Chapter 6

Solution improvement

In this chapter, the refined model of the previous chapter is solved using heuristic techniques, improving computational performance while being as accurate as feasibly possible. Heuristics based on elementary rules and reduced model solving are tested along with an existing heuristic known as BCA before analysing segmentation based methods. Segmentation is done through the k-means clustering algorithm, showing improved performance in terms of accuracy, computational complexity and memory requirements relative to the other heuristics tested.

6.1 Motivation

As stated in chapter 3, heuristics are used to reduce the complexity of a problem by using some insight into their underlying structure. This enables them to arrive at feasible solutions to extremely complex problems within a reasonable amount of time. With regards to the PON planning problem, these heuristics can typically be divided into two groups - those based on using an approximation or relaxation of some aspect of the total problem and those based on dividing the problem into smaller sub-problems before solving.

The first type is realized through solving a reduced version of the model which contains fewer constraints. This solution can then be used as a *good* basis for the solution of the full model, allowing a large number of weak solution candidates to be discarded at the onset of the branch and bound procedure. Since the number of solution candidates are reduced, computational performance increases.

The second type of heuristic usually divides the problem by segmenting the search space through clustering or cuts. This creates a number of smaller problems that can be solved individually and combined in some way to give a full solution. For problems where computational effort increases exponentially with the number of input variables, a linear increase in sub-problems can result in an exponential decrease in computation time.

6.2 Testing methodology

6.2.1 Input data

Unless otherwise stated, all tests use the *SubNet*, *MedNet* and *CityNet* datasets and all parameters as used in the complete model test in section 5.4.6 contained in tables 5.3, 5.6 and 5.8. All tests will be conducted in such a manner as to ensure fair relative comparison with the complete model solutions of the refined model.

6.2.2 Result interpretation

Similar to the testing methodology of the refined model but with a change in emphasis, this chapter will focus on the following aspects to determine relative performance when conducting experiments.

- **Performance** - Since this chapter focusses on improving computational performance of the solutions introduced in the previous chapter, performance is of

utmost importance and is the main focus of all heuristic experiments. The numerical value associated with performance is solution time in seconds, although peak memory usage is also taken into account. Reasonable computation times are regarded to be less than one hour and the goal of this chapter is to arrive at a reasonable solution to *HugeNet* in this allotted time.

- **Deployment cost** - Total deployment cost is used to determine the relative difference in accuracy between tests as it is the figure being optimized in the solution.
- **Topological accuracy** - As with the refined model, all solutions still have to exhibit topological accuracy to be considered feasible.

Now that a testing methodology is established, the different heuristic approaches can be evaluated, starting with an elementary approximation heuristic.

6.3 ELEM - Elementary heuristic

As a first step to arrive at a feasible solution in a very short time, an elementary heuristic is devised. It calculates a feasible solution using a basic algorithm that iteratively allocates ONUs to the closest splitter with spare capacity. This feasible solution can then be used as a stand-alone solution guess or used as a *warm start* for the solver. When a warm start is used, an additional preprocessing step to the branch and bound algorithm discards all solution candidates with higher cost than the warm start.

The elementary heuristic should give a ballpark figure of the accuracy of the most basic approximation heuristic and the sophistication required to arrive at a decent solution, allowing worthwhile improvements to be made in further iterations. For simplicity, this heuristic will henceforth be referred to as ELEM.

6.3.1 Algorithm

The algorithm starts by iterating through all ONUs $j \in \mathbf{U}$ and determining the closest splitter $b \in \mathbf{S}$ to each ONU by iterating through all paths between the j-th ONU and each splitter $i \in \mathbf{S}$. The algorithm then makes an additional check to determine if a slightly longer path does not share more edges with an already used path up to a specified threshold. If this splitter b still has some potential capacity left, the ONU is allocated to it and the remaining capacity of b is reduced accordingly. Otherwise, the process repeats until a viable splitter is found. The path used to connect the ONU j to the splitter b is then marked as used.

With a function length(x) giving the length of path x and a function sharedEdges(x) giving the number of edges shared with other used paths, the heuristic can be defined as follows in algorithm 6.1. $\mathbf{P}(i, j)$ is the set of all paths between ONU $j \in \mathbf{U}$ and splitter $i \in \mathbf{S}$ and is equivalent to $\mathbf{P}(k_{ij}^{SP-ONU})$.

When all ONUs have been allocated to splitters, the shortest paths between used splitters and the CO are marked as used. Finally, all splitter types are set to the smallest capacity type that can still serve all its allocated ONUs.

Recall that fiber duct sharing allows for lower overall deployment cost due to the sharing of more edges. To allow for this phenomena, a threshold of 1.1 is used, allowing for the usage of a 10 % longer path in the event that it shares more edges. This value greatly depends on the difference between trenching and fiber costs and the value shown is only a very rough estimate. Further studies regarding the exact optimal value is beyond the scope of this heuristic.

6.3.2 Methodology

To test the effectiveness of ELEM, two aspects will be scrutinized. Firstly, its performance as a solution guess and secondly, its effectiveness when used as a warm start for the full model. Both tests will be conducted on the complete refined model using

Algorithm 6.1 ELEM

$capacity_i \leftarrow \kappa^S_{ONU}, \quad \forall i \in \mathbf{S}$
$B_p \leftarrow 0$ ▷ Best path
for all ONUs j in **U do**
  $B_s \leftarrow 0$ ▷ Best splitter
  for all splitters i in **S do**
    if $(capacity_i > 0)$ **then**
      for all paths p in $\mathbf{P}(i, j)$ **do**
        if $\left[\text{length}(p) < \text{length}(B_p)\right]$ **or** $\left[\text{length}(p) < 1.1 * \text{length}(B_p)\right.$ **and** $\text{sharedEdges}(p) > \text{sharedEdges}(B_p)\left.\right]$ **then**
          $B_p \leftarrow p$
          $B_s \leftarrow i$
        end if
      end for
    end if
  end for
  Set B_p and B_s used
  $capacity_{B_s} \leftarrow capacity_{B_s} - 1$
end for

the *SubNet*, *CityNet* and *MedNet* datasets and the parameters as set out in the previous chapter. This will allow direct comparison with the complete model solution in section 5.4.6.

6.3.3 Result analysis

The total deployment cost results of ELEM as a solution guess are given in figure 6.1 while the complete numerical results are given in table 6.1. It shows that as far as computational performance is concerned, the heuristic takes only 1.26 s on average to compute, a 99.96 % reduction in computation time. As a solution guess, ELEM does fairly well, showing a 23.48 % increase in deployment cost relative to the complete

model results. Since the complete model results all have optimality gaps greater than zero (due to the imposed time limit), it can be said that the heuristic is worst case 50.25 % from optimal in this case, although actual performance is likely closer.

