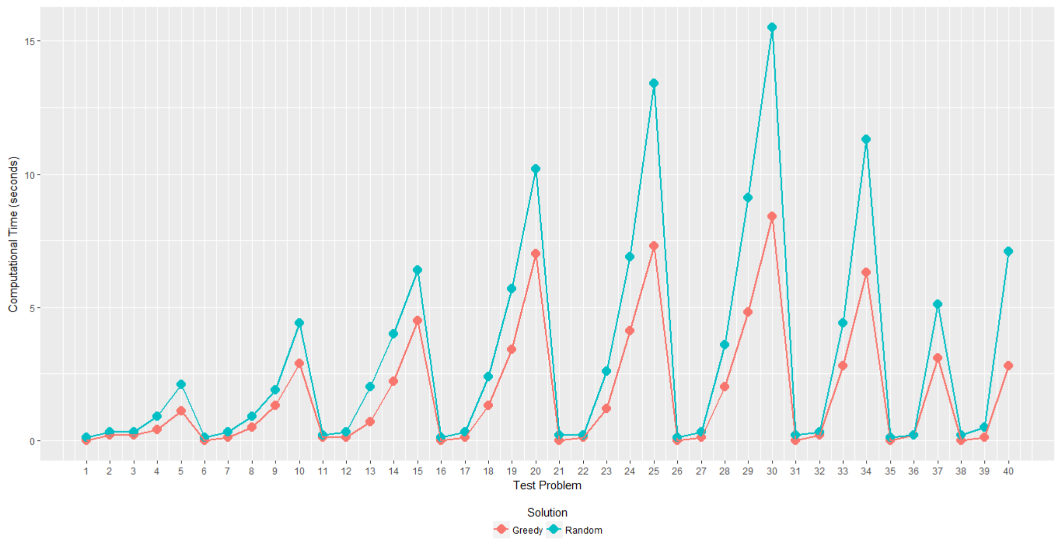

Figure 16: Average Computational Time of each Test Problem

Figure 17 shows that the greedy starting solution consistently results in a near-optimal final solution, while the random starting solution is far less consistent and only problems with a low value of p result in good final solutions.

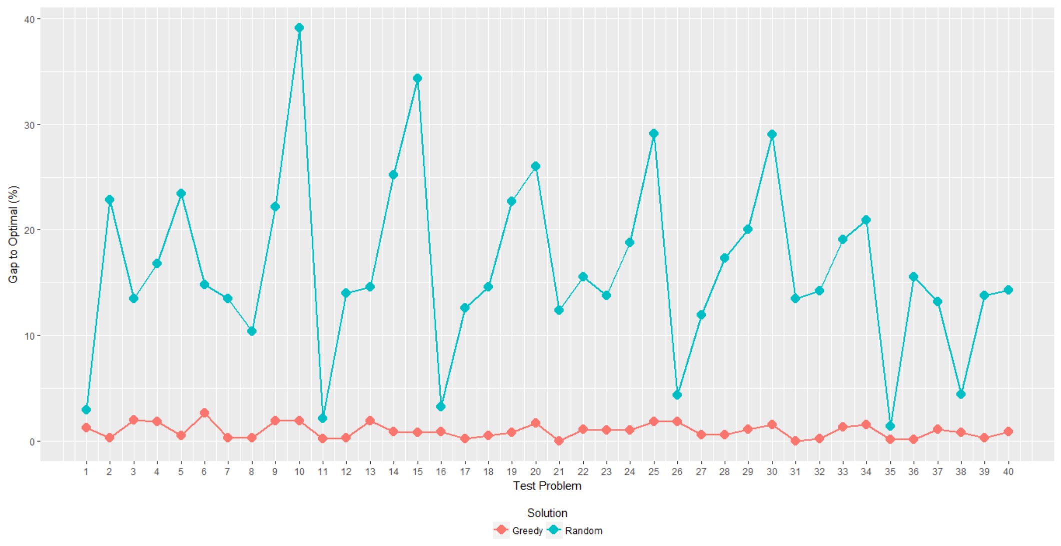

Figure 17: Average Gap to Optimal Value of each Test Problem

It can be seen in figure 18, that a greedy starting solution requires less iterations than the random starting solution for every test problem.

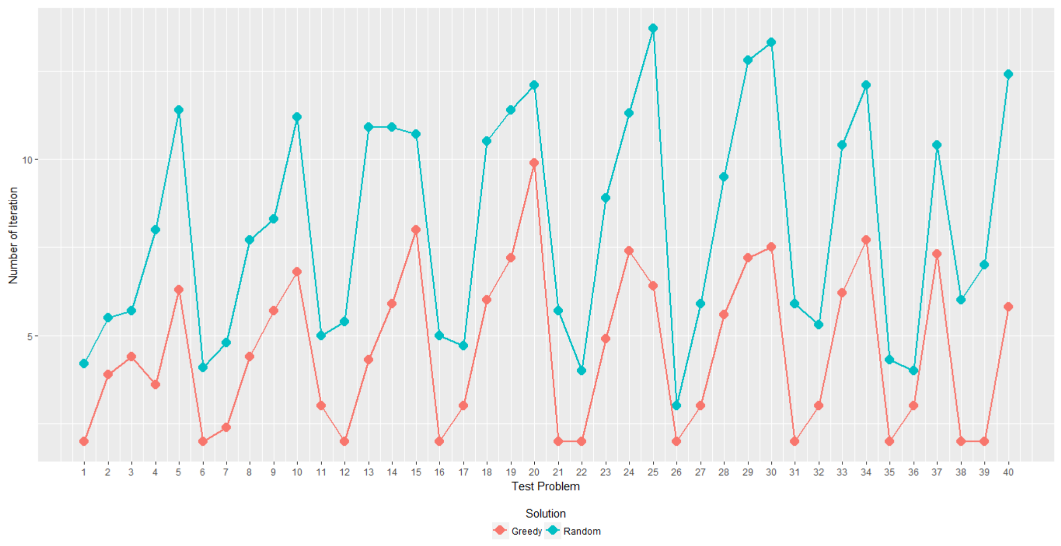

Figure 18: Average Number Iterations for each Test Problem

Table 8: Results of the Alternate Algorithm - Greedy (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (percent)**	**Num. of Iterations**
1	5	100	198	5819	0 ± 0.01	1.2 ± 0	2 ± 0
2	10	100	193	4093	0.2 ± 0.06	0.3 ± 0	3.9 ± 0.81
3	10	100	198	4250	0.2 ± 0.09	2 ± 0.03	4.4 ± 1.31
4	20	100	196	3034	0.4 ± 0.09	1.8 ± 0	3.7 ± 0.56
5	33	100	196	1355	1.1 ± 0.21	0.5 ± 0.04	6.1 ± 0.95
6	5	200	786	7824	0 ± 0.01	2.6 ± 0	2 ± 0
7	10	200	779	5631	0.1 ± 0.04	0.3 ± 0.01	2.4 ± 0.5
8	20	200	792	4445	0.5 ± 0.11	0.3 ± 0	4.3 ± 0.84
9	40	200	785	2734	1.3 ± 0.25	1.9 ± 0.07	5.8 ± 0.94
10	67	200	786	1255	2.9 ± 0.7	1.9 ± 0.3	7.5 ± 1.55
11	5	300	1772	7696	0.1 ± 0.01	0.2 ± 0	3 ± 0
12	10	300	1758	6634	0.1 ± 0.01	0.3 ± 0	2 ± 0
13	30	300	1760	4374	0.7 ± 0.17	1.9 ± 0	4.4 ± 0.78
14	60	300	1771	2968	2.2 ± 0.55	0.9 ± 0.02	6.4 ± 1.35
15	100	300	1754	1729	4.5 ± 1.12	0.8 ± 0.08	7.8 ± 1.71
16	5	400	3153	8162	0 ± 0.01	0.9 ± 0	2 ± 0
17	10	400	3142	6999	0.1 ± 0.01	0.2 ± 0	3 ± 0
18	40	400	3134	4809	1.3 ± 0.13	0.5 ± 0	5.9 ± 0.48
19	80	400	3134	2845	3.4 ± 0.67	0.8 ± 0.08	7.3 ± 1.25
20	133	400	3144	1789	7 ± 1.7	1.7 ± 0.35	8.9 ± 1.87
21	5	500	4909	9138	0 ± 0.01	0 ± 0	2 ± 0
22	10	500	4896	8579	0.1 ± 0.01	1.1 ± 0	2 ± 0
23	50	500	4903	4619	1.2 ± 0.33	1 ± 0.03	4.6 ± 0.99
24	100	500	4914	2961	4.1 ± 0.87	1 ± 0.04	7.1 ± 1.25
25	167	500	4894	1828	7.3 ± 1.45	1.8 ± 0.04	7.5 ± 1.31
26	5	600	7069	9917	0 ± 0.01	1.8 ± 0	2 ± 0
27	10	600	7072	8307	0.1 ± 0.01	0.6 ± 0	3 ± 0
28	60	600	7054	4498	2 ± 0.56	0.6 ± 0.03	5.9 ± 1.34
29	120	600	7042	3033	4.8 ± 1.13	1.1 ± 0.05	7 ± 1.4
30	200	600	7042	1989	8.4 ± 1.61	1.5 ± 0.06	7.3 ± 1.19
31	5	700	9601	10086	0 ± 0.01	0 ± 0	2 ± 0
32	10	700	9584	9297	0.2 ± 0.02	0.2 ± 0.02	3 ± 0
33	70	700	9616	4700	2.8 ± 0.65	1.3 ± 0.03	6.8 ± 1.33
34	140	700	9585	3013	6.3 ± 1.53	1.5 ± 0.05	7.8 ± 1.64
35	5	800	12548	10400	0 ± 0.01	0.1 ± 0	2 ± 0
36	10	800	12560	9934	0.2 ± 0.02	0.1 ± 0	3 ± 0
37	80	800	12564	5057	3.1 ± 0.65	1.1 ± 0.01	6.7 ± 1.16
38	5	900	15898	11060	0 ± 0.01	0.8 ± 0	2 ± 0
39	10	900	15896	9423	0.1 ± 0.01	0.3 ± 0	2 ± 0
40	90	900	15879	5128	2.8 ± 0.69	0.9 ± 0.02	5.6 ± 1.13

