An analysis of heuristics for the p-median problem

Assignment 2: Preliminary Report

Assignment Unique Number: 650898

Gemma Dawson

50223909

Study Leader: Ms J L le Roux

30 August 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:

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 heuristic and metaheuristic 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 [4]. 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 [5] 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. [6] 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 [7] while Alp et al. [8] investigated their proposed genetic algorithm. Hansen and Mladenović [9] 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. [10] 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. [11]. 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 [1], 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 [2] 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 [3]. 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 an the 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 introducing 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 100 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 100 runs was timed.

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. [6] 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 the same as 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 by 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 increase (or decrease) to S* resulting from the interchange of
          node r with median t

I, J, K_j : subsets of the n nodes as defined in the pseudocode

a, b, z, q : indices and parameters as defined 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 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

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

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.
$$S^* = S_r \text{ and STOP}$$

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

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

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.

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

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
Set $I = \emptyset$ and 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.

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

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.
$$S^* = S_r \text{ and STOP}$$

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

$$\text{Set } P^* = M; \ P = \emptyset \text{ and } u_i = 0 \text{ for all } i = 1, 2, \ldots, n$$

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.

$$P^* = P^* \backslash \{r\} \qquad\qquad P = P \cup \{r\}$$

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.
$$S^* = S_r \text{ and STOP}$$

6.4   Computational Results

The average solution accuracy for each of the forty test problems from both the greedy algorithm and the fast greedy algorithm as well as the average computational time can be seen in table 1.

Figure 1 shows the average accuracy of the solution's objective function value 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 slight decreases as the number of medians required increased. In table 1, you can see that all the average objective function value was at most 4% less than the optimal objective function value.

The fast greedy algorithm produced results for about twenty problems that were 50% worse than the true optimal. The test problems with the highest accuracy rate when using the fast greedy correspond to the test problems that demanded five medians. The massive difference in accuracy for the fast greedy algorithm is clearly visible in figure 1 and as such the code for this algorithm needs to be reviewed.
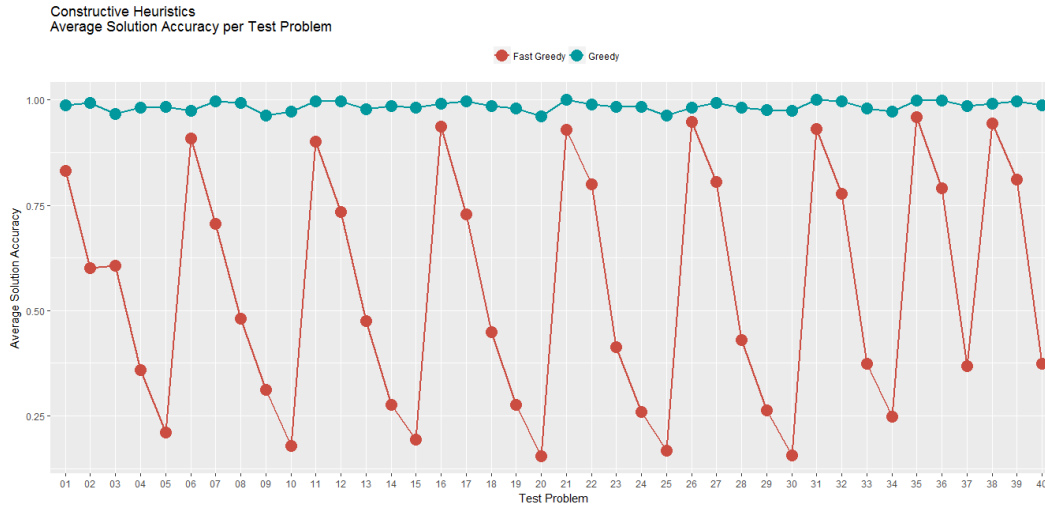