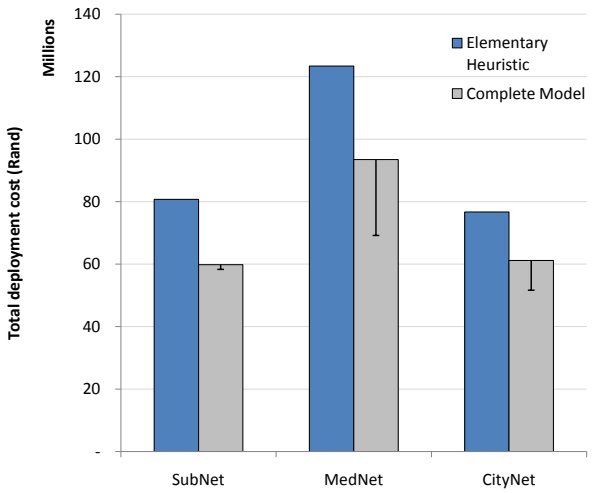


Figure 6.1: Total deployment cost comparison between complete model and ELEM

As for the numerical results, it is clear that ELEM uses far more splitters than the complete model solution, using more than twice the number in each dataset scenario. Given the algorithm it is to be expected, since there is no cost associated with deploying additional splitters.

Table 6.1: ELEM results: Solution guess

		Dataset		
Result	Unit	*MedNet*	*SubNet*	*CityNet*
Numerical				
Total cost	R (mil)	123.41	80.74	76.72
Cost per ONU	R (1000)	116.86	74.83	39.32
Splitters used		127	108	224
Avg. split ratio		8.31	9.99	8.71
Computational				
Time to solve	s	1.87	0.55	1.38

The results of the complete model solved using ELEM as a warm start feasible solution are given in table 6.2. Since a time limit of 1 hour was specified, relative performance

is deduced from the differences in optimality gap, since a lower optimality gap after a fixed period indicates faster convergence to optimal and likely better computational performance.

From the results, it is evident that the warm start barely has an effect on computational performance, with *SubNet* showing the largest improvement amounting to a solution 1.11 % closer to optimal. *CityNet* showed no improvement at all, registering the same optimality gap with or without the warm start.

Table 6.2: ELEM results: Warm start

		Dataset		
Result	Unit	*MedNet*	*SubNet*	*CityNet*
Numerical				
Total cost	R (mil)	93.46	59.80	61.16
Cost per ONU	R (1000)	88.50	55.43	31.35
Splitters used		58	48	94
Avg. split ratio		18.21	22.48	20.76
Optimality gap	%	24.87	2.30	15.54
Computational				
Setup time	s	56.4	14.0	36.6
Time to solve	s	3,672	3,622	3,647
CPU time	s	6,871	9,528	7,101
Peak memory	MiB	17,080	11,604	13,286

In conclusion, even though ELEM shows great computational performance as a solution guess, large deviations from optimal does not allow its use as a practical stand alone solver. When used as a warm start, the performance increase might warrant its use, but only due to its negligible computation time. Following this, a more sophisticated heuristic might prove to have better characteristics.

6.4 Reduced model input

Another strategy to design an approximation heuristic is to use a relaxed version of the model to be solved as a starting point. By relaxing the constraints on the model, complexity is reduced, improving computational performance. This reduced model can then be used as a starting point or a warm start for the full model, once again allowing the pruning of entire branches of solution candidates as a preprocessing step to the branch and bound algorithm.

Since the solution of the reduced model does not include all the variables contained within the complete model, the remaining variables will have to be allocated accordingly. This assignment of values is heuristic, where smart assignments will lead to improved feasible solutions.

6.4.1 Reduced model

In the previous chapter it is evident that the inclusion of fiber duct sharing negatively affected computational performance to a large extent. By removing the constraints that allow optimization of fiber duct sharing, a relaxed model can be constructed and solved to optimality. This almost reduces the complete model to the complexity of the baseline model described in the previous chapter.

Therefore, the reduced model can be defined as the refined model given in section 5.2.5, but without constraint 5.76 and with a new objective function as given below:

Minimize

$$\begin{aligned} C_{total} =& C_{OLT}\Phi + \sum_{t\in\mathbf{T}} \tilde{c}_t^{SP} + \tilde{c}^{ONU} + \sum_{\ell\in\mathbf{C}} \sum_{i\in\mathbf{S}} \sum_{p\in\mathbf{P}(k_{i\ell}^{CO-SP})} \sigma_p^{CO-SP} y_p \\ &+ \sum_{i\in\mathbf{S}} \sum_{j\in\mathbf{U}} \sum_{p\in\mathbf{P}(k_{ij}^{SP-ONU})} \sigma_p^{SP-ONU} y_p \end{aligned} \tag{6.1}$$

Since constraint 5.76 evaluates to a number of constraints equal to $|\mathbf{E}|$, with each con-

straint having $|\mathbf{K}||\mathbf{P}(k,e)|$ terms in the left side, the exclusion results in a reduction in the computational effort required to solve the model *.

Even though this model does not include fiber duct sharing, the solution is altered to become a feasible solution for the complete model, accounting for all other constraints of the model. To accomplish this, edge costs are included in the solution of the reduced model by marking all edges contained in used paths as used themselves. Then, the feasible solution is used as a warm start for CPLEX when solving the complete model.

6.4.2 Methodology

As with ELEM, the reduced model heuristic is tested both as a solution guess and as a warm start for the complete model. To prevent bias to input data, once again all three smaller datasets are tested, using the parameters from the previous chapter. Since the reduced model is already an approximation and computational performance is preferred at this stage, the CPLEX solver is set to quit if a feasible solution is found with an optimality gap below 0.5 %.

6.4.3 Result analysis

Figure 6.2 shows the total deployment cost of the complete model from the previous chapter relative to the solution guess from the reduced model. Numerical results of the solution guess are given in table 6.3. The reduced model as a solution guess is evidently superior to ELEM in terms of accuracy, showing similar to and even lower deployment cost as could be managed by the solution to the complete model.

Given the complete model's optimality gaps, the reduced model shows deployment costs for the *MedNet*, *SubNet* and *CityNet* datasets with worst case deviations from optimal of 20.04 %, 3.5 % and 11.32 % respectively. This effectively gives better answers

*To illustrate, in the case of *SubNet*, the exclusion of path sharing results in a 1.5 % reduction in the number of variables and a 87.5 % reduction in non zero elements in the MIP matrix.