Table 9: Results of the Alternate Algorithm - Random (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Value**	**Computational Time (seconds)**	**Gap to Optimal (percent)**	**Num. of Iterations**
1	5	100	198	5819	0.1 ± 0.03	2.9 ± 0	4.3 ± 0.44
2	10	100	193	4093	0.3 ± 0.09	22.8 ± 1.78	5.2 ± 1.2
3	10	100	198	4250	0.3 ± 0.1	13.5 ± 3.08	5.8 ± 1.49
4	20	100	196	3034	0.9 ± 0.26	16.8 ± 3.57	7.4 ± 1.89
5	33	100	196	1355	2.1 ± 0.53	23.4 ± 5.87	10.7 ± 2.43
6	5	200	786	7824	0.1 ± 0.05	14.8 ± 1.66	4.6 ± 1.26
7	10	200	779	5631	0.3 ± 0.06	13.5 ± 2.42	4.9 ± 0.77
8	20	200	792	4445	0.9 ± 0.25	10.4 ± 0.98	7.6 ± 1.86
9	40	200	785	2734	1.9 ± 0.37	22.2 ± 1.37	8 ± 1.37
10	67	200	786	1255	4.4 ± 0.99	39.1 ± 1.24	10.8 ± 2.2
11	5	300	1772	7696	0.2 ± 0.02	2.1 ± 0	5 ± 0
12	10	300	1758	6634	0.3 ± 0.07	14 ± 0.73	5.6 ± 0.84
13	30	300	1760	4374	2 ± 0.74	14.6 ± 2.43	10.6 ± 3.69
14	60	300	1771	2968	4 ± 0.94	25.2 ± 1.3	11 ± 2.31
15	100	300	1754	1729	6.4 ± 1.12	34.3 ± 1.88	10.7 ± 1.64
16	5	400	3153	8162	0.1 ± 0.02	3.2 ± 0	5 ± 0
17	10	400	3142	6999	0.3 ± 0.05	12.6 ± 0.22	4.8 ± 0.77
18	40	400	3134	4809	2.4 ± 0.66	14.6 ± 3.06	10 ± 2.33
19	80	400	3134	2845	5.7 ± 1.04	22.7 ± 1.74	11.6 ± 1.88
20	133	400	3144	1789	10.2 ± 1.88	26 ± 1.45	12.5 ± 2.13
21	5	500	4909	9138	0.2 ± 0.04	12.4 ± 0	5.8 ± 0.93
22	10	500	4896	8579	0.2 ± 0.03	15.5 ± 0	4 ± 0
23	50	500	4903	4619	2.6 ± 0.56	13.8 ± 1.2	8.7 ± 1.62
24	100	500	4914	2961	6.9 ± 1.23	18.8 ± 0.66	11.4 ± 1.74
25	167	500	4894	1828	13.4 ± 2.42	29.1 ± 2.03	13.1 ± 2.15
26	5	600	7069	9917	0.1 ± 0.02	4.3 ± 0	3 ± 0
27	10	600	7072	8307	0.3 ± 0.08	11.9 ± 0.22	5.4 ± 1.03
28	60	600	7054	4498	3.6 ± 0.73	17.3 ± 0.79	9.8 ± 1.8
29	120	600	7042	3033	9.1 ± 1.52	20 ± 1.15	12.3 ± 1.86
30	200	600	7042	1989	15.5 ± 2.18	29 ± 1.49	12.6 ± 1.62
31	5	700	9601	10086	0.2 ± 0.05	13.5 ± 0.86	6 ± 1.37
32	10	700	9584	9297	0.3 ± 0.03	14.2 ± 0.04	5.2 ± 0.43
33	70	700	9616	4700	4.4 ± 0.97	19.1 ± 0.85	10.1 ± 1.91
34	140	700	9585	3013	11.3 ± 1.82	20.9 ± 0.88	13.1 ± 1.94
35	5	800	12548	10400	0.1 ± 0.03	1.4 ± 0	4.5 ± 0.5
36	10	800	12560	9934	0.2 ± 0.02	15.5 ± 0	4 ± 0
37	80	800	12564	5057	5.1 ± 1.46	13.2 ± 0.81	10.5 ± 2.76
38	5	900	15898	11060	0.2 ± 0.03	4.4 ± 0	6 ± 0
39	10	900	15896	9423	0.5 ± 0.03	13.8 ± 0.26	7 ± 0
40	90	900	15879	5128	7.1 ± 1.02	14.3 ± 0.93	12.7 ± 1.73

7.4 Comparison of Results of Local Search Heuristics

Figure 19 shows the average computational time for each of the three local search algorithms starting with a greedy solution. It is clear that the computational time is highest for each network size group when the value of p is the highest. Time to completion for the Teitz and Bart interchange algorithm along with the fast interchange algorithm increases as the network size increases while the time that the alternate algorithm takes to find a final solution is less dependant on the network size and more related to the percentage of medians within the network.

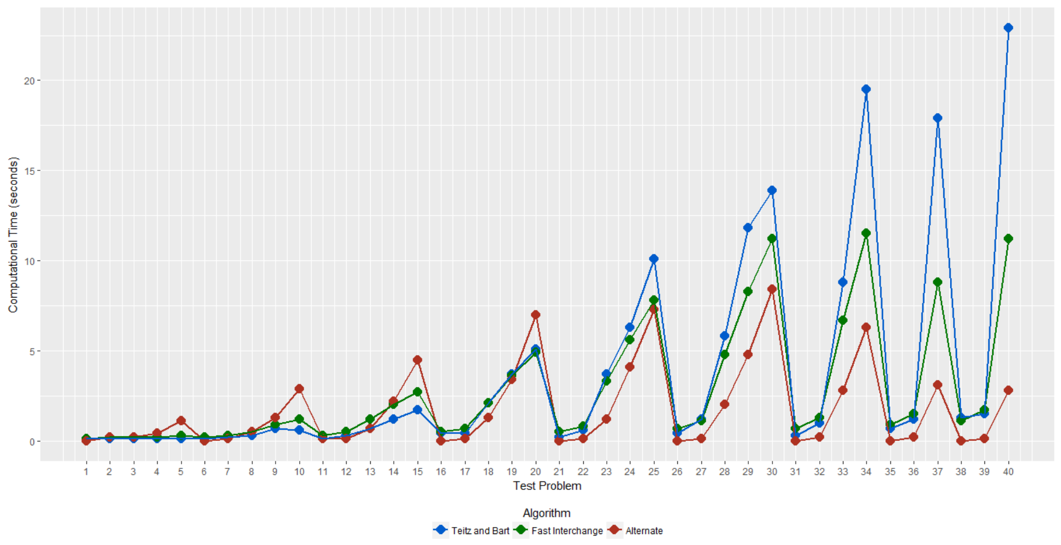

Figure 19: Average Computational Time of each Test Problem - Greedy

Staring a local search algorithm with a random solution does not result in a drastic difference in computational time. As in figure 19, figure 20 shows that all three algorithms follow a similar curve and only as the size of the network increase does the variability between th algorithms begin to show.