Figure 1: Average Solution Accuracy per Test Problem

In figure 2, the average computational time for both the greedy algorithm and the fast greedy algorithm increases as the number of medians demanded by the test problem increases. It can be seen in figure 2 that the fast greedy algorithm takes longer than the greedy algorithm for every test problem.
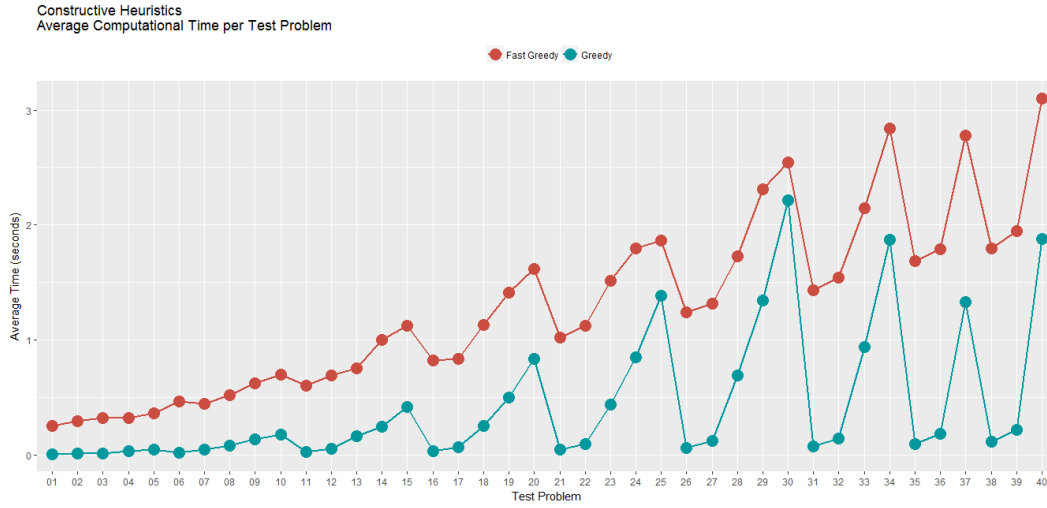
Figure 2: Average Computational Time per Test Problem

Figures 3 and 4 show the number of medians demanded by a test problem against the average computational time for the greedy algorithm and the fast greedy algorithm respectively. For the nine test problems with $p = 5$, the greedy algorithm produces a solution between 0.01 and 0.12 seconds where the increase in time can be attributed to the number of nodes in the network and this relatively small increase in time can be seen in figure 3. As the network size increases, so does the spread between the average computational time required by the greedy algorithm to return a solution.
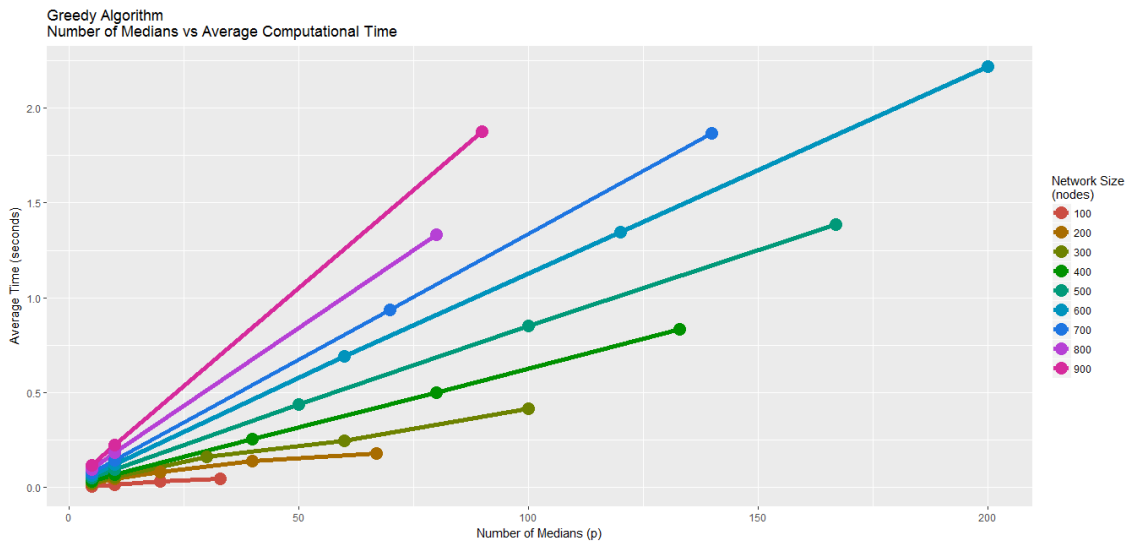

