

---

# Routing Tree Optimization for QoS Trade-off Analysis in Sensor Networks

---

Marcel Steine

Eindhoven University of Technology (TU/e)

National University of Singapore (NUS)

December 2007

**Supervisors:**

Ir. Rob Hoes (TU/e, NUS)

Prof. Twan Basten (TU/e)

Prof. Chen-Khong Tham (NUS)

Prof. Henk Corporaal (TU/e)

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Problem statement . . . . .	3
1.2	Research questions . . . . .	6
1.3	Overview . . . . .	6
<b>2</b>	<b>Optimizing the tree without trade-offs</b>	<b>7</b>
2.1	Introduction . . . . .	7
2.2	Algorithms . . . . .	7
2.3	Experiments . . . . .	8
2.4	Conclusion . . . . .	17
<b>3</b>	<b>Optimizing the tree with trade-offs</b>	<b>18</b>
3.1	Introduction . . . . .	18
3.2	Observations . . . . .	18
3.3	Checking the constraints . . . . .	19
3.4	Algorithms . . . . .	23
3.5	Experiments . . . . .	24
3.6	Conclusion . . . . .	27
<b>4</b>	<b>Distributed execution of the algorithms</b>	<b>28</b>
4.1	Introduction . . . . .	28
4.2	Algorithms . . . . .	28
4.3	Conclusion . . . . .	32
<b>5</b>	<b>Conclusion</b>	<b>33</b>
5.1	Future work . . . . .	34

# 1 Introduction

## 1.1 Problem statement

Research on wireless sensor networks (WSN) was originally motivated by military applications such as battlefield surveillance, but have been suggested for various other applications, like event detection applications and health care applications.

These WSN consist of a network of autonomous devices called (sensor) nodes. These nodes are usually equipped with one or more sensors, radio transceiver or other wireless communication, microcontroller and an energy source like a battery.

A node typically has multiple configurations, which are defined by a set of parameters like transmission power and sampling rate of the sensor. The system configuration is the combination of node configurations.

All nodes have a specified configuration and use their wireless communication to collaborate and fulfill the tasks of the application running on the WSN by sending their data (collected from their sensor or received from neighbouring nodes) towards a so called sink, which requires the information of the WSN. In general more sinks are possible in WSN, but these networks are out of scope of this report.

We also assume that nodes are randomly scattered in an area, and do not move once deployed. With these assumptions we see the need of a routing tree. This tree specifies how the data of the nodes is communicated to the sink, which is the root of the tree.

Recently research is focussing on Quality of Service (QoS) of a WSN [6]. To deliver QoS a particular WSN has to fulfill QoS constraints, by setting the appropriate configurations for its nodes. In this report we focus on the following metrics of our network on which we can apply QoS constraints.

1. (Information) completeness: This is a measure of the fraction of all generated data that arrives at the sink and is approximately equal to the average end-to-end communication reliability between all nodes. The end-to-end communication reliability of a path is defined by multiplying the reliability between each node that is linked on the path. The reliability of the link, which is the probability of the data transmitted being received by a node, between two nodes depends mainly on the received SNR and size of the data packet.
2. (Detection) speed: The detection speed of the application is defined by the minimum of the inverse of the detection delay of a node added to the time needed to send the data to the sink, for all nodes. The detection delay for a node is defined by the time it takes from the arrival of a nearby sensor event until its detection by the node. The time needed to send the data to the sink depends on the number of hops, or hop-count, from the node to the root, and the transmission delay of sending data between nodes.
3. Lifetime: This is defined as the time until one of the nodes is out of energy resources delivered by the energy source, like a battery.
4. Coverage degree: This is defined as the percentage of time an area is covered by at least one sensor, during a certain period.

More information on the exact details can be found in [6].

To investigate the feasibility of certain QoS constraints we need to configure the nodes in a WSN, which means setting the parameters of the configuration, in such a way that the QoS constraints are satisfied if possible.