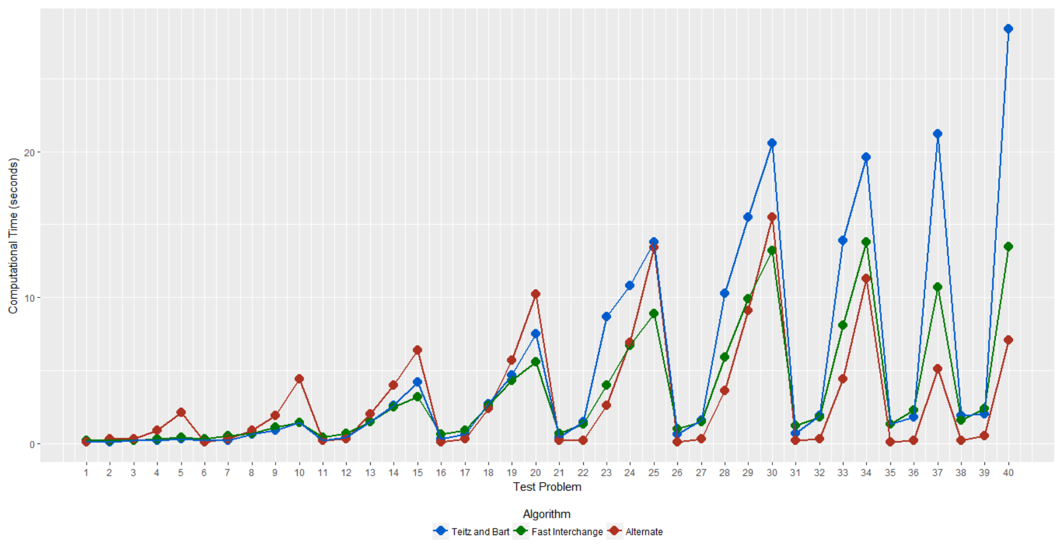

Figure 20: Average Computational Time of each Test Problem - Random

Figure 21 shows the percentage gap to the optimal value for a greedy starting solution and it is clear that the alternate algorithm produces far worse results than the Teitz and Bart interchange algorithm and the fast interchange algorithm. Although, all three algorithms produce results far superior to those produced by the fast greedy algorithm and the stingy algorithm.

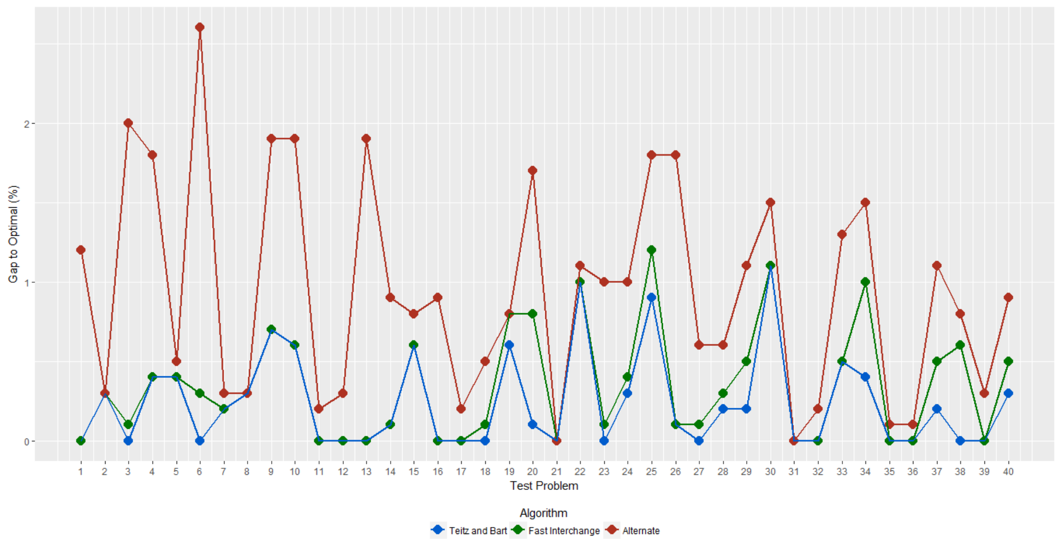

Figure 21: Average Gap to Optimal Value of each Test Problem - Greedy

Figure 22 displays the average percentage gap to optimal when the local search algorithms start with a random feasible solution. It is clear that the alternate algorithm requires a good starting solution in order to produce an adequate final solution while the Teitz and Bart interchange algorithm and the fast interchange algorithm do well without the greedy starting solution.

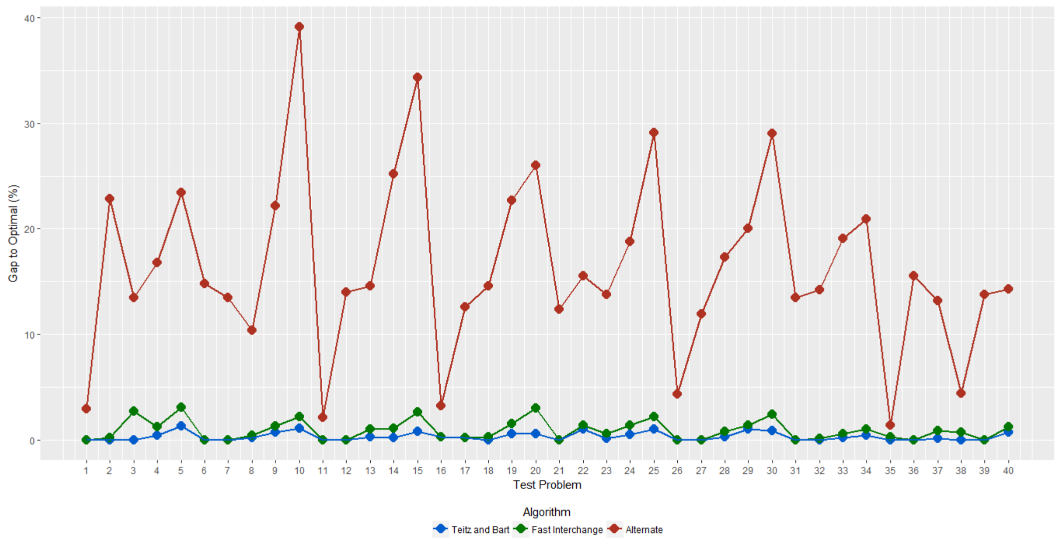

Figure 22: Average Gap to Optimal Value of each Test Problem - Random

8 Conclusion

When selecting an algorithm to help solve the p-median problem, it is clear that there are a few considerations that need to be taken into account.

If time is of the highest concern, then the fast greedy may be the best choice as it has been shown to produce feasible solutions in under 1 second.

If quality is the biggest concern, then the fast greedy will not be a good choice. Using a constructive heuristic to find a good starting solution to be improved upon with a local search algorithm would be a better answer.

There are many other classical heuristics that have not been covered in this study that may produce good results. The also exist metaheuristics that might provide even better solutions than the six covered in this study.

9 References

[1] S. L. Hakimi, “Optimum locations of switching centers and the absolute centers and medians of a graph,” *Operations Research*, vol. 12, no. 3, pp. 450–459, 1964.

[2] C. S. ReVelle and R. W. Swain, “Central facilities location,” *Geographical Analysis*, vol. 2, no. 1, pp. 30–42, 1970.

[3] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez, “The p-median problem: A survey of metaheuristic approaches,” *European Journal of Operational Research*, vol. 179, no. 3, pp. 927–939, 2007.

[4] F. Chiyoshi and R. D. Galvão, “A statistical analysis of simulated annealing applied to the p-median problem,” *Annals of Operations Research*, vol. 96, no. 1, pp. 61–74, 2000.

[5] O. Alp, E. Erkut, and Z. Drezner, “An efficient genetic algorithm for the p-median problem,” *Annals of Operations Research*, vol. 122, no. 1, pp. 21–42, 2003.

[6] P. Hansen and N. Mladenović, “Variable neighborhood search: Principles and applications,” *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

[7] E. Rolland, D. A. Schilling, and J. R. Current, “An efficient tabu search procedure for the p-median problem,” *European Journal of Operational Research*, vol. 96, no. 2, pp. 329–342, 1997.

[8] K. E. Rosing, C. S. ReVelle, E. Rolland, D. Schilling, and J. Current, “Heuristic concentration and tabu search: A head to head comparison,” *European Journal of Operational Research*, vol. 104, no. 1, pp. 93–99, 1998.

[9] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/

[10] J. E. Beasley, “OR-Library: distributing test problems by electronic mail,” *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[11] G. Csardi and T. Nepusz, “The igraph software package for complex network research,” *InterJournal*, vol. Complex Systems, p. 1695, 2006. [Online]. Available: http://igraph.org

[12] R. Whitaker, “A fast algorithm for the greedy interchange for large-scale clustering and median location problems,” *INFOR: Information Systems and Operational Research*, vol. 21, no. 2, pp. 95–108, 1983.

[13] E. Feldman, F. Lehrer, and T. Ray, “Warehouse location under continuous economies of scale,” *Management Science*, vol. 12, no. 9, pp. 670–684, 1966.

