Controller Placement Methods Analysis

Joseph Hollinghurst, Ayalvadi Ganesh
University of Bristol
United Kingdom
[j.hollinghurst,a.ganesh]@bristol.ac.uk

Timothy Baugé
Thales UK
United Kingdom

***Abstract*—Software-Defined Networking (SDN) offers flexibility and programmability to the network infrastructure through the introduction of a controller. However, the controller introduces extra delay into the system as new data flows must query the controller for instructions of how to route traffic. This becomes an increasing problem for large scale and delay sensitive networks such as those found in high-criticality infrastructure. The delay introduced can be minimised by optimal placement of the controller or decreased further by introducing additional controllers. Although the problem of optimal placement for multiple controllers is known to be NP hard, approximations can be used. The analysis of four different methods has therefore been conducted and looks at the scalability, through the lens of complexity. It is found the four methods, full search, linear programming, local search and an adapted version of the k-means++ algorithm, vary significantly in their complexity. It is also found that the accuracy of the methods varies with the complexity, creating a definitive trade-off between the two attributes.**

Keywords; SDN, Controller Placement, Optimal, k-median, linear programming, local search, k-means

I. INTRODUCTION

Legacy networks typically co-locate the control and forwarding elements, this often means the network is inflexible and bespoke networks must be used for applications when specific requirements such as Quality of Service are required. SDN decouples the control element (control plane) and forwarding element (data plane). A controller is then used to service one or more forwarding elements in the network. This allows for programmability in the network creating a dynamic platform that can be used to create new networking services using COTS (commercial of the shelf) hardware.

However, by physically moving the control plane additional delay is introduced in the system. This is caused by unknown traffic having to query the controller for forwarding instructions as with the OpenFlow protocol[1]. Hence, to minimise this additional delay to an acceptable level, the controller placement and number of controllers become of paramount importance. The problem of optimal placement is addressed in [2] where it is considered analogous to the mathematical k-median problem. This involves finding k controllers that minimise the delay from the networking nodes to the controllers. The authors use a full search approach to find the optimal placement for the given networks. Although the full search method will always find the optimal solution, it becomes infeasible for networks with a large number of nodes due to an extensive amount of computation. Fortunately, the k-median problem is a well-known NP complete problem and therefore has many existing approximation algorithms [3, 4]. The main focus of this paper is therefore to compare the scalability and accuracy of known algorithms for the k-median problem, applied specifically to networking.

II. PROBLEM FORMULATION

The k-median problem asks: if we are a given a network, how do we optimally place k controllers in the system such that every node is linked to a controller in such a way that a metric is minimised. This formulation means that the solution to the problem is a set of k clusters of nodes, with each cluster containing a controller and each node within the cluster linked to that specific controller. We assume that a controller is always located on a pre-existing node and the control traffic is sent over the existing infrastructure.

A mathematical formulation of this problem is as follows: Let G=(N, E) be a network where N is the set of vertices and E the set of edges. Each edge i has a cost c_i representing the link latency. The aim is to partition N into k disjoint clusters and allocate a controller node in each cluster such that the latency between each node and its allocated controller is minimised over the whole network. Let us denote:

- $C_{i,j}$ as the lowest cost path between nodes i and j
- P the set of all clustering permutations for N into k disjoint clusters of at least one vertex. The m^{th} member of P is defined as $P_m=\{n_{1,m}, n_{2,m}, n_{3,m},\ldots, n_{k,m}\}$ where each $n_{i,m}$ is a subset of N such that $n_{1,m} \cup n_{2,m} \cup \ldots \cup n_{k,m} = N$ and $n_{1,m} \cap n_{2,m} \cap \ldots \cap n_{k,m} = \{\ \}$
- $H_m= \{h_{1,m}, h_{2,m}, h_{3,m},\ldots, h_{k,m}\}$ a set of vertices where each $h_{i,m}$ belongs to $n_{i,m}$.

The problem described above can therefore be expressed as finding (P_m, H_m) such that F(m) is minimised, where:

$$F(m) = \sum_{i=1\ldots k} \sum_{j \in n_{i,m}} C_{h_{i,m},j} \qquad (1)$$

III. THE METHODS

A. Computing the metric

When looking at the controller placement problem it is important to be able to compute the metric for any particular cluster. Algorithm 1 gives a simple way to compute the metric.