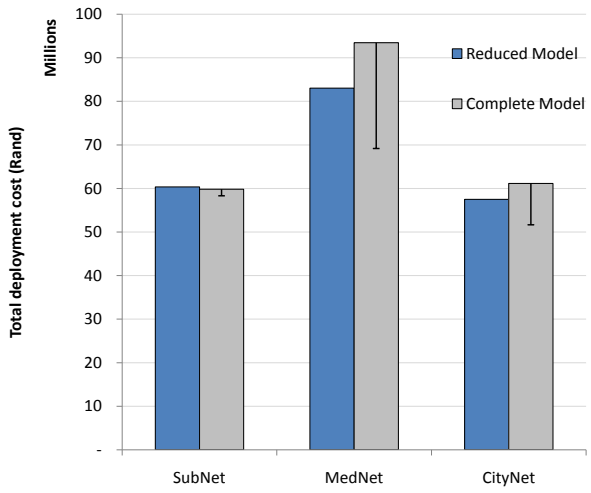


Figure 6.2: Total deployment cost comparison between complete model and reduced model solution guess

in under 3 minutes for each dataset than the initial complete model solution did after an hour. Furthermore, the number of splitters are lower in the reduced model solution, showing that the optimal solution for the complete model likely contains far less splitters.

Memory usage to solve the reduced model is also reduced significantly, allowing larger datasets to be solved with the same amount of resources. Unfortunately, it is still excessive, easily reaching 3 GiB for the smallest dataset, ensuring that solving the *HugeNet* dataset would be infeasible.

When looking at the results of the reduced model solution guess, it is evident that when this solution is used as a warm start, the complete model result should be *at least as good*, already showing better performance. In accordance with this logic, the numerical results of the warm start, given in table 6.4, show total deployment costs that are 11.22 %, 0.03 % and 6.19 % lower than the initial (no warm start) solution to the complete model for datasets *MedNet*, *SubNet* and *CityNet* respectively.

In conclusion, the reduced model provides a very good initial guess with a 96 % reduction in computation time relative to the complete model solution. When used as a warm start, the reduced model drastically improves the performance of the com-

Table 6.3: Reduced model results: Solution guess

		Dataset		
Result	Unit	*MedNet*	*SubNet*	*CityNet*
Numerical				
Total cost	R (mil)	83.04	60.37	57.51
Cost per ONU	R (1000)	78.63	55.95	29.47
Splitters used		40	50	84
Avg. split ratio		26.40	21.58	23.23
Optimality gap	%	0.50	0.15	0.24
Computational				
Setup time	s	57.8	1410	36.5
Time to solve	s	106.8	20.2	188.8
CPU time	s	313	62	584
Peak memory	MiB	10,307	3,031	7,240

Table 6.4: Reduced model results: Warm start

		Dataset		
Result	Unit	*MedNet*	*SubNet*	*CityNet*
Numerical				
Total cost	R (mil)	82.97	59.81	57.38
Cost per ONU	R (1000)	78.57	55.43	29.41
Splitters used		40	48	85
Avg. split ratio		26.40	22.48	22.95
Optimality gap	%	15.38	2.39	10.25
Computational				
Setup time	s	56.6	14.2	37.6
Time to solve	s	3,914	3,628	3,783
CPU time	s	7,391	9,309	7,564
Peak memory	MiB	12,813	11,608	12,767

plete model, although computation times are still prohibitive so far as to make solving the *HugeNet* dataset infeasible. Now that a good approximation heuristic has been constructed and tested, the results are compared to a known heuristic - the Branch Contracting Algorithm (BCA).

6.5 Modified BCA

The Branch Contracting Algorithm (BCA) is a general heuristic proposed by Mitcsenkov, Paksy and Cinkler [12, 48]. It is designed specifically for the PON planning problem and the authors claim good accuracy figures of 20 % to optimal as well as excellent computational performance relative to a lower bound ILP. BCA has no steps involving solving an LP like the reduced model or warm starts, and should ensure polynomial time complexity.

As was stated in chapter 3, another well known heuristic, referred to as RARA, was proposed by Li and Shen in 2009 [2]. This heuristic showed promise by employing sophisticated allocation techniques and a simulated annealing step to improve known feasible solutions. In [12] however, the authors claim to have implemented the complex RARA heuristic, with it showing similar performance to BCA while having worse computational performance due to the many iterations involved. Therefore, BCA is chosen for comparative purposes, with the RARA heuristic expected to have similar accuracy.

6.5.1 Algorithm

The BCA optimizes for fiber duct sharing through four steps, taking advantage of the fact that a PON plan has a tree topology. The algorithm can be described as [12]:

- **Step 1 (Initialization)** - Construct a tree $\mathcal{T} \subset \mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ from a graph $\mathcal{G}$ with edges $\mathcal{E}$ and vertices $\mathcal{V}$ by connecting all ONUs to the CO through the shortest paths in the graph.
- **Step 2 (Group Formulation)** - With N denoting the number of unconnected ONUs, pick a random leaf of tree $\mathcal{T}$ and move up toward the root at the CO, one node (or vertex) at a time. At each step, determine the amount of ONUs, n, in the subtree, with the current node at its root. If this number exceeds a threshold, $\mathcal{Q}$, all

ONUs in the subtree form a new group and are removed from the tree and from the unconnected ONUs list. Repeat this step until all ONUs are grouped.

- **Step 3 (Splitter Allocation)** - For each group in step 2 above, determine the splitter location with the minimum total distance from all ONUs in the group. Distances are summarized using the Bellman-Ford algorithm [54, 55].
- **Step 4 (Connection Establishment)** - Connect all ONUs to their assigned splitters using the shortest paths. Connect the splitters to the CO by constructing a Steiner tree rooted at the CO with splitters as Steiner nodes. The Steiner tree is then solved using a modified version of the distance network heuristic (DNH) proposed in [61].

For further illustration, a flowchart of the BCA is given in figure 6.3 [12].