[14] M. B. Teitz and P. Bart, "Heuristic methods for estimating the generalized vertex median of a weighted graph," *Operations Research*, vol. 16, no. 5, pp. 955–961, 1968.

[15] F. Maranzana, "On the location of supply points to minimize transport costs," *OR*, pp. 261–270, 1964.

10 Index of Terms

11 Index of Authors

Appendices

A Constructive Heuristics Results

A.1 Greedy Algorithm

The following table provides the results obtained from 50 runs of the greedy algorithm. The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file.

The next four columns relate the to recorded computational time in seconds.

The final four columns relate to the difference between the objective function value of the final solutions and the known optimal solution's objective function.

Table 10: Greedy Algorithm (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (%)**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	0	0	0.1	0.01	1.2	1.2	1.2	0
2	10	100	193	4093	0	0	0.0	0.01	0.6	0.6	0.6	0
3	10	100	198	4250	0	0	0.1	0.01	3.5	3.5	3.5	0
4	20	100	196	3034	0	0	0.1	0.01	2	1.8	2.2	0.2
5	33	100	196	1355	0	0	0.1	0.01	2.6	1.7	4.4	0.69
6	5	200	786	7824	0	0	0.1	0.02	2.6	2.6	2.6	0
7	10	200	779	5631	0	0	0.1	0.02	0.3	0.3	0.3	0
8	20	200	792	4445	0	0	0.1	0.02	1.6	1.5	2.3	0.16
9	40	200	785	2734	0.1	0	0.1	0.02	4.9	4.1	5.2	0.48
10	67	200	786	1255	0.1	0.1	0.2	0.02	5.5	3.6	7.8	1.6
11	5	300	1772	7696	0	0	0.1	0.02	0.3	0.3	0.3	0
12	10	300	1758	6634	0.1	0	0.1	0.03	0.3	0.3	0.3	0
13	30	300	1760	4374	0.2	0.1	0.2	0.03	2.3	2.3	2.3	0
14	60	300	1771	2968	0.3	0.2	0.4	0.03	2.5	2.1	3.1	0.26
15	100	300	1754	1729	0.5	0.4	0.6	0.04	6.2	4.5	7.9	0.83
16	5	400	3153	8162	0.1	0	0.1	0.03	0.9	0.9	0.9	0
17	10	400	3142	6999	0.1	0.1	0.2	0.04	0.3	0.3	0.3	0
18	40	400	3134	4809	0.4	0.3	0.6	0.07	1.4	1.3	1.5	0.07
19	80	400	3134	2845	0.8	0.6	0.9	0.06	3.9	3.6	4.3	0.19
20	133	400	3144	1789	1.4	1.3	1.6	0.07	7.6	5.8	8.9	0.69

Table 10: Greedy Algorithm (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (%)			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
21	5	500	4909	9138	0.1	0	0.2	0.04	0	0	0	0
22	10	500	4896	8579	0.2	0.1	0.2	0.04	1.1	1.1	1.1	0
23	50	500	4903	4619	0.7	0.5	0.8	0.05	1.6	1.5	1.8	0.11
24	100	500	4914	2961	1.9	1.4	2.5	0.36	2.2	1.8	2.5	0.16
25	167	500	4894	1828	2.8	2.6	3.2	0.12	7.9	6.6	9.6	0.65
26	5	600	7069	9917	0.1	0.1	0.2	0.03	1.8	1.8	1.8	0
27	10	600	7072	8307	0.2	0.2	0.3	0.04	0.8	0.8	0.8	0
28	60	600	7054	4498	1.5	1.2	1.7	0.1	1.8	1.7	2.2	0.12
29	120	600	7042	3033	3	2.7	3.5	0.17	3.5	2.5	4.5	0.53
30	200	600	7042	1989	5.2	4.9	5.8	0.22	9.1	7.1	10.1	0.55
31	5	700	9601	10086	0.1	0.1	0.2	0.03	0	0	0	0
32	10	700	9584	9297	0.2	0.1	0.3	0.04	0.4	0.4	0.4	0
33	70	700	9616	4700	1.8	1.6	2.0	0.1	2.1	2	2.3	0.11
34	140	700	9585	3013	3.7	3.5	4.1	0.13	2.9	2	3.8	0.56
35	5	800	12548	10400	0.2	0.1	0.4	0.06	0.1	0.1	0.1	0
36	10	800	12560	9934	0.6	0.5	0.7	0.03	0.2	0.2	0.2	0
37	80	800	12564	5057	2.8	2.4	3.9	0.25	1.5	1.2	1.9	0.2
38	5	900	15898	11060	0.3	0.3	0.4	0.02	0.8	0.8	0.8	0
39	10	900	15896	9423	0.6	0.5	0.7	0.04	0.3	0.3	0.3	0
40	90	900	15879	5128	3.6	3.2	4.1	0.22	1.8	1.3	2	0.18

A.2 Fast Greedy Algorithm

The following table provides the results obtained from 50 runs of the fast greedy algorithm. The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file.The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file. The next four columns relate the to recorded computational time in seconds.

The final four columns relate to the difference between the objective function value of the final solutions and the known optimal solution's objective function.

Table 11: Fast Greedy Algorithm Results (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (%)**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	0	0	0.1	0.01	20.3	20.3	20.3	0
2	10	100	193	4093	0	0	0.1	0.01	66.3	66.3	66.3	0
3	10	100	198	4250	0	0	0.1	0.01	64.9	64.9	64.9	0
4	20	100	196	3034	0.1	0.1	0.1	0.01	178.3	178.3	178.3	0
5	33	100	196	1355	0.1	0.1	0.1	0.01	376.4	376.4	376.4	0
6	5	200	786	7824	0	0	0.1	0.01	10.1	10.1	10.1	0
7	10	200	779	5631	0	0	0.1	0.01	41.8	41.8	41.8	0
8	20	200	792	4445	0.1	0.1	0.1	0.01	108.4	108.4	108.4	0
9	40	200	785	2734	0.1	0.1	0.2	0.01	221.2	221.2	221.2	0
10	67	200	786	1255	0.2	0.2	0.3	0.02	457.9	457.9	457.9	0
11	5	300	1772	7696	0	0	0.1	0.01	11.1	11.1	11.1	0
12	10	300	1758	6634	0.1	0	0.1	0.02	36.2	36.2	36.2	0
13	30	300	1760	4374	0.1	0.1	0.2	0.02	110.1	110.1	110.1	0
14	60	300	1771	2968	0.2	0.2	0.3	0.02	262.5	262.5	262.5	0
15	100	300	1754	1729	0.4	0.3	0.5	0.03	417.7	417.4	418.2	0.4
16	5	400	3153	8162	0.1	0	0.1	0.03	6.8	6.8	6.8	0
17	10	400	3142	6999	0.1	0	0.1	0.03	37.1	37.1	37.1	0
18	40	400	3134	4809	0.2	0.1	0.2	0.03	122.7	122.7	122.7	0
19	80	400	3134	2845	0.3	0.3	0.4	0.03	262.5	262.5	262.5	0
20	133	400	3144	1789	0.5	0.5	0.7	0.05	547	547	547	0

Table 11: Fast Greedy Algorithm Results (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (%)**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
21	5	500	4909	9138	0.1	0	0.2	0.04	7.6	7.6	7.6	0
22	10	500	4896	8579	0.1	0.1	0.2	0.02	24.9	24.8	24.9	0.06
23	50	500	4903	4619	0.3	0.2	0.3	0.03	142.3	142.3	142.3	0
24	100	500	4914	2961	0.5	0.4	0.8	0.08	286.1	285.9	286.3	0.2
25	167	500	4894	1828	0.6	0.6	0.8	0.04	499	499	499	0
26	5	600	7069	9917	0.1	0.1	0.2	0.04	5.4	5.4	5.4	0
27	10	600	7072	8307	0.1	0.1	0.2	0.02	24.1	24.1	24.1	0
28	60	600	7054	4498	0.3	0.3	0.4	0.04	132.7	132.2	133.3	0.55
29	120	600	7042	3033	0.5	0.5	0.6	0.04	279.7	279.7	279.7	0
30	200	600	7042	1989	0.8	0.7	0.9	0.04	542.6	542.6	542.6	0
31	5	700	9601	10086	0.2	0.1	0.2	0.03	7.4	7.4	7.4	0
32	10	700	9584	9297	0.2	0.1	0.2	0.03	28.5	28.5	28.5	0
33	70	700	9616	4700	0.4	0.3	0.5	0.04	167.8	167.8	167.8	0
34	140	700	9585	3013	0.6	0.6	0.9	0.05	302.9	302.9	302.9	0
35	5	800	12548	10400	0.2	0.1	0.3	0.04	4.3	4.3	4.3	0
36	10	800	12560	9934	0.2	0.2	0.3	0.04	26.4	26.4	26.4	0
37	80	800	12564	5057	0.5	0.4	0.5	0.03	172.2	172.2	172.2	0.02
38	5	900	15898	11060	0.2	0.2	0.3	0.03	5.9	5.9	5.9	0
39	10	900	15896	9423	0.2	0.2	0.3	0.03	23.3	23.3	23.3	0
40	90	900	15879	5128	0.6	0.5	0.7	0.04	167.6	167.6	167.6	0

A.3 Stingy Algorithm

The following table provides the results obtained from 50 runs of the stingy algorithm. The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file.