Algorithm 1 Metric Calculation

Require: Network $G = (N, E)$
  Find all shortest paths, $C_{i,j}$
  Compute the metrics: $\sum_{j \in N} C_{i,j}$
  Find the minimum: $\min_{i \in J} \sum_{j \in N} C_{i,j}$
  return Controller location (minimum), Metric

The same method is used in each algorithm to find the metric. However, the shortest path calculation only needs to be performed once on the whole network and saved, it can then be passed into the algorithm as an input.

B. Full Search Method

The full search method looks at every possibility of clusters available. The metrics and corresponding clusters are then kept in memory and searched to find the minimum. In an algorithmic sense this works as described in Algorithm 2.

Algorithm 2 Full Search

Require: Network $G = (N, E)$, number of controllers k
  Create all possible clusters, B
  for $j = 1 : length(B)$ **do**
    Work through all clusters of a permutation
    for $m = 1 : k$ **do**
      Calculate the best placement and metric for each cluster in a permutation
    end for
    Save the metric and controller placements for the permutation
  end for
  Search for the Minimum
  return Clusters, Controller locations, Metric

C. Linear Programming Method

Linear Programming is a well-known optimisation technique and is well documented in literature [5] [6].To solve the k-median problem the linear programming method relies on finding an initial non-integer solution, rounding to find an integer solution, and then forming the clusters by linking each of the nodes to its nearest median. This is the method used in [7] and is summarised in algorithm 3.

Algorithm 3 Linear Program

Require: Network $G = (N, E)$, number of controllers k , number of swaps
  Solve the non-integer linear program
  Randomly round the non-integer solution to find the k controllers
  Form the clusters by linking the unused nodes to their nearest controller
  Compute the metric, M
  for $i = 1 : number\ of\ swaps$ **do**
    Randomly round the non-integer solution to find the k controllers
    Form the clusters by linking the unused nodes to their nearest controller
    Compute the metric, M'
    if $M' < M$ **then**
      $M = M'$
    end if
  end for
  return Clusters, Controller locations, Metric

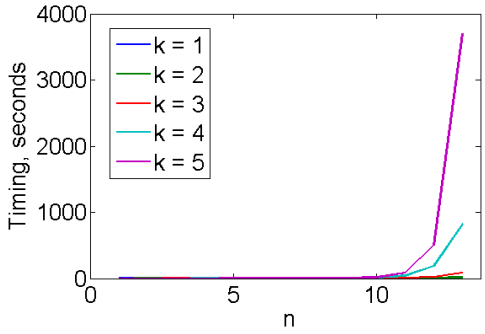
D. Local Search Method

For the local search methods initial clusters are initially formed randomly then nodes are swapped between clusters in each iteration. If the metric is improved with a swap then the configuration is kept, otherwise the previous configuration is used and another swap is performed. This approach is repeated for the number of iterations specified by the user as shown in the pseudo-code in algorithm 4.

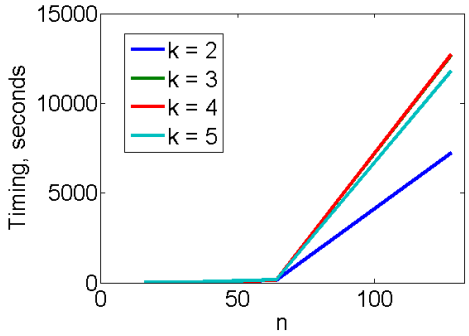
Algorithm 4 Local Search

Require: Network $G = (N, E)$, number of controllers k , number of swaps
  Randomly Form Initial clusters, C
  Compute the Metric on each cluster, call it M
  for $i = 1 : number\ of\ swaps$ **do**
    Choose a random cluster from C, call it c'
    Find the nearest node $\notin c'$
    Swap the node into c'
    Compute the new metric, call it M'
    if $M' < M$ **then**
      Keep the new clusters
      Let M = M'
    else
      Revert to previous clusters
    end if
  end for
  return Clusters, Controller locations, Metric

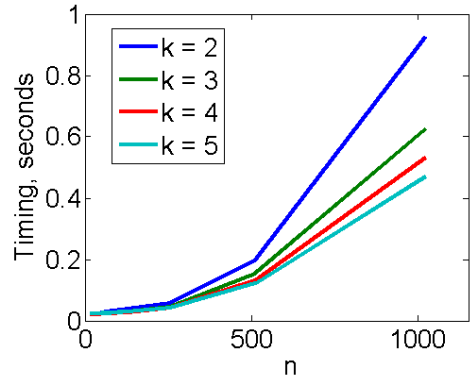
The Local Search Algorithm provides a fast method for finding a solution. However, the quality of the solution largely depends on the initial clusters chosen.