Figure 3: Greedy Algorithm

It can be seen in figure 4 that the fast greedy algorithm has not lived up to its name. The average time taken to produce results for the $p = 5$ test problems was between 0.26 and 1.79 seconds. This spread can be seen in figure 4 and can be attributed to the size of the network. This may be due to the additional step of identifying the nodes that were reassigned to a new median in the previous iteration. The design of the fast greedy algorithm's code needs to be reviewed to determine if the full power of R is being exploited. The algorithmic complexity of both the greedy algorithm and the fast greedy algorithm need to be evaluated in order to determine if it is possible for this algorithm to be faster than the greedy algorithm.
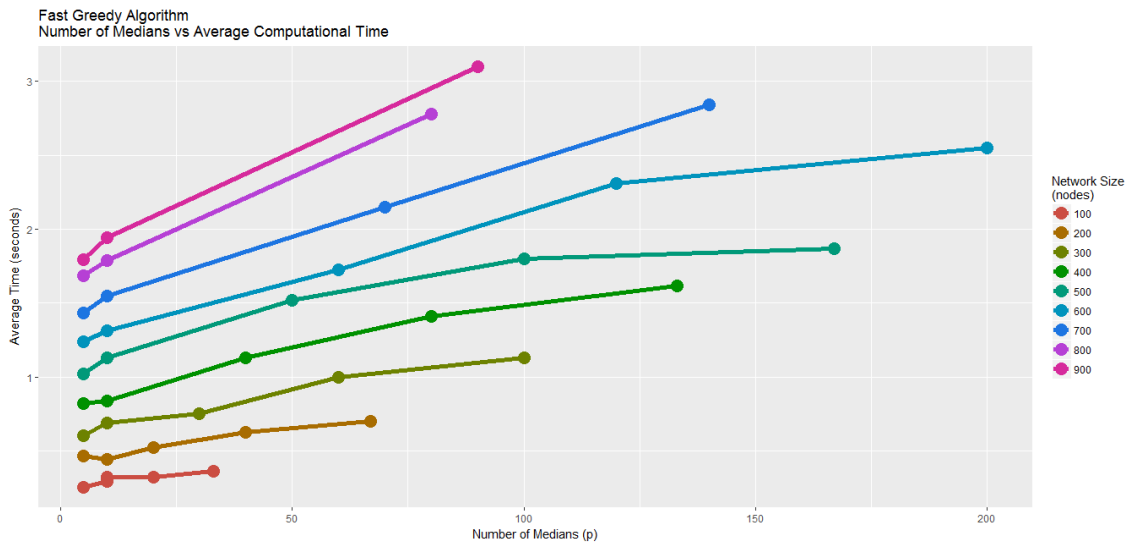

Figure 4: Fast Greedy Algorithm

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. The substantial drop in accuracy as the number of medians demanded increases is far too large to accept, even if the algorithm were producing results faster than the greedy algorithm.

The code for the stingy algorithm has been completed, but still needs to be thoroughly reviewed before running the 100 iterations for each of the forty test problems. Once this has been completed, the results along with the computational time will be included in the final report of this study. These results will also be analysed and compared to both the greedy algorithm and the fast greedy algorithm. The expectation is that this algorithm will take longer to run for the small median problems, but for test problems requiring a large number of medians, the computational time may be comparable.

Table 1: Constructive Heuristic Results

Test Problem	Medians (p)	Network (nodes)	GREEDY		FAST GREEDY	
			Av. Sol. Accuracy	Av. Comp. Time (sec)	Av. Sol. Accuracy	Av. Comp. Time (sec)
01	5	100	0.99	0.01	0.83	0.26
02	10	100	0.99	0.02	0.60	0.29
03	10	100	0.97	0.02	0.61	0.32
04	20	100	0.98	0.03	0.36	0.32
05	33	100	0.98	0.05	0.21	0.36
06	5	200	0.97	0.02	0.91	0.47
07	10	200	1.00	0.05	0.71	0.44
08	20	200	0.99	0.08	0.48	0.52
09	40	200	0.96	0.14	0.31	0.63
10	67	200	0.97	0.18	0.18	0.70
11	5	300	1.00	0.03	0.90	0.61
12	10	300	1.00	0.06	0.73	0.69
13	30	300	0.98	0.16	0.48	0.75
14	60	300	0.99	0.25	0.28	1.00
15	100	300	0.98	0.42	0.19	1.13
16	5	400	0.99	0.03	0.94	0.82
17	10	400	1.00	0.07	0.73	0.84
18	40	400	0.99	0.25	0.45	1.13
19	80	400	0.98	0.50	0.28	1.41
20	133	400	0.96	0.84	0.15	1.62
21	5	500	1.00	0.05	0.93	1.02
22	10	500	0.99	0.09	0.80	1.13
23	50	500	0.98	0.44	0.41	1.52
24	100	500	0.98	0.85	0.26	1.80
25	167	500	0.96	1.39	0.17	1.87
26	5	600	0.98	0.06	0.95	1.24
27	10	600	0.99	0.12	0.81	1.31
28	60	600	0.98	0.69	0.43	1.73
29	120	600	0.98	1.35	0.26	2.31
30	200	600	0.97	2.22	0.16	2.55
31	5	700	1.00	0.08	0.93	1.43
32	10	700	1.00	0.15	0.78	1.55
33	70	700	0.98	0.94	0.37	2.15
34	140	700	0.97	1.87	0.25	2.84
35	5	800	1.00	0.10	0.96	1.68
36	10	800	1.00	0.18	0.79	1.79
37	80	800	0.99	1.33	0.37	2.78
38	5	900	0.99	0.12	0.94	1.79
39	10	900	1.00	0.22	0.81	1.95
40	90	900	0.99	1.88	0.37	3.10

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

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

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 qth node in the set of free nodes. The change in the objective function value if this interchange were to proceed is calculated.

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

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

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 objective 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 average accuracy of the solution obtained from the interchange algorithm of Teitz and Bart when presented with a solution from the greedy algorithm from section 6 as opposed to a feasible random solution can be seen in table 2. This table also presents the average computational time the algorithm took to complete given the two different input solutions.