The next four columns relate the to recorded computational time in seconds.

The final four columns relate to the difference between the objective function value of the final solutions and the known optimal solution's objective function.

Table 12: Stingy Algorithm Results (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (%)**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	3.3	3.2	4.2	0.16	116.1	116.1	116.1	0
2	10	100	193	4093	3.3	3.2	3.6	0.11	131.9	126.3	138.1	3.4
3	10	100	198	4250	3.3	3.2	3.6	0.11	130.6	130.4	131	0.29
4	20	100	196	3034	3.1	3	3.7	0.12	108.5	106.7	110.2	1.78
5	33	100	196	1355	2.9	2.8	3.4	0.15	145	138.5	155.6	8.07
6	5	200	786	7824	28.2	27.8	30.2	0.47	99.2	99.2	99.2	0
7	10	200	779	5631	28.1	27.6	28.7	0.36	125.3	124.5	126.1	0.81
8	20	200	792	4445	27.8	27.4	28.4	0.34	131.3	128.7	133	2.1
9	40	200	785	2734	26.9	26.5	27.5	0.37	135.3	132.2	137.1	2.34
10	67	200	786	1255	25	24.4	28.5	0.77	173.2	173.2	173.2	0
11	5	300	1772	7696	131.8	131.5	132.3	0.38	154.2	148.2	172.4	12.12
12	10	300	1758	6634	131.4	131.1	131.6	0.23	163	163	163	0
13	30	300	1760	4374	130.1	129.7	130.7	0.44	149.6	149.6	149.6	0
14	60	300	1771	2968	125.5	125.2	125.9	0.26	138.7	137	140.6	1.97
15	100	300	1754	1729	116.4	115.8	117.4	0.38	125.8	123.7	127.3	1.03
16	5	400	3153	8162	406.3	406.1	406.5	0.17	149.7	149.7	149.7	0
17	10	400	3142	6999	406.8	405.4	408.1	1.41	136.8	135.4	141	2.79
18	40	400	3134	4809	402.4	400.1	408.2	3.86	135.7	134.2	136.9	1.26
19	80	400	3134	2845	389.6	388.9	390.5	0.69	124.3	124	124.7	0.33
20	133	400	3144	1789	359	358.8	359.2	0.19	128.2	128.2	128.2	0

B Local Search Heuristics Results

B.1 Interchange Algorithm of Teitz and Bart

The following table provides the results obtained from 50 runs of the interchange algorithm of Teitz and Bart. The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file.

The next four columns relate the to recorded computational time in seconds.

The following four columns relate to the difference between the objective function value of the final solutions and the known optimal solution's objective function.

The final four columns provide information related to the total number of iterations preformed during each run of the test problems.

The first table gives the results from using a greedy starting solution while the second table provides the results from using a random feasible solution.

Table 13: Interchange Algorithm of Teitz and Bart - Greedy (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (percent)**				**Iterations**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	0.1	0	0.1	0.02	0	0	0	0	6	6	6	0
2	10	100	193	4093	0.1	0	0.1	0.01	0.3	0.3	0.3	0	10	10	10	0
3	10	100	198	4250	0.1	0.1	0.2	0.02	0	0	0	0	3	3	3	0
4	20	100	196	3034	0.1	0.1	0.2	0.02	0.4	0.4	0.4	0	4	4	4	0
5	33	100	196	1355	0.1	0.1	0.2	0.02	0.4	0.4	0.4	0	10	10	10	0
6	5	200	786	7824	0.1	0.1	0.2	0.02	0	0	0	0	16	16	16	0
7	10	200	779	5631	0.2	0.1	0.2	0.02	0.2	0.2	0.2	0	12	12	12	0
8	20	200	792	4445	0.3	0.2	0.3	0.02	0.3	0.3	0.3	0	6	6	6	0
9	40	200	785	2734	0.7	0.7	0.9	0.04	0.7	0.7	0.7	0	3	3	3	0
10	67	200	786	1255	0.6	0.6	0.7	0.02	0.6	0.6	0.6	0	12	12	12	0
11	5	300	1772	7696	0.1	0.1	0.2	0.02	0	0	0	0	21	21	21	0
12	10	300	1758	6634	0.3	0.2	0.4	0.03	0	0	0	0	3	3	3	0
13	30	300	1760	4374	0.7	0.7	0.9	0.04	0	0	0	0	30	30	30	0
14	60	300	1771	2968	1.2	1.2	1.5	0.07	0.1	0.1	0.1	0	1	1	1	0
15	100	300	1754	1729	1.7	1.6	2	0.07	0.6	0.6	0.6	0	2	2	2	0
16	5	400	3153	8162	0.4	0.3	0.5	0.03	0	0	0	0	18	18	18	0
17	10	400	3142	6999	0.4	0.4	0.5	0.02	0	0	0	0	16.8	16	18	1.03
18	40	400	3134	4809	2.1	2	2.4	0.08	0	0	0	0	28.9	26	30	1.66
19	80	400	3134	2845	3.7	3.6	4.1	0.12	0.6	0.6	0.6	0	7	7	7	0
20	133	400	3144	1789	5.1	5	5.9	0.19	0.1	0.1	0.1	0	8	8	8	0

Table 13: Interchange Algorithm of Teitz and Bart - Greedy (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
21	5	500	4909	9138	0.2	0.1	0.2	0.02	0	0	0	0	17.8	17	18	0.42
22	10	500	4896	8579	0.6	0.5	0.7	0.04	1	1	1	0	27.9	27	28	0.32
23	50	500	4903	4619	3.7	3.6	4.1	0.1	0	0	0	0	7	7	7	0
24	100	500	4914	2961	6.3	6.2	6.9	0.19	0.3	0.3	0.3	0	17	17	17	0
25	167	500	4894	1828	10.1	8.6	12.3	1.43	0.9	0.7	1	0.07	1	1	1	0
26	5	600	7069	9917	0.4	0.4	0.6	0.03	0.1	0.1	0.1	0	6	6	6	0
27	10	600	7072	8307	1.2	1.1	1.4	0.06	0	0	0	0	23	23	23	0
28	60	600	7054	4498	5.8	5.7	6.5	0.2	0.2	0.2	0.2	0	37.7	37	38	0.48
29	120	600	7042	3033	11.8	10	14	1.65	0.2	0.2	0.2	0	2	2	2	0
30	200	600	7042	1989	13.9	13.7	15.6	0.44	1.1	1.1	1.1	0	4	4	4	0
31	5	700	9601	10086	0.3	0.2	0.3	0.02	0	0	0	0	29.3	28	30	0.95
32	10	700	9584	9297	1	1	1.2	0.05	0	0	0	0	9	9	9	0
33	70	700	9616	4700	8.8	8.6	9.4	0.21	0.5	0.5	0.5	0	2	2	2	0
34	140	700	9585	3013	19.5	15.2	20.9	1.88	0.4	0.4	0.5	0.01	3	3	3	0
35	5	800	12548	10400	0.7	0.6	0.7	0.03	0	0	0	0	24	24	24	0
36	10	800	12560	9934	1.2	1.2	1.4	0.05	0	0	0	0	3	3	3	0
37	80	800	12564	5057	17.9	12.4	21.2	3.65	0.2	0.2	0.3	0.01	8	8	8	0
38	5	900	15898	11060	1.3	1.2	1.5	0.06	0	0	0	0	2	2	2	0
39	10	900	15896	9423	1.5	1.5	1.7	0.05	0	0	0	0	4	4	4	0
40	90	900	15879	5128	22.9	22.6	23.6	0.3	0.3	0.3	0.3	0	12	12	12	0