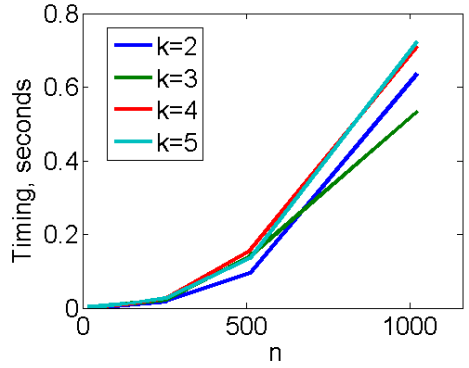
(a) Full Search



(b) Linear Programming



(c) Local Search



(d) K-means++ Adaptation

Figure 1. Numerical timings

E. K-means++ Adaptation

The K-means algorithm is arguably the most well-known clustering algorithm to date. The K-means++ algorithm extends the traditional k-means by introducing a way of computing the initial clusters [8]. This method allows the authors to bound the optimality of the algorithm, which is not possible with the traditional k-means. In this paper we are concerned with the k-median problem hence an adaptation is realised where the k-means++ is used to initially find clusters according to the k-means criterion, then the median of each cluster is found which then gives a solution to the k-median problem. This method can be formulated as in algorithm 5.

Algorithm 5 K-means adaptation

Require: Network $G = (N, E)$, number of controllers k , number of initialisations
Form clusters by using the k-means++ algorithm
for *Each Cluster* **do**
Find the median for each cluster
Compute the metric
end for
return Clusters, Controller locations, Metric

IV. Timing and Complexity

A. Metric Complexity

In order to compute the metric for each method the shortest path matrix must be calculated. This can be calculated using Johnson's algorithm which has $O(n^2\log(n) + n\log(n))$ complexity[9]. When computing the metric a sum of all the rows or columns must then be performed to gain the metric for each node, this requires a total of $(n-k+1)^2$ additions. A search for the minimum must then be performed on each row but this can be processed in $O(n-k+1)$, meaning a total complexity of $O((n-k+1)^2)$. This leads to a total complexity of $\max\{ O(n^2\log(n) + n\log(n)), O((n-k+1)^2)\}$.

B. Full Search

The complexity of this method can be analysed by looking at the number of combinations of possible clusters, the complexity of computing the metric and searching for the minimum. First, the number of possible permutations is a Stirling number of the second kind, the leading term in the Stirling summation is $k^n/k!$. Next, the metric has $(n-k+1)^2$ operations, shown in section IV.A. Hence, by the theorem on Polynomial orders[10] this leads to a complexity of $O((k^n/k!)(n-k+1)^2)$.

C. Linear Programming

The complexity of the linear programming method is dominated by the interior point algorithm as described in [11]. This method has complexity $O\left(\frac{n^3}{\ln(n)}\lceil n\log_{10}(2)\rceil\right)$ where n is the size of the network [12].

D. *Local Search*

First, to randomly generate initial clusters a loop of size k can be iterated, with $\lfloor n/k \rfloor$ nodes being chosen each time. Hence, this step requires $k\lfloor n/k \rfloor$ operations. For the metric to be computed the complexity can be taken from section IV.A, which is $(n-k+1)^2$. Next, within the loop in order to find the nearest node a search of $max(n-k)$ nodes is required and another metric calculation is performed. As the local search method has a user defined amount of repetitions, R, the complexity becomes $O(R(n-k+1)^2)$. Finally, the complexity is found to be $O(max(R(n-k+1)^2, n^2log(n)+nelog(n))$ as the shortest path calculation must be considered.

E. *K-means++*

The K-means++ has no proven complexity bounds. However, the simpler k-means algorithm has shown to be bounded in complexity to $2^{O\sqrt{n}}$, where n is the number of points in the network [13]. This is a worst case theoretical bound and the practical results show much better performance.