Figure 5 shows the average computational time taken to produce a final result by the Teitz and Bart algorithm when presented with a solution that was produced by the greedy algorithm. When the value of p is small, the computational time required is very low and this is quite likely due to the good quality of the greedy algorithm solution. The number of interchanges required to improve the starting solution was not recorded for these results but this statistic will be included and analysed in the final report. What is expected is that as the number of demanded medians increases, the number of interchanges required to find the final result will increase. The number of interchanges processed will directly affect the computational time.

Table 2: Local Search Heuristic Results - Interchange Algorithm of Teitz & Bart

| Test Problem	Medians (p)	Network (nodes)	GREEDY		RANDOM	
			Av. Sol. Accuracy	Av. Comp. Time (sec)	Av. Sol. Accuracy	Av. Comp. Time (sec)
01	5	100	1.00	0.07	1.00	0.08
02	10	100	1.00	0.07	1.00	0.15
03	10	100	1.00	0.16	1.00	0.19
04	20	100	1.00	0.12	1.00	0.22
05	33	100	1.00	0.17	0.99	0.32
06	5	200	1.00	0.15	1.00	0.16
07	10	200	1.00	0.18	1.00	0.25
08	20	200	1.00	0.29	1.00	0.59
09	40	200	0.99	0.79	0.99	0.94
10	67	200	0.99	0.71	0.99	1.39
11	5	300	1.00	0.16	1.00	0.24
12	10	300	1.00	0.30	1.00	0.41
13	30	300	1.00	0.81	1.00	1.44
14	60	300	1.00	1.41	1.00	2.89
15	100	300	0.99	2.30	0.99	4.22
16	5	400	1.00	0.39	1.00	0.35
17	10	400	1.00	0.49	1.00	0.70
18	40	400	1.00	2.33	1.00	2.84
19	80	400	0.99	4.26	0.99	4.97
20	133	400	1.00	5.78	0.99	7.67
21	5	500	1.00	0.16	1.00	0.42
22	10	500	0.99	0.63	0.99	1.54
23	50	500	1.00	4.56	1.00	8.92
24	100	500	1.00	7.07	0.99	10.98
25	167	500	0.99	15.23	0.99	14.53
26	5	600	1.00	0.51	1.00	0.68
27	10	600	1.00	1.38	1.00	1.58
28	60	600	1.00	6.32	1.00	14.04
29	120	600	1.00	15.45	0.99	15.97
30	200	600	0.99	15.47	0.99	20.10
31	5	700	1.00	0.28	1.00	0.79
32	10	700	1.00	1.09	1.00	2.02
33	70	700	0.99	9.91	1.00	15.23
34	140	700	1.00	21.84	1.00	20.61
35	5	800	1.00	0.75	1.00	1.42
36	10	800	1.00	1.49	1.00	1.91
37	80	800	1.00	18.56	1.00	21.28
38	5	900	1.00	1.34	1.00	1.99
39	10	900	1.00	1.59	1.00	2.11
40	90	900	1.00	24.19	0.99	27.81
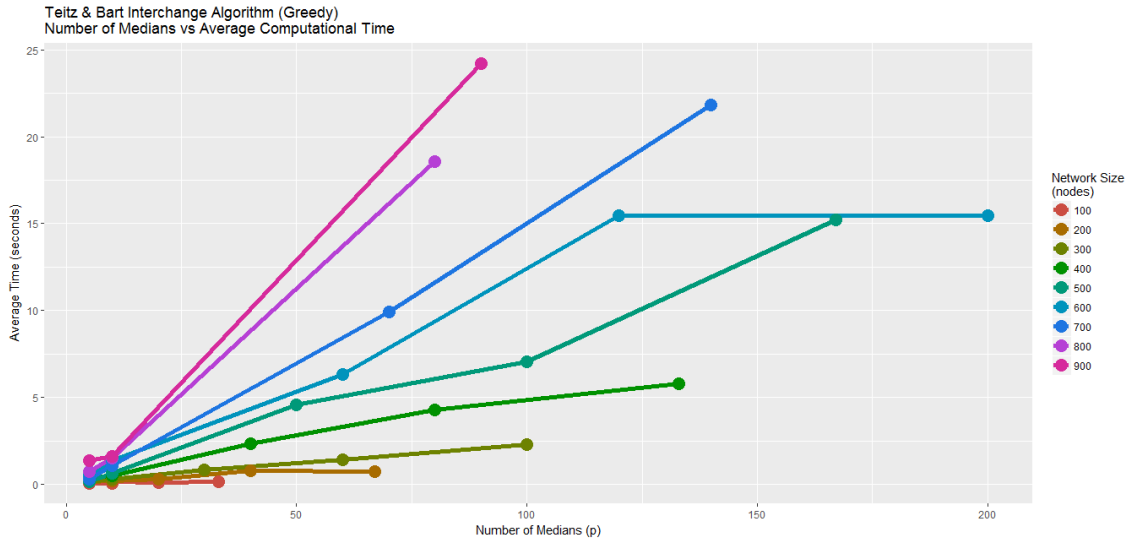
Figure 5: Interchange Algorithm of Teitz and Bart with a Greedy Starting Solution