Table 14: Interchange Algorithm of Teitz and Bart - Random (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (percent)**				**Iterations**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	0.1	0.1	0.1	0.01	0	0	0	0	17	17	17	0
2	10	100	193	4093	0.1	0.1	0.2	0.02	0	0	0	0	65.5	65	66	0.53
3	10	100	198	4250	0.2	0.1	0.2	0.01	0	0	0	0	21	21	21	0
4	20	100	196	3034	0.2	0.2	0.3	0.02	0.4	0.4	0.4	0	33	33	33	0
5	33	100	196	1355	0.3	0.2	0.3	0.02	1.3	1.3	1.3	0	63.6	63	64	0.52
6	5	200	786	7824	0.2	0.1	0.2	0.03	0	0	0	0	101.8	100	103	1.23
7	10	200	779	5631	0.2	0.2	0.3	0.02	0	0	0	0	111.5	106	116	3.14
8	20	200	792	4445	0.6	0.5	0.6	0.03	0.2	0.2	0.2	0	20	20	20	0
9	40	200	785	2734	0.9	0.9	1.1	0.06	0.7	0.7	0.7	0.02	33	33	33	0
10	67	200	786	1255	1.4	1.2	1.8	0.19	1.1	0.8	1.2	0.16	81.8	81	82	0.42
11	5	300	1772	7696	0.2	0.2	0.3	0.02	0	0	0	0	122.2	119	128	3.19
12	10	300	1758	6634	0.4	0.4	0.5	0.02	0	0	0	0	31	31	31	0
13	30	300	1760	4374	1.5	1.4	1.7	0.08	0.3	0.3	0.3	0	135.1	133	137	1.45
14	60	300	1771	2968	2.6	2.4	3.4	0.27	0.2	0.1	0.2	0.05	17	17	17	0
15	100	300	1754	1729	4.2	3.3	5.7	0.78	0.8	0.5	1.2	0.17	49	49	49	0
16	5	400	3153	8162	0.3	0.3	0.4	0.02	0.3	0.3	0.3	0	112.1	111	114	1.45
17	10	400	3142	6999	0.6	0.5	0.6	0.02	0.2	0.2	0.2	0	151.4	149	154	1.65
18	40	400	3134	4809	2.7	2.6	3.1	0.1	0	0	0	0	154.6	152	158	1.84
19	80	400	3134	2845	4.7	4.6	6	0.25	0.6	0.4	0.6	0.09	28	28	28	0
20	133	400	3144	1789	7.5	6.3	8.9	0.83	0.6	0.4	1	0.13	43	43	43	0

Table 14: Interchange Algorithm of Teitz and Bart - Random (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
21	5	500	4909	9138	0.4	0.4	0.5	0.02	0	0	0	0	139.6	136	140	1.26
22	10	500	4896	8579	1.5	1.5	1.8	0.06	1	1	1	0	190.9	189	194	1.6
23	50	500	4903	4619	8.7	8.5	9.3	0.21	0.1	0.1	0.1	0	31.2	30	33	1.55
24	100	500	4914	2961	10.8	10.4	12.8	0.65	0.5	0.3	0.7	0.13	195.2	193	198	1.81
25	167	500	4894	1828	13.8	11	17.3	1.62	1	0.6	1.4	0.2	25	25	25	0
26	5	600	7069	9917	0.6	0.6	0.7	0.03	0	0	0	0	47	47	47	0
27	10	600	7072	8307	1.6	1.5	1.8	0.05	0	0	0	0	166.7	163	171	2.26
28	60	600	7054	4498	10.3	9.8	12.3	0.8	0.3	0.2	0.4	0.08	217	212	224	3.65
29	120	600	7042	3033	15.5	13.4	18	1.9	1	0.9	1.1	0.07	25	25	25	0
30	200	600	7042	1989	20.6	18.1	28.4	2.98	0.9	0.5	1.3	0.2	52	52	52	0
31	5	700	9601	10086	0.7	0.7	0.8	0.03	0	0	0	0	187.5	182	193	2.95
32	10	700	9584	9297	1.9	1.8	2.3	0.09	0	0	0	0	30	30	30	0
33	70	700	9616	4700	13.9	10.9	16.8	0.74	0.2	0.1	0.5	0.08	46	46	46	0
34	140	700	9585	3013	19.6	18.8	24.8	1.63	0.4	0.2	0.5	0.07	32	32	32	0
35	5	800	12548	10400	1.3	1.2	1.5	0.06	0	0	0	0	189.5	177	200	9.01
36	10	800	12560	9934	1.8	1.7	2.1	0.08	0	0	0	0	40	40	40	0
37	80	800	12564	5057	21.2	15.6	25.1	3.31	0.1	0	0.2	0.04	24	24	24	0
38	5	900	15898	11060	1.9	1.8	2.2	0.06	0	0	0	0	32	32	32	0
39	10	900	15896	9423	2	1.9	2.3	0.09	0	0	0	0	50	50	50	0
40	90	900	15879	5128	28.4	19.7	43.1	7.87	0.7	0.3	1.1	0.24	56.6	56	57	0.52

B.2 Fast Interchange Algorithm

The following table provides the results obtained from 50 runs of the fast interchange algorithm. The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file.

The next four columns relate the to recorded computational time in seconds.

The following four columns relate to the difference between the objective function value of the final solutions and the known optimal solution's objective function.

The final four columns provide information related to the total number of iterations preformed during each run of the test problems.

The first table gives the results from using a greedy starting solution while the second table provides the results from using a random feasible solution.

Table 15: Fast Interchange Algorithm - Greedy (50 Runs)

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (percent)**				**Iterations**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	0.1	0.1	0.2	0.02	0	0	0	0	5	5	5	0
2	10	100	193	4093	0.2	0.1	0.2	0.02	0.3	0.3	0.3	0	3	3	3	0
3	10	100	198	4250	0.2	0.1	0.2	0.02	0.1	0.1	0.1	0	5	5	5	0
4	20	100	196	3034	0.2	0.2	0.3	0.02	0.4	0.4	0.4	0	3	3	3	0
5	33	100	196	1355	0.3	0.3	0.4	0.03	0.4	0.4	0.4	0	3	3	3	0
6	5	200	786	7824	0.2	0.2	0.3	0.02	0.3	0.3	0.3	0	7	7	7	0
7	10	200	779	5631	0.3	0.3	0.4	0.02	0.2	0.2	0.2	0	2	2	2	0
8	20	200	792	4445	0.5	0.5	0.7	0.03	0.3	0.3	0.3	0	4	4	4	0
9	40	200	785	2734	0.9	0.9	1.2	0.05	0.7	0.7	0.7	0	11	11	11	0
10	67	200	786	1255	1.2	1.2	1.4	0.05	0.6	0.6	0.6	0	10	10	10	0
11	5	300	1772	7696	0.3	0.3	0.4	0.02	0	0	0	0	3	3	3	0
12	10	300	1758	6634	0.5	0.5	0.6	0.04	0	0	0	0	4	4	4	0
13	30	300	1760	4374	1.2	1.2	1.5	0.06	0	0	0	0	10	10	10	0
14	60	300	1771	2968	2	2	2.4	0.08	0.1	0.1	0.1	0	16	16	16	0
15	100	300	1754	1729	2.7	2.6	3.1	0.11	0.6	0.6	0.6	0	12	12	12	0
16	5	400	3153	8162	0.5	0.4	0.6	0.03	0	0	0	0	5	5	5	0
17	10	400	3142	6999	0.7	0.6	0.7	0.02	0	0	0	0	3	3	3	0
18	40	400	3134	4809	2.1	2	2.4	0.09	0.1	0.1	0.1	0	11	11	11	0
19	80	400	3134	2845	3.6	3.5	4.1	0.11	0.8	0.8	0.8	0	16	16	16	0
20	133	400	3144	1789	4.9	4.8	5.6	0.15	0.8	0.8	0.8	0	24	24	24	0

Table 15: Fast Interchange Algorithm - Greedy (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
21	5	500	4909	9138	0.5	0.5	0.7	0.04	0	0	0	0	1	1	1	0
22	10	500	4896	8579	0.8	0.8	0.9	0.02	1	1	1	0	2	2	2	0
23	50	500	4903	4619	3.3	3.2	3.9	0.14	0.1	0.1	0.1	0	16	16	16	0
24	100	500	4914	2961	5.6	5.5	6.3	0.2	0.4	0.4	0.5	0.03	14	14	14	0
25	167	500	4894	1828	7.8	7.6	8.4	0.22	1.2	1	1.3	0.08	23.8	23	25	0.48
26	5	600	7069	9917	0.7	0.7	0.9	0.04	0.1	0.1	0.1	0	7	7	7	0
27	10	600	7072	8307	1.1	1	1.3	0.07	0.1	0.1	0.1	0	6	6	6	0
28	60	600	7054	4498	4.8	4.6	5.6	0.22	0.3	0.3	0.3	0	16.5	16	17	0.51
29	120	600	7042	3033	8.3	8.1	9	0.22	0.5	0.5	0.5	0	21.9	21	22	0.35
30	200	600	7042	1989	11.2	11	11.9	0.25	1.1	1.1	1.1	0	16	16	16	0
31	5	700	9601	10086	0.7	0.7	0.9	0.04	0	0	0	0	1	1	1	0
32	10	700	9584	9297	1.3	1.2	1.5	0.06	0	0	0	0	6	6	6	0
33	70	700	9616	4700	6.7	6.5	7.2	0.18	0.5	0.5	0.5	0	22	22	22	0
34	140	700	9585	3013	11.5	11.3	12.2	0.29	1	0.9	1	0.03	28.3	28	29	0.47
35	5	800	12548	10400	0.9	0.8	0.9	0.03	0	0	0	0	2	2	2	0
36	10	800	12560	9934	1.5	1.4	1.7	0.08	0	0	0	0	4	4	4	0
37	80	800	12564	5057	8.8	8.6	9.6	0.24	0.5	0.4	0.6	0.07	20.8	19	22	1.33
38	5	900	15898	11060	1.1	1	1.2	0.05	0.6	0.6	0.6	0	4	4	4	0
39	10	900	15896	9423	1.7	1.6	1.9	0.07	0	0	0	0	2	2	2	0
40	90	900	15879	5128	11.2	11	11.7	0.21	0.5	0.4	0.5	0.02	18.5	18	19	0.5