F. *Methodology for timing*

For each of the methods described the timings are calculated by implementing the algorithms in MATLAB version 2012a. The system used an i7 3GHz processor with 8GB RAM running on a 64-bit version of windows 7. The initial network used to test the full search algorithm is made up of a single 16 node network with unit length links. For the approximation algorithms the 16 node network is then duplicated to create a network of size 32, 64, 128, 256, 512 and 1024. The algorithms were all run 101 times on each network, with the first implementation discarded and an average taken. Figure 1 shows the results of the timings. In Figure 1 (c) 200 swaps were performed as this is comparable with the k-means++ method with 1 initialisation.

V. ACCURACY

In this section the results are presented from testing the approximation algorithms on a number of different networks. This was done to test the effectiveness of the algorithms on a variety of network topologies.

A. *Method for testing accuracy*

In order to test the accuracy of the methods a test network was formed. In order to test real world applicability the test network was chosen to be a rail network. The network is formed of 124 stations, i.e. nodes, where each node only has 1, 2 or 3 links to other stations. This means the network is sparse in nature. The full search method was not used in this section as the number of nodes was too high. In each test the algorithms were run 100 times in order to compute the statistics.

B. *Results*

A number of results were found, first of all it is apparent that the linear programming method was superior to the other methods in terms of accuracy. Figure 3 shows that the linear programming method gives the best metric. Furthermore, the metric is consistent over each initialisation, shown by the small standard deviation. This is believed to be because the non-integer linear program provides a good initial solution, so the rounding to an integer solution does not affect the metric in each instance.

Second, it can be seen that the number of repetitions improves the local search method significantly, up to a point where it is competitive with the linear program. At 10 and 100 repetitions the algorithm performs particularly badly, but when increasing the number of repetitions to 200 a significant improvement in performance can be seen. Increasing the repetitions further to 500 and then 1000 repetitions the minimum metric is very close to that of the linear programming. This is an interesting point as the complexity of the local search method is significantly less than that of the linear programming, and importantly can be used on much larger scale problems.

Although the metric improves with the number of repetitions for the local search method, this is not the case for the k-means++ method. When looking at Figure 4 it can be seen that having a larger number of initialisations does not improve the minimum metric, this could be caused by the method finding a local minimum from the initialisation each time or due to the rounding involved to find the median. Furthermore, the standard deviation of the method is smaller than that of the local search method, indicating less variation, which in this case is not always a good thing as the variation can lead to an improved solution.

TABLE I. RAW DATA

Linear Programming	k = 2	k = 4	k = 6	k = 8
Mean	17693.67	11004.04	8099.15	6311.207
Standard Deviation	3.66E-12	1.65E-11	1.55E-11	9.14E-13
Min	17693.67	11004.04	8099.15	6311.207
Max	17693.67	11004.04	8099.15	6311.207
Interquartile Range	0	0	0	0
Local Search	k = 2	k = 4	k = 6	k = 8
Mean	17693.67	12465.46	9245.76	7227.499
Standard Deviation	4.82E-12	1046.997	880.6258	556.6958
Min	17693.67	11004.04	8475.173	6321.83
Max	17693.67	13458.47	12195.89	8647.53
Interquartile Range	0	2188.92	558.4767	994.2567
K-means++	k = 2	k = 4	k = 6	k = 8
Mean	17750.43	13335.10	11199.17	9339.12
Standard Deviation	2.56E-11	993.05	853.52	1034.32
Min	17750.43	11216.09	8743.77	6612.96
Max	17750.43	14926.76	13460.34	11527.81
Interquartile Range	0	1167.42	657.1	1716.47

In order to show the differences in the methods Table I shows the raw statistics of 100 instances of each method, the local search method was run with 1000 repetitions and the k-means with 1 initialisation. In particular the interquartile range shows the significant difference between the methods, while the linear programming has a value of 0 for all the values of k, the local search and k-means methods have a large range for values of k = 4 and above. The variance in

the results for the local search method is believed to be because of the random initial clusters, this also partially determines how good the solution becomes after the swaps have occurred. The variance is slightly reduced in the case of the k-means due to the probability weighting applied to the initial clusters.