Figure 6 shows the average computational time taken to produce the final result by the Teitz and Bart algorithm when presented with a solution that was produced by random feasible solution. Clearly, more interchanges will be required before the algorithm finds a good solution and stops and as such, more time is required. This can be seen in the bigger spread in computational time for the $p = 5$ test problems.
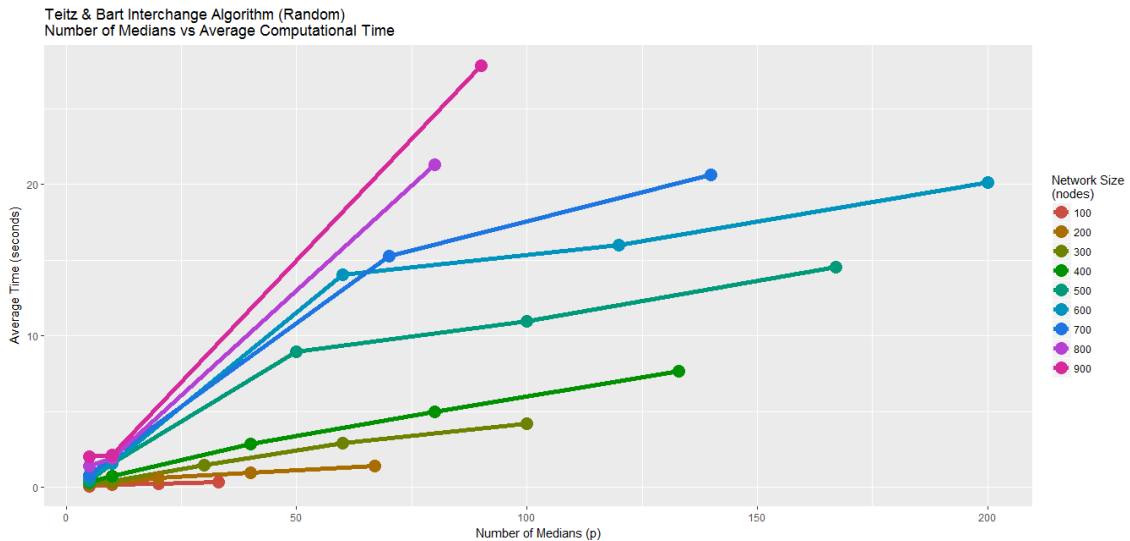

Figure 6: Interchange Algorithm of Teitz and Bart with a Random Starting Solution

Generally, using a greedy starting solution does result in some computational time

savings during the local search when compared to using a random solution. However, there are some exceptions with test problems 25, 29, and 30. These three problems have large networks and demand a high number of medians.

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

$$\text{Set } k = z = q = 1 \text{ and } b = m - p$$

$$\text{Define } P = M - P^*$$

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

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

$$\text{where } I = \{\text{all } i \in G : d_{ir} < u_i^k\}$$
$$\text{and } K_j = \{\text{all } i \in G : d_{ir} \geq u_i^k \text{ and } d_{ij} - u_i^k\}$$

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**

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

]

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 $d_{is} = u_i^k$

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

[

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 $d_{is} = u_i^k$

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

]

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**

$$z = z + 1,\, q = q + 1 \text{ and go to STEP 1}$$

]

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.

$$a = \tfrac{q}{b} \text{ and STOP}$$

7.2.2   Results

Table 3 presents the average solution accuracy and the average computational time for the fast interchange algorithm. The results from using a solution obtained from the greedy algorithm given in section 6 as well as a randomly created feasible solution are provided alongside one another for comparison.

A comparison of the number of required medians versus the average computational time is given in figure 7 for a greedy starting solution and in figure 8 for a random staring solution. Once again, when the algorithm was presented with a good solution it was quicker to find a final answer. It can also be seen that as the size of the network and required number of medians increases, so does the computational time required by the algorithm.