Table 16: Fast Interchange Algorithm - Random (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
1	5	100	198	5819	0.2	0.1	0.2	0.02	0	0	0	0	16	16	16	0
2	10	100	193	4093	0.2	0.2	0.3	0.02	0.2	0.2	0.2	0	30	30	30	0
3	10	100	198	4250	0.2	0.2	0.3	0.02	2.7	2.3	3.1	0.38	24.9	24	26	1
4	20	100	196	3034	0.3	0.3	0.4	0.02	1.2	1.2	1.2	0	31	31	31	0
5	33	100	196	1355	0.4	0.4	0.5	0.02	3.1	3.1	3.1	0	37	37	37	0
6	5	200	786	7824	0.3	0.3	0.4	0.02	0	0	0	0	24	24	24	0
7	10	200	779	5631	0.5	0.4	0.6	0.03	0	0	0	0	32	32	32	0
8	20	200	792	4445	0.7	0.7	0.8	0.03	0.4	0.4	0.4	0	47	47	47	0
9	40	200	785	2734	1.1	1	1.3	0.06	1.3	1.2	1.3	0.02	54.5	54	55	0.5
10	67	200	786	1255	1.4	1.4	1.7	0.06	2.2	2.2	2.2	0	61	61	61	0
11	5	300	1772	7696	0.4	0.4	0.6	0.04	0	0	0	0	21	21	21	0
12	10	300	1758	6634	0.7	0.7	0.7	0.02	0	0	0	0	33	33	33	0
13	30	300	1760	4374	1.5	1.4	1.7	0.06	1	0.9	1.1	0.07	61.2	60	62	0.98
14	60	300	1771	2968	2.5	2.4	2.9	0.12	1.1	1.1	1.2	0.04	93.4	90	95	1.3
15	100	300	1754	1729	3.2	3.1	3.6	0.13	2.6	2	3.3	0.33	95.8	94	98	1.02
16	5	400	3153	8162	0.6	0.6	0.6	0.02	0.3	0.3	0.3	0	20	20	20	0
17	10	400	3142	6999	0.9	0.9	1.1	0.04	0.2	0.2	0.2	0	33	33	33	0
18	40	400	3134	4809	2.6	2.5	3.3	0.16	0.3	0.2	0.3	0.01	78.3	78	79	0.46
19	80	400	3134	2845	4.3	4.2	4.8	0.13	1.5	1.3	1.7	0.12	112.7	110	116	1.5
20	133	400	3144	1789	5.6	5.5	6.3	0.17	3	2.8	3.4	0.16	119.2	117	122	1.13

Table 16: Fast Interchange Algorithm - Random (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
21	5	500	4909	9138	0.7	0.7	0.9	0.04	0	0	0	0	17	17	17	0
22	10	500	4896	8579	1.3	1.2	1.5	0.06	1.4	1.4	1.4	0	45	45	45	0
23	50	500	4903	4619	4	3.9	4.6	0.14	0.6	0.5	0.7	0.1	101.4	100	102	0.93
24	100	500	4914	2961	6.7	6.6	7.5	0.2	1.4	1.3	1.5	0.07	141.2	138	144	1.84
25	167	500	4894	1828	8.9	8.8	9.6	0.25	2.2	1.8	2.8	0.24	142.3	139	144	1.24
26	5	600	7069	9917	1	1	1.2	0.05	0	0	0	0	28	28	28	0
27	10	600	7072	8307	1.5	1.4	1.8	0.08	0	0	0	0	42	42	42	0
28	60	600	7054	4498	5.9	5.8	6.6	0.2	0.8	0.7	0.9	0.07	131.3	129	133	1.25
29	120	600	7042	3033	9.9	9.8	10.7	0.24	1.4	1.2	1.6	0.1	183	180	187	1.95
30	200	600	7042	1989	13.2	13	13.8	0.22	2.4	2	2.9	0.22	179.1	176	184	2
31	5	700	9601	10086	1.2	1.1	1.3	0.05	0	0	0	0	25	25	25	0
32	10	700	9584	9297	1.8	1.8	2.1	0.07	0.1	0.1	0.1	0	46	46	46	0
33	70	700	9616	4700	8.1	8	8.7	0.15	0.6	0.5	1	0.12	156.4	155	161	0.99
34	140	700	9585	3013	13.8	13.5	14.4	0.25	1	0.7	1.4	0.17	204.7	200	211	2.8
35	5	800	12548	10400	1.3	1.3	1.5	0.05	0.3	0.3	0.3	0	24	24	24	0
36	10	800	12560	9934	2.3	2.2	2.7	0.1	0	0	0	0	52	52	52	0
37	80	800	12564	5057	10.7	10.5	11.4	0.25	0.9	0.7	1	0.1	171.4	169	178	2.69
38	5	900	15898	11060	1.6	1.5	1.9	0.06	0.7	0.7	0.7	0	26	26	26	0
39	10	900	15896	9423	2.4	2.3	2.7	0.1	0	0	0	0	46	46	46	0
40	90	900	15879	5128	13.5	13.2	14	0.25	1.2	1	1.3	0.07	175	173	178	1.7

B.3 Alternate Algorithm

ast alternate algorithm

The following table provides the results obtained from 50 runs of the alternate algorithm. The first five columns provide information related to each of the forty test problems obtained from Beasley's library [10] and was provided in the corresponding text file.

The next four columns relate the to recorded computational time in seconds.

The following four columns relate to the difference between the objective function value of the final solutions and the known optimal solution's objective function.

The final four columns provide information related to the total number of iterations preformed during each run of the test problems.

The first table gives the results from using a greedy starting solution while the second table provides the results from using a random feasible solution.

ast alternate algorithm

Table 17: **Alternate Algorithm - Greedy (50 Runs)**

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (percent)**				**Iterations**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
1	5	100	198	5819	0	0	0.1	0.01	1.2	1.2	1.2	0	2	2	2	0
2	10	100	193	4093	0.2	0.1	0.3	0.06	0.3	0.3	0.3	0	3.9	3	5	0.81
3	10	100	198	4250	0.2	0.1	0.4	0.09	2	2	2	0.03	4.4	3	7	1.31
4	20	100	196	3034	0.4	0.1	0.5	0.09	1.8	1.8	1.8	0	3.7	2	4	0.56
5	33	100	196	1355	1.1	0.8	1.6	0.21	0.5	0.4	0.6	0.04	6.1	5	8	0.95
6	5	200	786	7824	0	0	0.1	0.01	2.6	2.6	2.6	0	2	2	2	0
7	10	200	779	5631	0.1	0	0.2	0.04	0.3	0.2	0.3	0.01	2.4	2	3	0.5
8	20	200	792	4445	0.5	0.2	0.6	0.11	0.3	0.3	0.3	0	4.3	3	5	0.84
9	40	200	785	2734	1.3	1	1.9	0.25	1.9	1.9	2.4	0.07	5.8	5	8	0.94
10	67	200	786	1255	2.9	1.8	4.4	0.7	1.9	1.8	3	0.3	7.5	5	11	1.55
11	5	300	1772	7696	0.1	0.1	0.1	0.01	0.2	0.2	0.2	0	3	3	3	0
12	10	300	1758	6634	0.1	0.1	0.1	0.01	0.3	0.3	0.3	0	2	2	2	0
13	30	300	1760	4374	0.7	0.4	1	0.17	1.9	1.9	1.9	0	4.4	3	5	0.78
14	60	300	1771	2968	2.2	1.6	3.7	0.55	0.9	0.9	1	0.02	6.4	5	10	1.35
15	100	300	1754	1729	4.5	2.6	7.2	1.12	0.8	0.8	1.3	0.08	7.8	5	12	1.71
16	5	400	3153	8162	0	0	0.1	0.01	0.9	0.9	0.9	0	2	2	2	0
17	10	400	3142	6999	0.1	0.1	0.2	0.01	0.2	0.2	0.2	0	3	3	3	0
18	40	400	3134	4809	1.3	0.8	1.6	0.13	0.5	0.5	0.5	0	5.9	4	6	0.48
19	80	400	3134	2845	3.4	2.1	4.8	0.67	0.8	0.8	1.2	0.08	7.3	5	10	1.25
20	133	400	3144	1789	7	3.5	11.3	1.7	1.7	1.1	2.3	0.35	8.9	5	13	1.87