As there are usually multiple configurations possible per node and QoS constraints are usually conflicting this is becoming a complex task, because the solution search space is typically very large.

As an example we can look at a WSN with 100 nodes with three parameters and two possible values for the parameters. This already has  $8^{100}$  different configurations which need to be compared to the QoS constraints to check the feasibility of complying to them.

In [6] an efficient and scalable method is suggested to search the configurations that satisfy or trade off multiple QoS constraints. The approach uses a Pareto analysis approach based on Pareto algebra [5], in order to explore trade-offs between system properties, and to incrementally compute a set of feasible configurations.

This Pareto algebra offers a structured way of analyzing the configuration space, which guarantees to find all Pareto points in a quick hierarchical way. This is an improvement compared to classical approaches of searching solution spaces, like genetic algorithms [13], which are not guaranteed to find all Pareto points and sometimes require exhaustive search of the whole search space.

A Pareto point describes the different metrics, like completeness and speed of the system while using a particular system configuration. Pareto optimality is used as a criterion to compare configurations and discard the ones that are not optimal, to keep the set of configurations manageable.

A Pareto point is optimal when it is not dominated by another point. A Pareto point  $x$  dominates a point  $y$  if at least one of the metrics of  $x$  is better than the metric of  $y$  and none of the metrics of  $x$  are worse than those of  $y$ .

This idea as discussed in [5] is used to construct the algorithm of [6] to search the Pareto optimal configurations of a WSN.

A brief and simplified description of the algorithm can be found below. More information on the exact details can be found in [6].

1. The algorithm starts with calculating the metrics and configuration set of the leafs of the routing tree. Finally the Pareto optimal configurations, which is called the minimized configuration set, is calculated and sent to its parent together with the corresponding metrics.
2. After receiving the minimized configuration set of its children a node combines these configurations by the free-product and new quality metrics are derived. Finally the minimized configuration set is calculated and sent to its parent together with the corresponding metrics. The group of nodes from which its minimized set is combined is called a cluster (in the case a node  $x$  combines the minimized sets of its children, the clusters of the children are combined into a new cluster).
3. The previous step repeats itself until the root receives the minimized set of all its children and combines these. Now all possible configurations of the network have been investigated as is proved in [6]. The minimized set of the root is the final set of Pareto optimal configurations.

The result of the algorithm is a set of configurations and the corresponding metrics. These metrics are subject to the QoS constraints to determine the feasible configurations. This could be more than one configuration and trade-offs can be made between metrics.

As can be seen the routing tree is important in deciding how to combine the configuration sets.

Currently the algorithm of [6] makes use of a shortest path spanning tree (SPST) as the routing tree. This SPST is created by Dijkstra's algorithm.

In this report we look at the influence of the routing tree on the run time needed to run this Pareto analysis and try to optimize the tree given the following observations:

1. The run time of the algorithm mainly depends on the number of clusters that are combined per step, and the number of configurations in a cluster [6]. The number of clusters that are combined by the free-product is defined by the number of children of a node. This makes the number of children of a node (which is one less than the node degree) an important factor in the run time of the algorithm. Therefore, we want to keep the number of clusters to combine and consequently the number of children as low as possible. We have to rely on minimizing the sets to keep the clusters small. For the number of configurations that are not optimal, and therefore can be removed from the configuration set, it is hard to give a bound [6].
2. Even though we cannot predict the number of configurations that are combined we see from experiments that higher upward the tree (towards the root) we need more run time for combining two configuration sets compared with the time needed to combine sets closer to the leaves. As a consequence the total run time of the algorithm is mainly determined by the time needed by the nodes close to the root and the root itself.
3. The algorithm can be run in either a centralized or distributed way. In the centralized version the run time of the algorithm is determined by the time needed to construct the minimized set and metrics for every node. For the distributed version the run time of the algorithm is determined by the time needed to construct the minimized set and metrics for every node on the critical path. This is the path from leaf to root from which the nodes require the most run time.