C. *Further Accuracy Tests*

In order to test the accuracy of the algorithms further a sample of topologies from the internet topology zoo was used [14]. The sample taken consisted of 127 different topologies, with sizes varying from 13 to 113 nodes. In order to show the performance of the k-means++ and local search on a single graph the CDF is presented for the minimums of both algorithms in Figure 2, with a normalised metric with respect to the linear program. The local search method used 200 swaps and the k-means used 1 initialisation to maintain a similar timing constraint, the value of k = 4 was used to give significant variation between methods. The main result from the graph shows that the local search method generates a metric the same as the linear program for 70% of networks, whereas the k-means++ algorithm only remains the same for 25%, showing the local search method outperforms the k-means++ in this scenario.

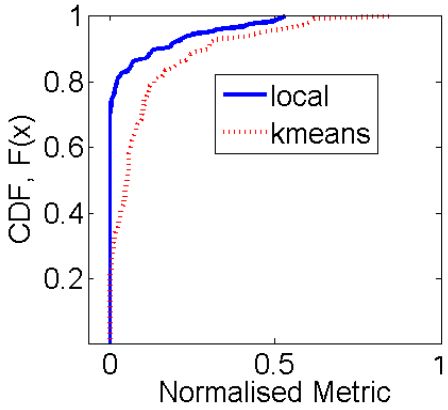
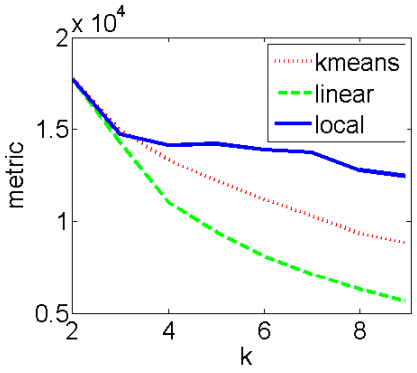
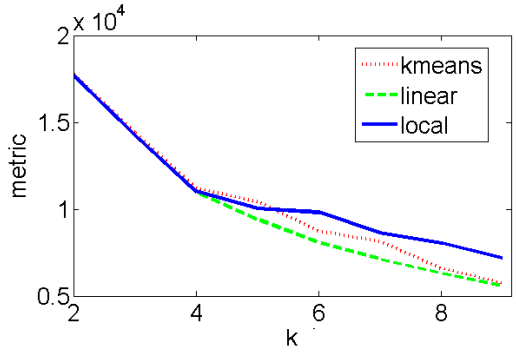


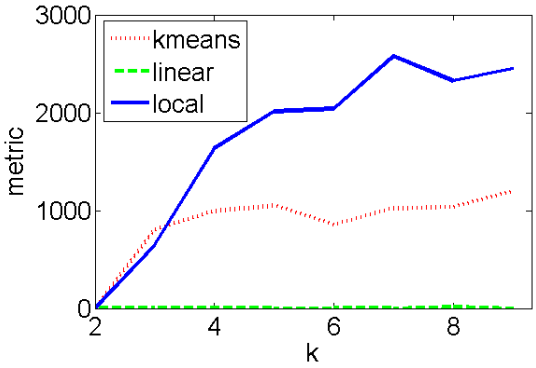
Figure 2. Minimum Metric CDF, k = 4



(a) Means

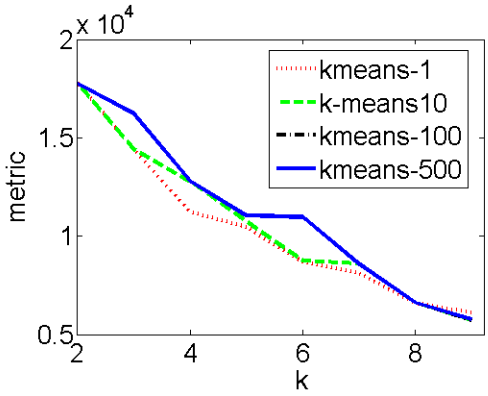


(b) Minimums

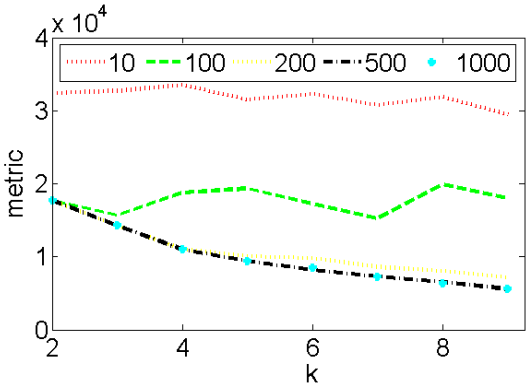


(c) Standard Deviations

Figure 3. Comparison between, K-means++ with 1 initialisation, Local Search with 200 Repetitions & Linear Program



(a) K-means++



(b) Local Search

Figure 4. K-means++ vs. Local Search

VI. Provable Bounds For Accuracy

Provable bounds for the accuracy of methods similar to the methods in this paper have the restriction that the approximation is constrained to being α-optimal with β*k*-medians. This is known as an (α,β)-approximation, the known approximation bounds are given in Table II.