Figure 7 and figure 8 are very similar, but when the algorithm is presented with a random solution, it does take longer for every test problem. Generally, the larger the number of nodes in the network along with the higher the number of required medians, the bigger the difference in time between the greedy solution and the random solution with the biggest difference occurring in test problem 30 which has the highest value of p out of all the forty test problems. When given the greedy solution for test problem 30, the fast interchange completed after 21.16 seconds compared to the 24.51 seconds for the random solution.
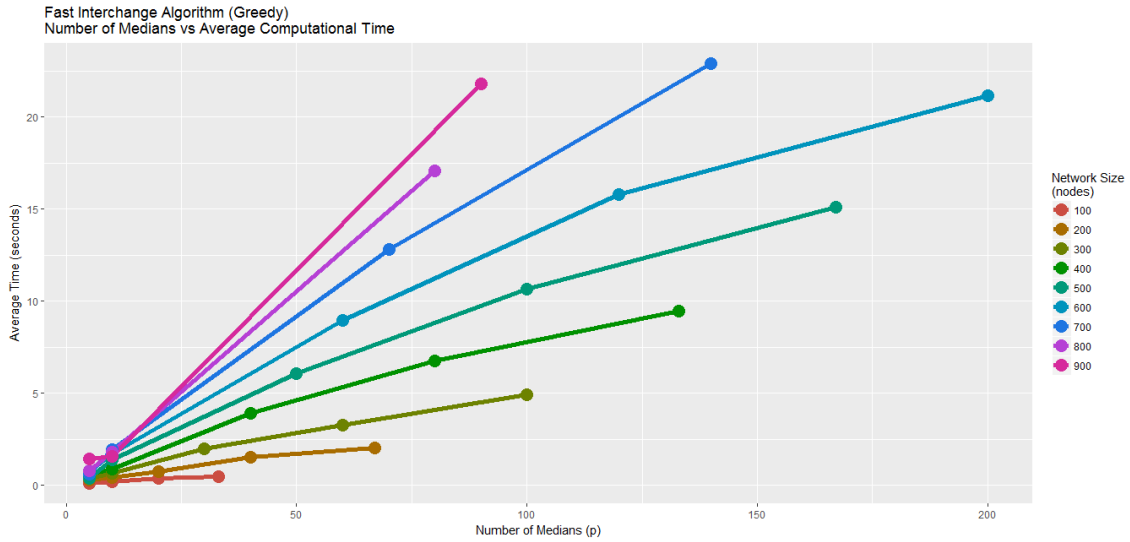
Figure 7: Fast Interchange Algorithm with a Greedy Starting Solution
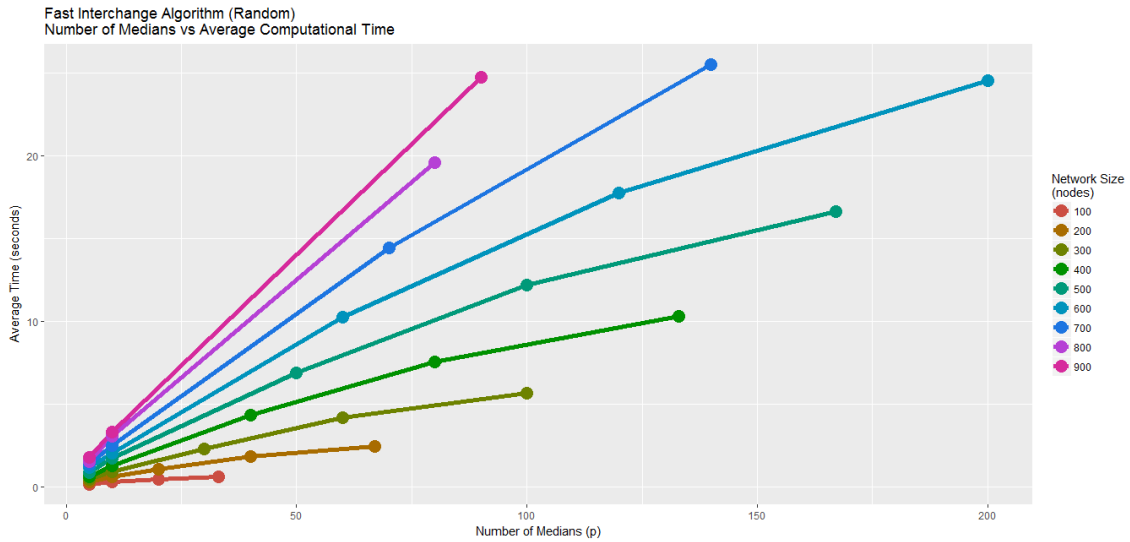

Figure 8: Fast Interchange Algorithm with a Random Starting Solution

Once again, this algorithm does not live up to its name with the fast interchange algorithm producing results faster than the interchange of Teitz and Bart on only four occasions. As such, the design and code of this algorithm needs to be reviewed in order to see where possible savings can be found.