In this report we also analytically want to calculate the run time of the algorithm while using different routing trees to see the expected difference between using different routing trees. We then need to make an assumption on the number of configurations kept when minimizing the set, because no bound can be given for this. Using the above observations and assumption we use the following formulas for the analysis:

1. Centralized version:  $\sum_{x \in nodes} n^{D_x}$ , with *nodes*: set of nodes in the tree, *n*: fixed number of configurations in the minimized configuration set and  $D_x$ : degree of node *x*.
2. Distributed version:  $MAX_{L \in paths} \sum_{x \in on L} n^{D_x}$ , with *paths*: set of paths from any leaf to the root, *n*: fixed number of configurations in the minimized configuration set and  $D_x$ : degree of node *x*.

## 1.2 Research questions

The research on the influence and optimization of the routing tree of the Pareto analysis to reduce its run time, will be driven by the following questions:

1. How do we construct a shortest path spanning tree (SPST) that has the lowest maximal node degree?
2. What is the influence of constructing such SPST, compared with the original Dijkstra method?
3. How can we construct a routing tree that minimizes the node degree given information completeness and detection speed constraints?
4. What is the influence of using such minimized node degree routing tree compared with using a tree created with the original Dijkstra method and the SPST with the lowest maximal node degree?

## 1.3 Overview

In section 2 we will take a look at research question 1 and 2. After that in section 3 question 3 and 4 will be discussed. We will mainly look at the centralized versions of the algorithms to construct the trees. The new challenges with making these algorithms distributed are discussed in section 4.

## 2 Optimizing the tree without trade-offs

### 2.1 Introduction

The construction of a shortest path spanning tree (SPST) with a single source as is used for the algorithm in [6] until now, usually called the single-source shortest path problem or shortest path tree problem, is a well known problem and can easily be solved by using for example Dijkstra's shortest path algorithm.

In our case all edges have the same weight, which turns using Dijkstra's algorithm in a breadth first search. This approach leads to an SPST in which the node degree is not taken into account and due to the behaviour of the breadth first approach the node degree is usually not balanced, which means there is a large difference between the degrees of different nodes. The node degree is the main factor in the run time needed to perform a cluster step for that node as can be read in section 1.

If we want to reduce the run time of the algorithm and we do not want to affect the quality metrics, we need to find an SPST with a balanced node degree. Because there are usually more SPSTs for the same network we need to have an algorithm to find an SPST with a balanced node degree.

We could find the balanced SPST from scratch, which means directly from the given network, or try to balance a given SPST for the network. In this report we look at balancing the SPST we get from running Dijkstra's algorithm. Constructing a balanced SPST from scratch might be a subject of future work.

In this section we first make some observations about the problem, followed by an algorithm based on these observations. With this algorithm some experiments are done before concluding the part about optimizing the tree in which we want to avoid any trade-off between run time and quality in terms of the obtained metrics.

### 2.2 Algorithms

In literature the problem of finding a spanning tree with the minimum maximal degree is called minimal degree spanning tree. The problem of finding the minimal degree spanning tree, is proved to be in NP ([3]).

A related problem of degree constrained shortest path can be found in [8]. The algorithm in this report is not applicable, because only a part of the problem is discussed.

An approximation to the directed minimum degree spanning tree problem, which is closely related to the problem we have here, is given in [9]. Before we can use this approximation in our case, we need to adjust the algorithm to keep an SPST after running the algorithm given. In the algorithm a node can choose every neighbour as its parent as long as that parent has a path to the root, but in our case a node can only choose a parent which has the same hop-count from the root as the hop-count of the parent after running a shortest path algorithm. The algorithm can be simplified because in our case the edges all have the same weight and all nodes already have a connection to the root.

After changing the input and simplifying the algorithm of [9] the algorithm can be described by the following pseudo code.