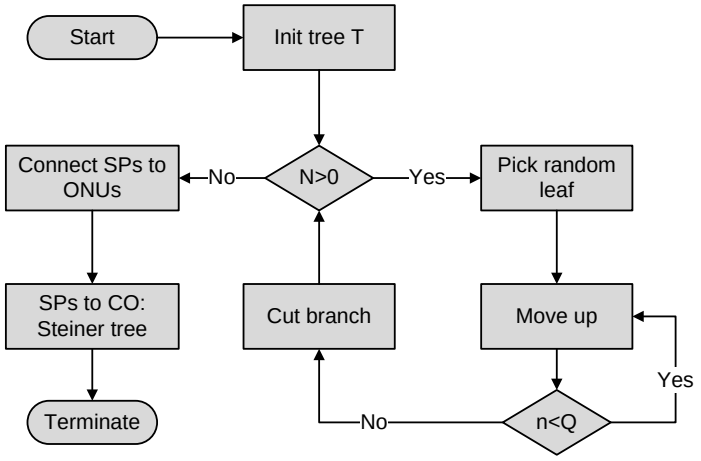


Figure 6.3: Flowchart of the Branch Contracting Algorithm (BCA)

It should be noted that the algorithm does not allow for additional refinements included in the complete model such as multiple COs, coverage and differential distance network constraints. It is however possible to modify the original algorithm to allow for these refinements, although this is outside of the scope of this dissertation.

6.5.2 Modifications to algorithm

Two modifications are made to BCA. Firstly, in step 3, the Bellman-Ford algorithm is replaced with Dijkstra's algorithm since it is faster and neither link cost nor link distances can be negative for the PON planning problem as defined in chapter 3. Secondly, and most notably, the Steiner tree implementation to connect the CO to splitters is removed. This is due to insufficient information concerning how the DNH heuristic was modified to solve the Steiner tree as defined in the algorithm.

To avoid making biased claims after modifying BCA, tests are run with two replacements for step 4. The first is to simply connect the splitters to the CO through their shortest paths, doing opportunistic fiber duct sharing when the shortest paths share a common edge (BCAMod). Since this might result in a higher deployment cost and hence decrease the accuracy of BCA, a second test is run without any fiber connecting splitters to the CO (BCAModNoF), resulting in a lower bound. Therefore, the true performance of the DNH implementation should lie somewhere between these two replacements.

Lastly, there is insufficient information concerning the formulation of the $\mathcal{Q}$ factor mentioned in step 2 since the authors simply state that it is a function of the maximum splitter ratio K. The details of this mythical $\mathcal{Q}$ factor can be avoided though, by enumerating all valid values for $\mathcal{Q}$ and running the algorithm for each value. This gives BCA the benefit of the doubt as the best possible $\mathcal{Q}$ is found for each dataset.

6.5.3 Methodology

MedNet, *SubNet* and *CityNet* are solved using BCA with values for $\mathcal{Q}$ ranging from 1 to κ^{S}_{ONU}, the theoretical maximum. Also, since BCA is randomly initialized, each iteration is repeated 10 times to avoid unfair interpretation of BCA performance, saving only the solution of the iteration with the lowest deployment cost. Standard deviations are calculated for all random initializations at a given $\mathcal{Q}$ value to determine practical

performance of BCA. The same parameters are used as defined in the complete model solution in the previous chapter where applicable.

Due to the infeasible memory requirements to generate the sets and subsets to build the tree $\mathcal{T}$ for the *HugeNet* dataset, it will not be solved at this stage.

6.5.4 Result analysis

A comparison in total deployment cost between the complete model solution and the modified BCAs is given in figure 6.4 while numerical results for BCAMod and BCAModNoF can be found in tables 6.5 and 6.6 respectively. Concerning deployment cost, BCA still has some way to go, displaying minimum deviations from the complete model solution of up to 62.98 % for *MedNet* in the case of BCAMod and up to 41.78 % for *SubNet* in the case of BCAModNoF. Looking at average worst case deviation from optimal, BCAModNoF leads with 34.38 % while BCAMod struggles with 44.26%. With the original BCA lying between BCAMod and BCAModNoF, performance is on average far from the claimed 20 % from optimal, even when run with no connecting fiber between CO and splitters.

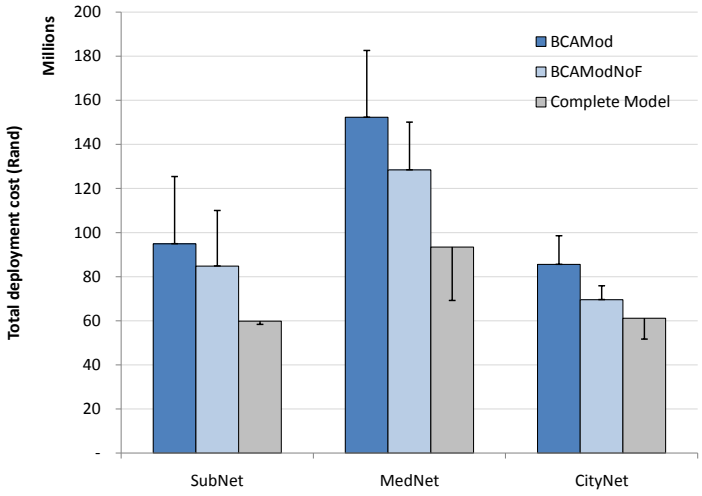


Figure 6.4: Total deployment cost comparison between complete model, BCAMod and BCAModNoF

Numerical results for BCAMod shows excellent computational performance, with single iterations taking less than two seconds to compute. Unfortunately, the large standard deviations between random initializations mean that single run performance can vary wildly, as seen in the error bars of figure 6.4. Since the computation time of BCAMod is so low however, it would be feasible to run the algorithm 10 or 100 times, saving the solution with the lowest deployment cost.

Table 6.5: BCA Results: Shortest path SP-CO fiber (BCAMod)

		Dataset		
Result	Unit	*MedNet*	*SubNet*	*CityNet*
Numerical				
Total cost	R (mil)	152.3	94.93	85.57
Cost per ONU	R (1000)	144.2	87.98	43.86
Splitters used		76	34	82
Avg. split ratio		13.89	31.74	23.79
Best Q factor		14	32	24
Standard deviation	R (mil)	11.24	9.39	4.08
Computational				
Time to solve	s	2.072	0.655	1.503
Peak memory	MiB	8,117	2,307	5,187

BCAModNoF showed similar performance to BCAMod, since the extra allocation step is negligible in terms of computational complexity. Memory usage, while lower than that of the complete model solution, is still high, with values ranging from 2.3 GiB for *SubNet* to 8.1 GiB for *MedNet*. This is largely due to the memory requirements of the sets and subsets needed to construct and traverse the tree $\mathcal{T}$.