Table 17: **Alternate Algorithm - Greedy (50 Runs)**

Test Problem	**p**	**n**	**Edges**	**Optimal Solution**	**Computational Time (seconds)**				**Gap to Optimal (percent)**				**Iterations**			
					Mean	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**	**Mean**	**Min**	**Max**	**Std Dev**
21	5	500	4909	9138	0	0	0.1	0.01	0	0	0	0	2	2	2	0
22	10	500	4896	8579	0.1	0.1	0.1	0.01	1.1	1.1	1.1	0	2	2	2	0
23	50	500	4903	4619	1.2	0.7	2.4	0.33	1	1	1.1	0.03	4.6	3	8	0.99
24	100	500	4914	2961	4.1	2.7	5.8	0.87	1	0.9	1.1	0.04	7.1	5	9	1.25
25	167	500	4894	1828	7.3	4.4	10.5	1.45	1.8	1.8	1.9	0.04	7.5	5	10	1.31
26	5	600	7069	9917	0	0	0.1	0.01	1.8	1.8	1.8	0	2	2	2	0
27	10	600	7072	8307	0.1	0.1	0.2	0.01	0.6	0.6	0.6	0	3	3	3	0
28	60	600	7054	4498	2	0.8	3.5	0.56	0.6	0.5	0.6	0.03	5.9	3	9	1.34
29	120	600	7042	3033	4.8	3.1	7.1	1.13	1.1	1.1	1.4	0.05	7	5	10	1.4
30	200	600	7042	1989	8.4	5.3	11.9	1.61	1.5	1.5	1.7	0.06	7.3	5	10	1.19
31	5	700	9601	10086	0	0	0.1	0.01	0	0	0	0	2	2	2	0
32	10	700	9584	9297	0.2	0.1	0.2	0.02	0.2	0.2	0.2	0.02	3	3	3	0
33	70	700	9616	4700	2.8	1.4	5.2	0.65	1.3	1.3	1.4	0.03	6.8	4	11	1.33
34	140	700	9585	3013	6.3	3.9	10.2	1.53	1.5	1.5	1.6	0.05	7.8	5	12	1.64
35	5	800	12548	10400	0	0	0.1	0.01	0.1	0.1	0.1	0	2	2	2	0
36	10	800	12560	9934	0.2	0.1	0.2	0.02	0.1	0.1	0.1	0	3	3	3	0
37	80	800	12564	5057	3.1	2.1	5.2	0.65	1.1	1.1	1.1	0.01	6.7	5	10	1.16
38	5	900	15898	11060	0	0	0.1	0.01	0.8	0.8	0.8	0	2	2	2	0
39	10	900	15896	9423	0.1	0.1	0.1	0.01	0.3	0.3	0.3	0	2	2	2	0
40	90	900	15879	5128	2.8	1.2	4.5	0.69	0.9	0.9	0.9	0.02	5.6	3	8	1.13

ast alternate algorithm

Table 18: Alternate Algorithm - Random (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
1	5	100	198	5819	0.1	0.1	0.2	0.03	2.9	2.9	2.9	0	4.3	4	5	0.44
2	10	100	193	4093	0.3	0.2	0.5	0.09	22.8	21.8	26.4	1.78	5.2	4	8	1.2
3	10	100	198	4250	0.3	0.1	0.5	0.1	13.5	11.4	22.6	3.08	5.8	3	9	1.49
4	20	100	196	3034	0.9	0.5	1.5	0.26	16.8	7.2	25	3.57	7.4	5	12	1.89
5	33	100	196	1355	2.1	1.1	3.3	0.53	23.4	14.4	33.1	5.87	10.7	6	16	2.43
6	5	200	786	7824	0.1	0.1	0.3	0.05	14.8	13.8	17.6	1.66	4.6	3	9	1.26
7	10	200	779	5631	0.3	0.2	0.5	0.06	13.5	12	19	2.42	4.9	4	7	0.77
8	20	200	792	4445	0.9	0.4	1.6	0.25	10.4	9.1	11.8	0.98	7.6	4	13	1.86
9	40	200	785	2734	1.9	1.3	3.2	0.37	22.2	20.8	23.8	1.37	8	6	13	1.37
10	67	200	786	1255	4.4	2.2	7	0.99	39.1	38.3	45.6	1.24	10.8	6	17	2.2
11	5	300	1772	7696	0.2	0.1	0.2	0.02	2.1	2.1	2.1	0	5	5	5	0
12	10	300	1758	6634	0.3	0.3	0.5	0.07	14	13.1	14.6	0.73	5.6	5	7	0.84
13	30	300	1760	4374	2	0.8	4	0.74	14.6	11.5	19.2	2.43	10.6	5	21	3.69
14	60	300	1771	2968	4	2.4	6	0.94	25.2	22.5	28.2	1.3	11	7	16	2.31
15	100	300	1754	1729	6.4	3.9	9.1	1.12	34.3	29	37.7	1.88	10.7	7	14	1.64
16	5	400	3153	8162	0.1	0.1	0.2	0.02	3.2	3.2	3.2	0	5	5	5	0
17	10	400	3142	6999	0.3	0.2	0.4	0.05	12.6	12.4	12.9	0.22	4.8	4	6	0.77
18	40	400	3134	4809	2.4	1.3	4.8	0.66	14.6	9.6	19.5	3.06	10	6	19	2.33
19	80	400	3134	2845	5.7	3.7	8.1	1.04	22.7	19.3	26.1	1.74	11.6	8	16	1.88
20	133	400	3144	1789	10.2	6.1	14.9	1.88	26	23.5	28.7	1.45	12.5	8	18	2.13

Table 18: Alternate Algorithm - Random (50 Runs)

Test Problem	p	n	Edges	Optimal Solution	Computational Time (seconds)				Gap to Optimal (percent)				Iterations			
					Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev	Mean	Min	Max	Std Dev
21	5	500	4909	9138	0.2	0.1	0.2	0.04	12.4	12.4	12.4	0	5.8	5	7	0.93
22	10	500	4896	8579	0.2	0.2	0.3	0.03	15.5	15.5	15.5	0	4	4	4	0
23	50	500	4903	4619	2.6	1.6	4.3	0.56	13.8	11.6	16.2	1.2	8.7	6	14	1.62
24	100	500	4914	2961	6.9	4.6	11.4	1.23	18.8	17.3	20.5	0.66	11.4	8	17	1.74
25	167	500	4894	1828	13.4	7.6	20.9	2.42	29.1	24.7	32.8	2.03	13.1	8	20	2.15
26	5	600	7069	9917	0.1	0.1	0.1	0.02	4.3	4.3	4.3	0	3	3	3	0
27	10	600	7072	8307	0.3	0.2	0.4	0.08	11.9	11.7	12.2	0.22	5.4	4	7	1.03
28	60	600	7054	4498	3.6	1.6	5.2	0.73	17.3	16	18.7	0.79	9.8	5	14	1.8
29	120	600	7042	3033	9.1	5	13	1.52	20	17.9	23.1	1.15	12.3	7	17	1.86
30	200	600	7042	1989	15.5	10.6	19.9	2.18	29	25.8	31.9	1.49	12.6	9	16	1.62
31	5	700	9601	10086	0.2	0.1	0.3	0.05	13.5	13	15.1	0.86	6	4	8	1.37
32	10	700	9584	9297	0.3	0.3	0.4	0.03	14.2	14.1	14.2	0.04	5.2	5	6	0.43
33	70	700	9616	4700	4.4	2.8	6.6	0.97	19.1	17.7	21.1	0.85	10.1	7	15	1.91
34	140	700	9585	3013	11.3	8.3	15.8	1.82	20.9	19.3	22.9	0.88	13.1	10	18	1.94
35	5	800	12548	10400	0.1	0.1	0.2	0.03	1.4	1.4	1.4	0	4.5	4	5	0.5
36	10	800	12560	9934	0.2	0.2	0.3	0.02	15.5	15.5	15.5	0	4	4	4	0
37	80	800	12564	5057	5.1	2.2	9.1	1.46	13.2	11.6	15.1	0.81	10.5	5	18	2.76
38	5	900	15898	11060	0.2	0.2	0.3	0.03	4.4	4.4	4.4	0	6	6	6	0
39	10	900	15896	9423	0.5	0.4	0.6	0.03	13.8	13.6	14.2	0.26	7	7	7	0
40	90	900	15879	5128	7.1	4.8	9	1.02	14.3	12.4	15.6	0.93	12.7	9	16	1.73