Table 3: Local Search Heuristic Results - Fast Interchange Algorithm

Test Problem	Medians (p)	Network (nodes)	GREEDY		RANDOM	
			Av. Sol. Accuracy	Av. Comp. Time (sec)	Av. Sol. Accuracy	Av. Comp. Time (sec)
01	5	100	1.00	0.11	1.00	0.16
02	10	100	1.00	0.16	1.00	0.29
03	10	100	1.00	0.19	0.97	0.28
04	20	100	1.00	0.37	0.99	0.44
05	33	100	1.00	0.45	0.97	0.61
06	5	200	1.00	0.27	1.00	0.30
07	10	200	1.00	0.40	1.00	0.60
08	20	200	1.00	0.71	1.00	1.06
09	40	200	0.99	1.52	0.99	1.81
10	67	200	0.99	2.00	0.98	2.45
11	5	300	1.00	0.36	1.00	0.44
12	10	300	1.00	0.62	1.00	0.92
13	30	300	1.00	1.99	0.99	2.30
14	60	300	1.00	3.24	0.99	4.18
15	100	300	0.99	4.93	0.97	5.64
16	5	400	1.00	0.58	1.00	0.58
17	10	400	1.00	0.86	1.00	1.26
18	40	400	1.00	3.89	1.00	4.31
19	80	400	0.99	6.76	0.99	7.56
20	133	400	0.99	9.44	0.97	10.31
21	5	500	1.00	0.38	1.00	0.86
22	10	500	0.99	1.37	0.99	1.72
23	50	500	1.00	6.06	0.99	6.89
24	100	500	1.00	10.66	0.99	12.19
25	167	500	0.99	15.12	0.98	16.59
26	5	600	1.00	0.76	1.00	1.15
27	10	600	1.00	1.68	1.00	2.05
28	60	600	1.00	8.95	0.99	10.24
29	120	600	1.00	15.80	0.99	17.73
30	200	600	0.99	21.16	0.98	24.51
31	5	700	1.00	0.57	1.00	1.35
32	10	700	1.00	1.93	1.00	2.49
33	70	700	0.99	12.82	0.99	14.42
34	140	700	0.99	22.89	0.99	25.48
35	5	800	1.00	0.78	1.00	1.50
36	10	800	1.00	1.79	1.00	3.05
37	80	800	1.00	17.07	0.99	19.58
38	5	900	0.99	1.43	0.99	1.79
39	10	900	1.00	1.56	1.00	3.28
40	90	900	1.00	21.83	0.99	24.73

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

STEP 0: Initialisation
Randomly assign p nodes to P^* and set $P = M - P^*$

STEP 1: Assign each node in the network to its closest median.

$$K_j = \{\text{all } i \in G \text{ such that } \min(d_{ij})\} \text{ where } j \in P^*$$

STEP 2: Find the median node for each current median's set of associated nodes.

$$\text{Find } k_j \in K_j \text{ where } \min\left[\sum_{i \in K_j} d_{ik}\right]$$

STEP 3: Update P^* with the new set of medians.

$$P^* = \{\text{all } k_j\} \text{ for } j \in P^*$$

7.3.2   Results

The code for this algorithm has not been completed and as such there are no results that can be presented at this time.

7.4   Computational Results

Figure 9 shows the average accuracy of the objective function value obtained from the
two algorithms that were run for both of the different starting solutions. The results
fluctuate between returning a result that is 3% worse than the optimal and the perfect
solution. The fast interchange algorithm using a random starting solution produced the
most variation. It is the fast interchange with a random starting solution that results
in the greatest variation in accuracy, while the fast interchange with a greedy solution
along with both variations in the interchange algorithm of Teitz and Bart produce
consistently accurate results.
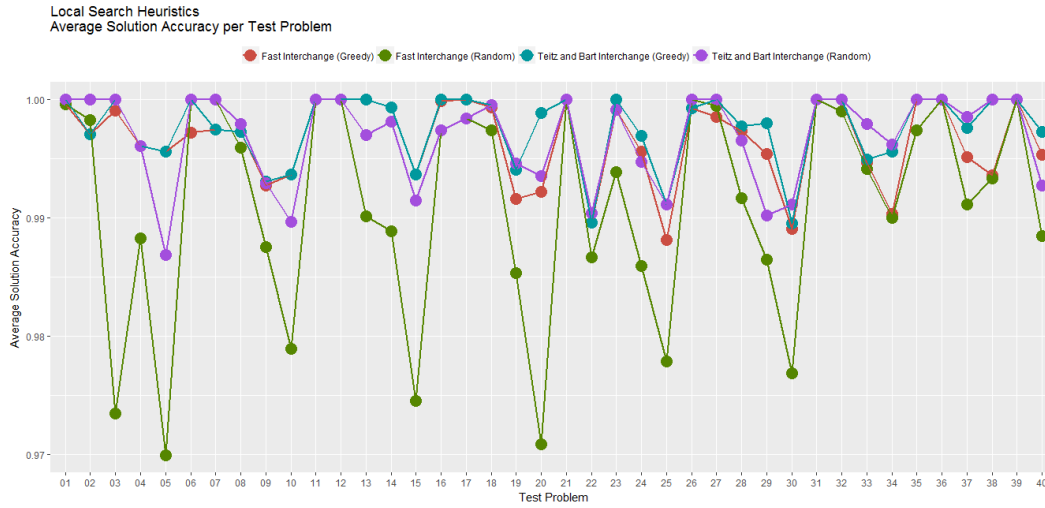
Figure 9: Average Solution Accuracy per Test Problem