From the results it is evident that BCA may be data dependent, with clustered datasets displaying worse than claimed performance. Uniformly spaced datasets should show improved performance with BCA though, likely to the point of the claimed 20 %, since the probability of close proximity groups is higher in non-clustered graphs. For example, if a group of size $\mathcal{Q}$ is formed from two clusters with the same root, one of size 2 and the other of size $\mathcal{Q} - 2$, with the clusters spaced far apart, the group will be allocated to a single splitter. This will ensure higher than necessary connection costs for

Table 6.6: BCA Results: No SP-CO fiber (BCAModNoF)

		Dataset		
Result	Unit	*MedNet*	*SubNet*	*CityNet*
Numerical				
Total cost	R (mil)	128.4	84.82	69.58
Cost per ONU	R (1000)	121.6	78.61	35.67
Splitters used		76	45	140
Avg. split ratio		13.89	23.98	13.94
Best Q factor		14	24	14
Standard deviation	R (mil)	6.50	8.02	1.89
Computational				
Time to solve	s	2.084	0.662	1.509
Peak memory	MiB	8,100	2,311	5,187

the small cluster.

Now that the approximation heuristics have been tested, a heuristic based on problem division, or segmentation, will be investigated for possible performance improvements.

6.6 Segmentation

As was stated in the introduction, problem decomposition could greatly reduce the required computational effort, depending on the underlying structure of the problem. Unfortunately, the division of the original problem can break interdependencies between variables contained in different sub-problems, resulting in a less optimal solution. In the case of the PON planning problem, ONUs contained in one sub-problem can not be allocated to a splitter from another sub-problem, even if they might be allocated to each other in the optimal solution of the full problem. Also, since each problem has different input data, problem division is problem specific. Therefore, the challenge is to minimize these sub-optimal allocation occurrences through intelligent problem division.

6.6.1 Random-cut segmentation

The simplest type of problem division is the technique known as random-cut segmentation. In the case of the PON planning problem, it works on the principle that the input map is segmented by means of rotating a radius line 360 degrees around the graph mid-point, making a cut each time the amount of nodes in the current segment exceeds a specified threshold. Since the first cut is made randomly from the mid-point in any direction, this segmentation method will result in different segments for each initialization. Figure 6.5 illustrates the principle in greater detail

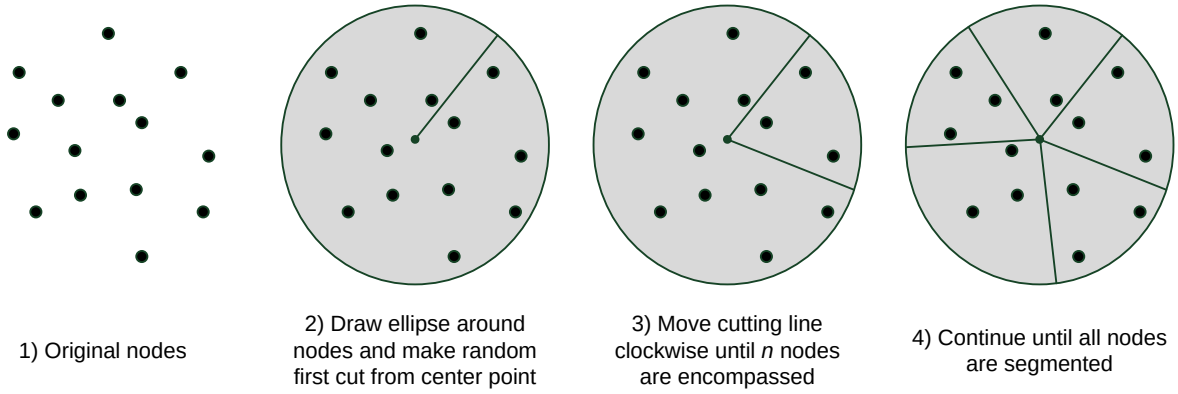


Figure 6.5: Steps of the random-cut segmentation method

Since the PON planning problem data is inherently clustered into neighbourhoods etc. though, this method will likely cut through some naturally occurring clusters, inflating total deployment cost due to ineffective ONU-splitter allocations. To combat this, a way to identify the natural clusters is needed.

6.6.2 *k*-means clustering

In 1982, Lloyd [62] published a way to segment data by sequentially constructing a fixed number of clusters so that the total distance between the centroid of the cluster and each member is minimal. This is known as the *k*-means clustering algorithm and is given below in algorithm 6.2.

Though other more sophisticated clustering techniques based on density (OPTICS [63]) and data distribution (expectation-maximization [64]) exist, the *k*-means clustering algorithm is especially notable due to its tendency to construct equi-sized clusters. This ensures computational complexity is distributed equally across all clusters, maximizing the performance benefit from segmentation.

Algorithm 6.2 *k*-means clustering

1: Generate k initial means $m_i^{(t)}$ randomly
2: **repeat**
3: **for** $i \leftarrow 0, k-1$ **do**
4: Associate all observations $\mathbf{x}_j$ closest to mean $m_i^{(t)}$ with cluster $S_i^{(t)}$
5: Calculate new centroids, $m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$
6: **end for**
7: **until** $S_i^{(t)} = S_i^{(t-1)}, \quad \forall i \in \{0, \ldots, k-1\}$ ▷ No clusters have changed this iteration

One drawback of *k*-means clustering however is the necessity to determine the value of *k* beforehand. Since computational complexity will largely be determined by the cluster size and not the number of clusters, a good value for *k* will need to be determined through experimentation.

6.6.3 Implementation remarks

Since the algorithm has no sense of the network constraints of the PON planning problem, it is possible to have a cluster that cannot be solved. This occurs when a cluster has no splitters or too few splitters for the amount of ONUs to connect. To ensure a feasible solution exists for each cluster, *valid clusters* are built from the output of the *k*-means algorithm. These are clusters containing enough splitters to serve all ONUs contained within the cluster. With the minimum split ratio defined as the capacity of a splitter, κ_{ONU}^{S}, we can define:

Definition 6.1 (Valid cluster)**.** *Given a set of splitters* $\mathbf{S} \neq \varnothing$*, a set of ONUs* $\mathbf{U} \neq \varnothing$ *and a*

minimum split ratio κ^S_{ONU}*, a set* $\mathbf{L} \subseteq \mathbf{U} \cup \mathbf{S}, \mathbf{L} \neq \varnothing$ *is said to be a* valid cluster *only if*

$$\kappa^S_{ONU}|\mathbf{L} \cap \mathbf{S}| \geq |\mathbf{L} \cap \mathbf{U}|. \tag{6.2}$$

For simplicity, if a cluster $\mathbf{L}_i$ is invalid, it is combined with the next cluster and emptied, with $\mathbf{L}_{i+1} = \mathbf{L}_{i+1} \cup \mathbf{L}_i, \forall i, 0 < i \leq k$. This process is repeated iteratively for each i until all clusters are valid. Whereas the number of requested clusters is k, the number of *valid clusters* is denoted by k_{valid}.