**Algorithm** *Optimize(Node x)*

1. **for** all children  $y$  of  $x$
2.       **for** all neighbours  $i$  of  $y$
3.           **if**  $i$  'better' parent than  $x$
4.               **then** change parent of  $y$  from  $x$  to  $i$

The input for this algorithm is a dynamic set of nodes which is based the node degrees. The initial set consists of all nodes. Currently the highest degree node from this set is taken as input for the algorithm. If the degree can be reduced after running algorithm *Optimize* the node is inserted in the input set to be optimized again if possible, otherwise it is removed from the input set. These steps repeat until the input set is empty.

The 'better' parent has a degree that is at least two less than the current parent and has the same hopcount as the current parent.

In this algorithm some implementation choices have to be made, especially the choice of order of searching children and neighbours.

In the current implementation the children are visited according to their distance (in meters not hops) from node  $x$ , starting with the closest one. The same criterion is used for visiting the neighbour of a child  $y$ . With the criterion we focus on finding a 'better' parent that is close to  $x$  to keep the decrease in completeness as little as possible.

With this algorithm some experiments are done, which are described in the next section.

## 2.3 Experiments

The experiments are done on a PC with a 1.86 GHz Core 2 Duo processor (same PC as used in [6]).

For optimizing the tree we need processor time which means that we expect the construction of the optimized tree to take longer than the construction of the initial tree. But we also expect a reduction in run time needed for the Pareto analysis due to this optimization. In general we could also lose quality while optimizing the tree. This means that the Pareto points we find with the optimized tree are worse than the Pareto point we find with the initial tree. (More information on comparing Pareto points can be found in [7].)

In this section we look at these three factors.

**Increase in run time needed to construct the tree** First we take a look at the extra run time we need for creating the tree. Figure 1 shows the average run time needed to construct the tree for 100 networks per network size.

We take 100 networks per network size for every experiment throughout this report because for this number it is still manageable to create experiments and it is high enough to cancel out most of the deviation in results.

What we can observe in figure 1 is that the optimization algorithm needs more run time to construct the tree than the original algorithm, as was expected. This is an important factor in the usefulness of the algorithm. The extra run time needed to optimize the tree should be less than the reduction in run time gained. More about this later in this section.

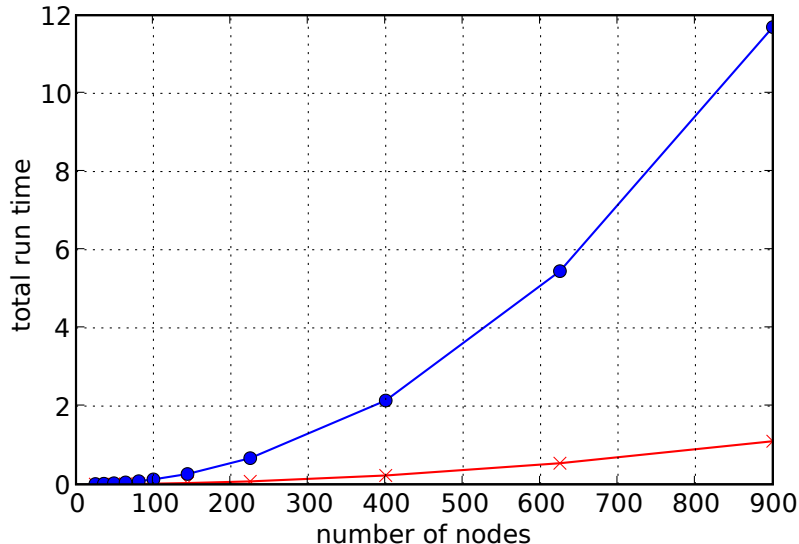


Figure 1: Run time needed to create optimized tree (dot marker) and initial tree (cross marker) in seconds

**Quality loss of the Pareto points** Another metric we trade off is the quality of the Pareto points. As the name of this section suggests we do not want any trade-off in quality. Keeping an SPST after optimizing an SPST is the safest choice to keep the same quality. But between SPSTs there are small differences in quality, the most obvious ones in information completeness. In the optimization steps parents of nodes are changed. If the new parent is further away than the old parent, the reliability of the link reduces even though the hop-count stays the same. This is a random effect and we do not expect this effect to be significant, especially in the dense sensor networks we consider, which is also confirmed by the results displayed in figure 2. We can experimentally get the reduction in information completeness of a network after optimization by looking at the difference between the maximal completeness of the Pareto points obtained when using the initial tree compared with the maximal value for the completeness metric of the Pareto points obtained when using the optimized tree, for a number of networks. This is done in figure 2 for 100 networks per network size.