The average computational time required by the two algorithms shows similar results to
that of the constructive greedy algorithm when the value of p is small, but as the network
size increases along with the value of p, the computational time increases dramatically.
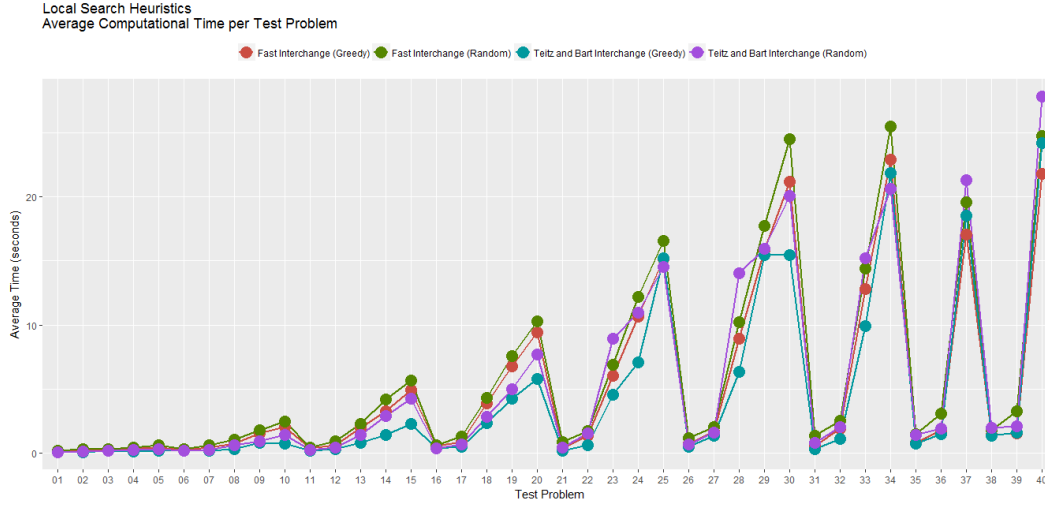
Figure 10: Average Computational per Test Problem

8   Conclusion

The results from the local search heuristics offer only a small improvement on those produced by the constructive greedy algorithm from section 6 as that algorithm produced solutions that were up to 4% worse than the optimal value. The increase in time between the constructive greedy algorithm and the local search algorithms are significantly large particularly for larger networks that require a lot of medians. Test problem 30 took just over 2 seconds to produce a solution that was on average 97% accurate while the interchange algorithm of Teitz and Bart and the fast interchange algorithm both improved this accuracy by 2%, but at a cost of an additional 13 and 19 seconds respectively. Of course, context is king and for many real-world problems a 2% improvement could equate to vast amounts of savings and as such this small improvement may be worth the additional computational time that is required.

Looking at the difference in time between both local search algorithms starting with a random solution and a greedy solution, it can be seen that the additional time taken to improve the random solution is generally greater than the time taken by the greedy algorithm to find a starting solution.

9   Remaining Work

In the coming weeks, the literature study in section 3 will be extended to give more background into the algorithms investigated in this study and the section giving an overview of heuristics may also be broadened.

Of the six algorithms currently included in this study, only the alternate algorithm needs to coded. This will be done and the results from running this code will be added to the local search computational results. The stingy algorithm code has been completed, but needs to be revised as the computational time required for even small problems is excessively high. The results from this algorithm will also be included and analysed, even if code cannot be improved.

The complexity of all the algorithms will be added to their respective subsections and may be referred to in the subsection comparing the results if appropriate.

As has already been mentioned, the fast greedy algorithm and the fast interchange algorithm are taking longer to run than the original algorithm. This goes against what was expected and the reason that this is occurring needs to be determined. Current thoughts are that algorithm code could be improved as well as the fact that the design R language is able to determine the closest median for all nodes in the network a computationally inexpensive which makes the proposed improvements offered by the two "fast" algorithms redundant. R code can be profiled to determine where bottlenecks slowing down the processing time exist and from this information, hopefully, the code can be improved. Vectorisation is a key component of the R language as it allows for an increase in speed. It is currently not known if this has been fully exploited in the code that has been written for this study and as such this needs to be assessed and applied.

Given time, a section on metaheuristics used to solve the p-median problem will also be added. At least two metaheuristics will be coded and used to solve the forty test problems. These results will also be added to the final report and compared to those produced in the classic heuristic sections.

10   References

[1] 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/

[2] 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.

[3] 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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[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.

11 Index of Terms

12   Index of Authors