Finally, the individual sub-problem solutions are combined to form the final solution to the complete model. Since the PON planning problem has an underlying tree structure, the sub-problems form sub-trees that are connected to the same root (the CO). Therefore, this combination step consists of simply adding the sub-trees together and recalculating the objective function.

6.6.4 Methodology

Four tests will be run for every dataset, including *HugeNet*, with cluster sizes of 50, 100, 200 and 400. Table 6.7 translates cluster sizes for each dataset into appropriate values for k by dividing the total number of ONUs and splitters by the specified cluster size.

It should be noted that constraints that can not be divided among sub-problems are those governing coverage and economies of scale. The cost benefits of EOS can be calculated manually after the solution is generated. For comparative purposes though, the model used is the exact complete refined model as specified in the previous chapter in section 5.2, with all parameters as used in section 5.4.6. The time limit for the CPLEX solver is reduced to 20 min for each sub-problem to avoid convergence issues skewing the results.

Additionally, the time to compute the k clusters is recorded, providing insight into the additional overhead created with this segmentation method.

Table 6.7: Values of k for the k-means algorithm based on cluster sizes

	Dataset			
Cluster size	*MedNet*	*SubNet*	*CityNet*	*HugeNet*
50	29	25	44	117
100	15	12	22	59
200	7	6	11	29
400	4	3	5	15

6.6.5 Result analysis

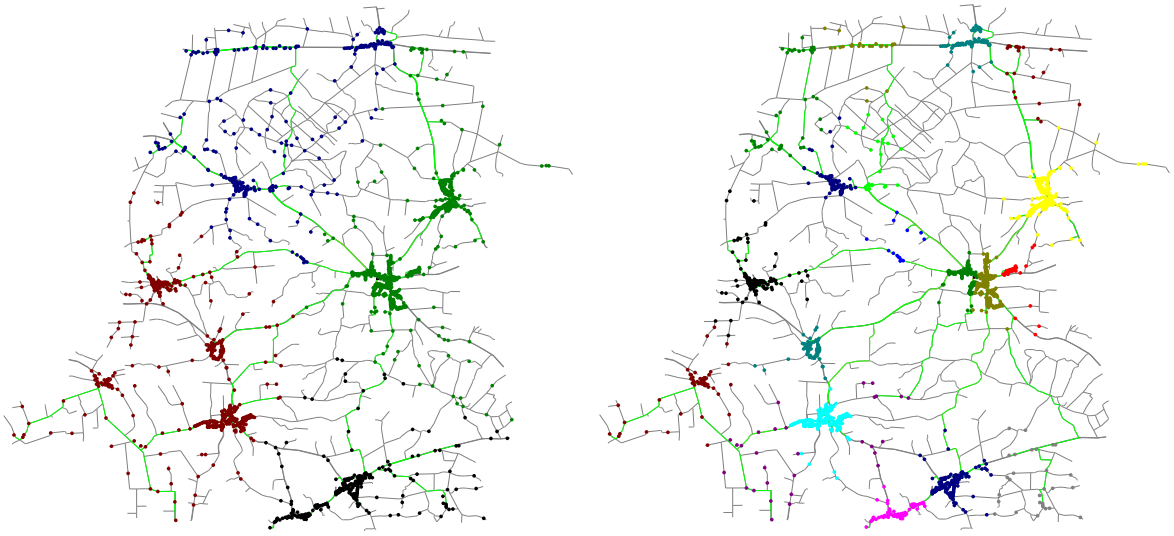
Figure 6.6 shows two clusterings of *MedNet* with $k = 4$ and $k = 29$, with each cluster colour-coded. In 6.6a, it is evident that the clusters are similar in size, with no segmentation cuts made through a natural cluster, as expected. As the number of clusters are increased, the data becomes over-segmented, with multiple clusters being identified inside a single natural cluster. With $k = 29$, the number of valid clusters also start to decrease, with only 21 being calculated.

The values for k_{valid} is given in table 6.8, where values with an asterisk indicating where $k \neq k_{valid}$. The data indicates that as cluster sizes are reduced to 50 or below (with the exception of *HugeNet* at a cluster size of 100), some computed clusters need to be grouped together to become valid. This is likely due to over-segmentation as well, with clusters suddenly being constructed from outlying nodes that are usually grouped with larger natural clusters.

Table 6.8: Valid clusters computed for different values of k

	Dataset			
Cluster size	*MedNet*	*SubNet*	*CityNet*	*HugeNet*
50	*21	*24	44	*105
100	15	12	22	*53
200	7	6	11	29
400	4	3	5	15

The effects of over-segmentation is evident in the total deployment cost given in figure 6.7, with a larger deviation as cluster sizes are set to 100 or below. Even so, overall



(a) $k = 4$, $k_{valid} = 4$ (b) $k = 29$, $k_{valid} = 21$

Figure 6.6: Clustering of *MedNet* with different values of k

performance is very promising, with figures across the board showing worst case optimality gaps of less than 20 %, averaging 12.92 %. The 400 cluster size test in particular shows an impressive average performance of 9.24 % to optimal, with the 200, 100 and 50 cluster size tests trailing with 10.74 %, 13.71 % and 17.98 % respectively.