TABLE II. Provable Bounds

Full Search	Optimal
Linear Program[1]	$(1+\epsilon, (1+1/\epsilon) * ln(n/\delta) * k)$ [7]
Local Search	$(1+\gamma, 3+5/\gamma)$ & $(1+5/\gamma, 3+\gamma)$ [15]
K-means++	$8(ln(k)+2)$ [8]

VII. Conclusion

In conclusion, the paper addresses the important issue of scalability for the placement of controllers. As the k-median problem is a known NP hard problem the scalability of the placement of controllers is linked directly to the complexity of the algorithms used to find a solution. It was found the complexity varied significantly between the methods discussed, causing the full search method to be infeasible for large scale networks, the linear programming method to be useful for networks of a limited size and the local search method and k-means++ the most scalable approaches. However, the scalability is traded off with the accuracy of the methods. It was found that although the local search method approaches the metric found by the more complex linear program the standard deviation varies considerably. The main reason for this could be the randomness involved in finding initial clusters, but using a more intelligent way of finding initial clusters increases the complexity as with the k-means++ method. This can also hinder the optimality of the algorithms as it was shown the k-means++ has a reduced variance in solutions, but the solutions are not necessarily as good.

Acknowledgment

This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/I028153/1]; Thales UK; and the University of Bristol. We would like to thank Thales UK Research & Technology for supporting this research through an EPSRC CASE Award

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69-74, 2008.

[2] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," *HotSDN'12*, 13 August 2012 2012.

[3] K. Jain and V. V. Vazirani, "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation," *Journal of the ACM (JACM)*, vol. 48, pp. 274-296, 2001.

[4] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on computing*, vol. 33, pp. 544-562, 2004.

[5] D. G. Luenberger, *Introduction to linear and nonlinear programming* vol. 28: Addison-Wesley Reading, MA, 1973.

[6] G. B. Dantzig, *Linear programming and extensions*: Princeton university press, 1998.

[7] J.-H. Lin and J. S. Vitter, "e-Approximations with minimum packing constraint violation," in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, 1992, pp. 771-782.

[8] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027-1035.

[9] D. B. Johnson, "Efficient Algorithms for Shortest Paths in Sparse Networks," *J. ACM*, vol. 24, pp. 1-13, 1977.

[10] S. S. Epp, *Discrete mathematics with applications*: Cengage Learning, 2010.

[11] Y. Zhang, "Solving large-scale linear programs by interior-point methods under the Matlab∗ Environment†," *Optimization Methods and Software*, vol. 10, pp. 1-31, 1998.

[12] K. M. Anstreicher, "Linear programming in O ([n3/ln n] L) operations," *SIAM Journal on Optimization*, vol. 9, pp. 803-812, 1999.

[13] D. Arthur and S. Vassilvitskii, "How Slow is the k-means Method?," in *Proceedings of the twenty-second annual symposium on Computational geometry*, 2006, pp. 144-153.

[14] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, pp. 1765-1775, 2011.

[15] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," *Journal of algorithms*, vol. 37, pp. 146-188, 2000.

[1] given any $\varepsilon > 0$ the algorithm can produce a metric within a factor of $(1 + \varepsilon)$, with a probability at least $1-\delta$ $(0 < \delta < 1)$

AUTHORS' BACKGROUND

Your Name	Title*	Research Field	Personal website
Joseph Hollinghurst	PhD candidate	Electrical and electronic engineering and Mathematics	n/a
Ayalvadi Ganesh	Lecturer	Mathematics	n/a
Timothy Baugé	n/a	Communications Networks	n/a

*This form helps us to understand your paper better, the form itself will not be published.

*Title can be chosen from: master student, Phd candidate, assistant professor, lecture, senior lecture, associate professor, full professor