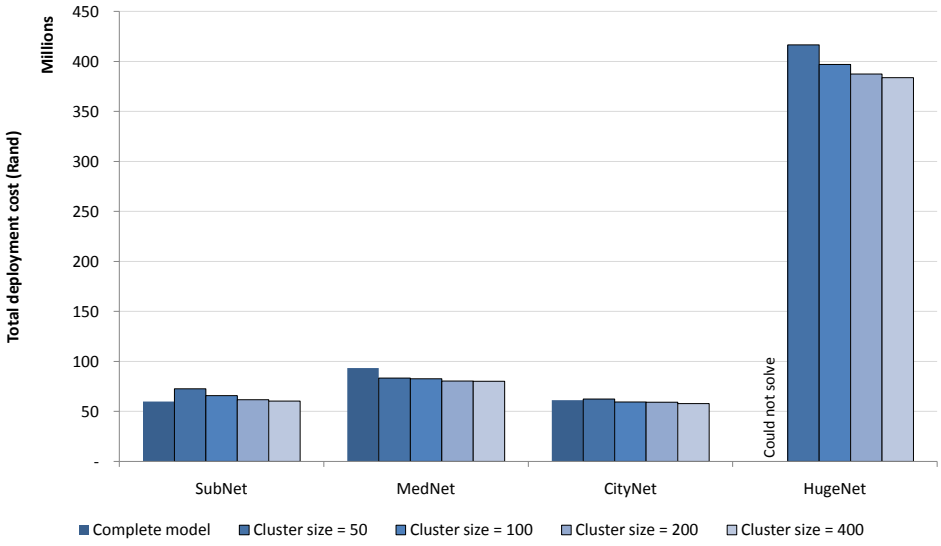


Figure 6.7: Total deployment cost of the segmented model with cluster sizes between 50 and 400

Likewise, computational performance shows impressive scaling in figure 6.8, with solution times reduced by 99.04 %, 97.98 % and 95.43 % for the 50, 100 and 200 cluster size tests respectively, relative to the complete model solution.

The 400 cluster size test sees solution times increase with *MedNet* and *CityNet*, largely due to the overall increased time limit, with each run limited to 20 minutes. However, these tests are also accompanied with reduced deployment costs, suggesting that overall performance per unit of time is still higher when segmentation is used. Conversely, *SubNet* shows a reduction of 71.51 % in computation time, as well as a lower deployment cost, further supporting the claim above.

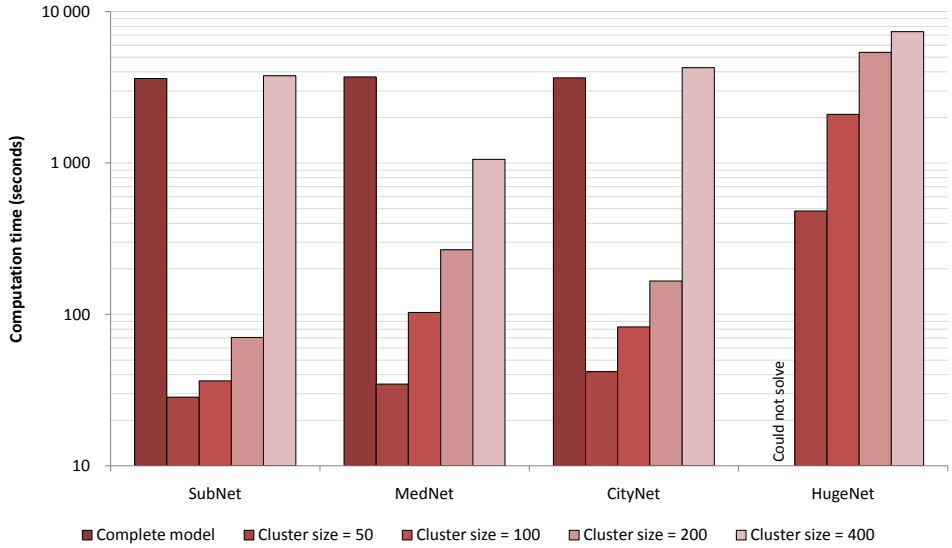


Figure 6.8: Logarithmic plot of the computation time of the segmented model with cluster sizes between 50 and 400

With regards to peak memory usage, the *k*-means clustering shows unrivalled performance over all previous heuristic methods, with markedly reduced memory resources required to solve the same problem as summarized in figure 6.9. This also allows *HugeNet* to be solved, with even the 400 cluster size test showing comfortable memory usage of 12.2 GiB. When the cluster size is decreased to 50, the resource hungry *HugeNet* only requires 1.6 GiB of memory, less than found on a standard consumer PC. As an additional example, relative to the complete model solution, *MedNet* with a cluster size of 50 shows a 98.55 % reduction in peak memory usage. This indicates that

segmentation is an excellent way to decrease overall memory requirements.

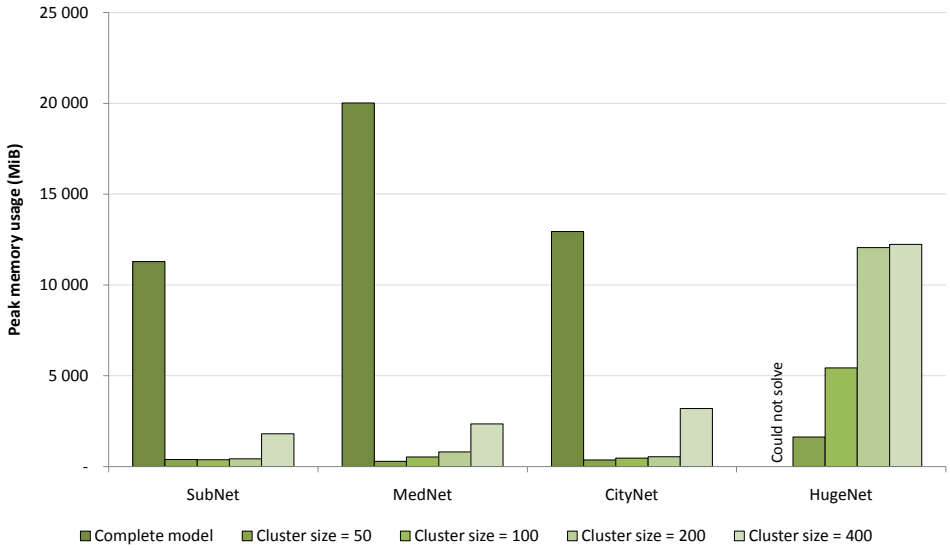


Figure 6.9: Peak memory usage when solving the segmented model with cluster sizes between 50 and 400

Additional overhead concerning the computation of clusters is negligible, with all clusters computed in less than a second for the three smaller datasets and in under 9 seconds for *HugeNet* in all tests.

Finally, and importantly, k-means clustering and segmentation allows *HugeNet* to be solved to some degree of optimality. Even though the exact optimality gap is not known since a lower bound could not be computed, results from the other datasets suggest a performance claim of 20 % to optimal should be conservative for larger cluster sizes. To complete figure 5.17 in the previous chapter, the topological output of *HugeNet*, with $k = 15$, is given in figure 6.10.

6.7 Solution improvement remarks

Recall from chapter 5 that not all paths were included in the model when fiber duct sharing was introduced, with only the shortest path being used. Since the complete model optimum, z^{CM}, is only an approximation of the true optimum, z^*, $z^{CM} \geq z^*$.

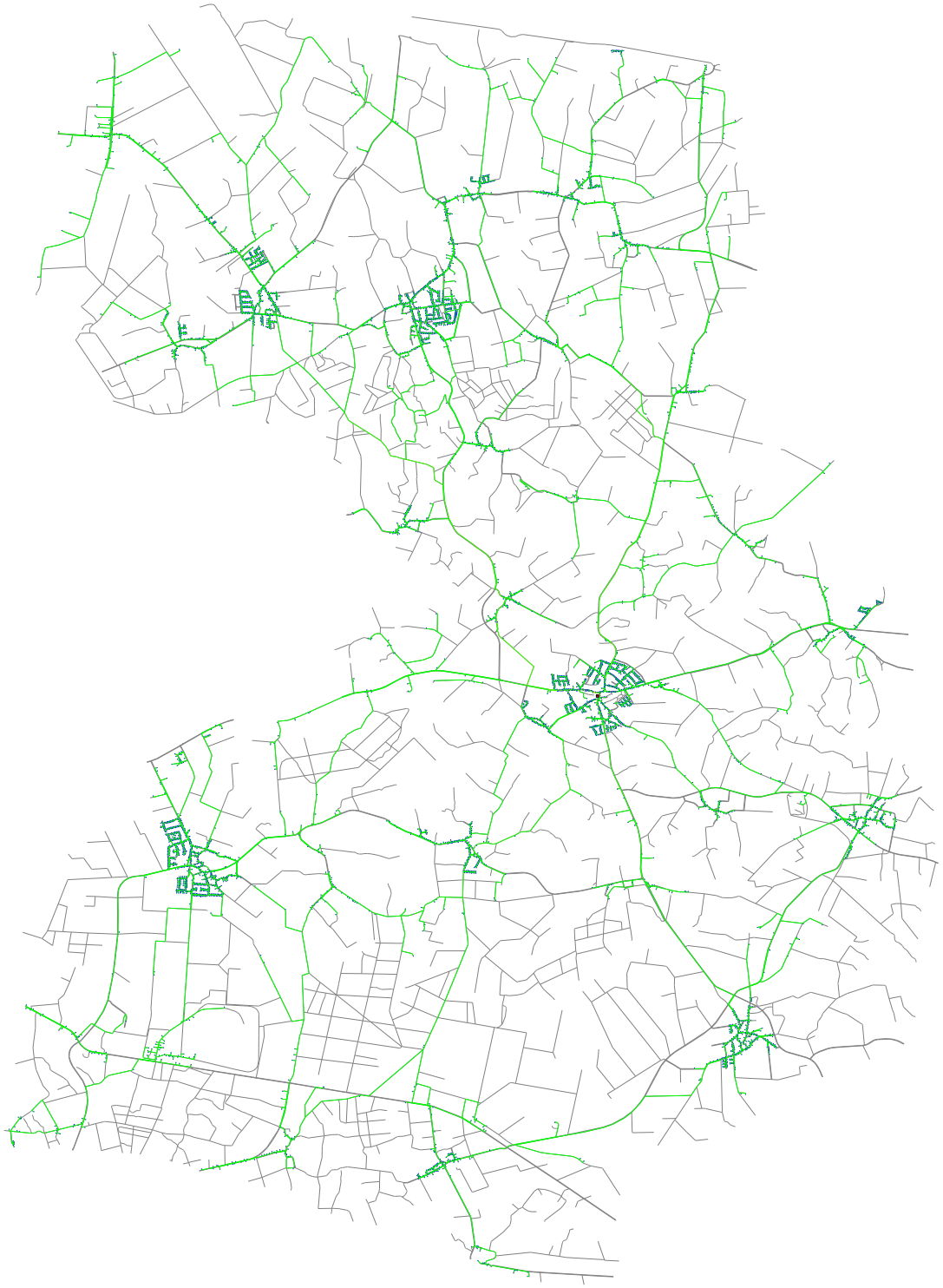


Figure 6.10: *HugeNet* solved, $k = 15$

Throughout this chapter, it should be noted that performance is compared to z^{CM} and *not* z^*. Therefore, the statement that the k-means clustering segmentation produced results 20 % from optimal does not imply any statement of truth concerning the deviation from the global optimal solution of the real world plan.

That said, the global optimal solution, z^*, or the solution of the refined model with input of $\mathbf{P}(k)$ containing all possible paths between commodity pair k, should be close enough to the results from this and the previous chapter so as to ensure all performance comparisons and optimality statements are of practical use.

6.8 Conclusion

In this chapter, the refined complete model solution from chapter 5 was improved through the use of heuristics. Firstly, a basic approximation heuristic appropriately called the elementary heuristic (ELEM) was tested, giving an extremely fast feasible solution but low accuracy of only 50.25 % from optimal. A more sophisticated heuristic based on a lower complexity model without fiber duct sharing constraints, or *reduced model*, was also tested, showing very good accuracy of lower than 20 % to optimal and good computational performance. Unfortunately, *HugeNet* could still not be solved due to excessive memory requirements.

The two approximation heuristics were also tested when used as warm starts to the complete refined model, providing a feasible solution as a good starting point. This gave a head start to the CPLEX solver, discarding entire branches of solution candidates, increasing computational performance. Unfortunately, using ELEM as a warm start showed negligible performance increases, with less than 1 % improvement seen across all datasets. The reduced model as a warm start did better, providing solutions that are on average 5.81 % closer to optimal.

In the comparison with BCA, the reduced model showed superior performance for the datasets tested, with BCA providing accuracies of between 34.38 % and 44.26 % from

optimal. It did however show very good computational performance, and takes its place between ELEM and the reduced model in terms of accuracy per time period. Unfortunately, BCA also suffered from high memory requirements, preventing *HugeNet* from being solved.

Lastly, another approach was followed to determine if the memory usage issue could be alleviated. This entailed segmentation through the use of the k-means clustering algorithm, showing similar performance in terms of accuracy along with reduced computational complexity relative to the reduced model, with average accuracy of less than 18 % to optimal. Most notably, memory requirements were drastically reduced when segmentation was used, with *MedNet* showing a 98.55 % reduction relative to the complete refined model when a cluster size of 50 was used. This development allowed *HugeNet* to finally be solved, taking only 8 minutes and consuming only 1.6 GiB of memory with the smallest cluster size.

The segmented model is therefore a good compromise between retaining the accuracy of the refined complete model in the previous chapter with drastically reduced computation times, even lower than the baseline model of section 5.4.1, surpassing all performance requirements.