For the speed metric we expect even less influence of the optimization algorithm. The distance between nodes is not taken into account for the calculation of the speed metric and the hop-count is not changed. There are some small changes possible due to the fact that parameters like the sample rate can be different for some nodes on the critical paths of the optimized and initial tree. But this effect is very minimal as can be seen in figure 3. We can experimentally get the reduction in speed of a network after optimization by looking at the difference between the maximal speed of the Pareto points obtained when using the initial tree compared with the maximal value for the speed metric of the Pareto points obtained when using the optimized tree, for a number of networks. This is done in figure 3 for 100 networks per network size.

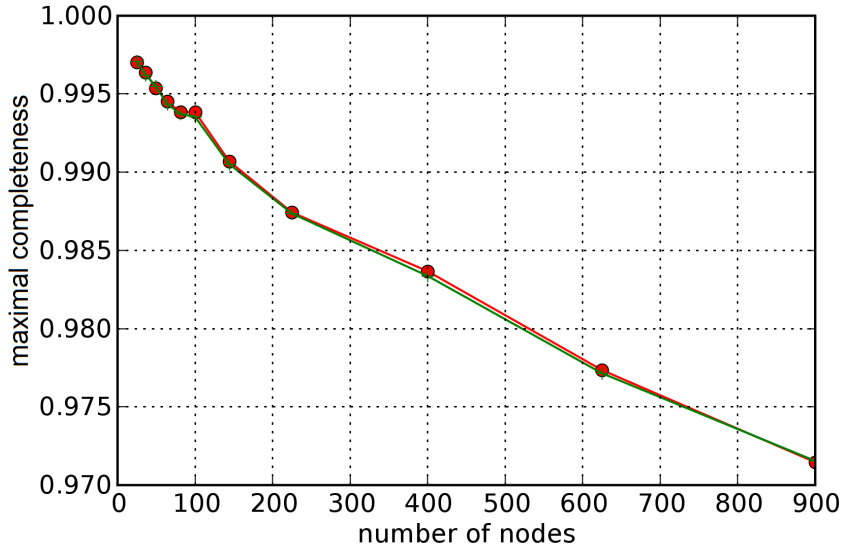


Figure 2: Maximum completeness of Pareto points with using optimized tree (plus marker) compared to the initial tree (dot marker)

For this experiment the average speed of the tree is taken for 100 networks per network size and compared with the average speed of the tree, optimized with the algorithm, of the same 100 networks per network size.

As the loss of quality after optimizing is negligible, we simplify the problem by saying that our algorithm does not reduce quality and therefore does not trade off any metric for the reduction of run time needed for the Pareto analysis.

**Analytical reduction in run time needed for the Pareto analysis** Now we take a look at the real improvement we get by optimizing the SPST: the reduction in run time needed for the Pareto analysis.

To view the average run time reduction of the optimization steps we can start by analytically calculating the expected run time.

In figure 4 we see the result of calculating the average run time of the tree of 100 networks per network size with 8 possible configurations per node and a minimized product set of 8 configurations for each cluster.

In figure 4 the analytically calculated average run time is visualized, which is possible if we make the assumptions described above. The time units in this figure are only for analysis and don't represent seconds, but time needed to combine configuration sets.

Due to the variations in the analytically computed execution times between different networks of the same size (caused by variations in node degrees), the lines do not have a behaviour as linear as expected.

In this experiment all the optimizations were better than the initial solutions (this results from the fact that any optimization done by the algorithm reduces the maximum node degree

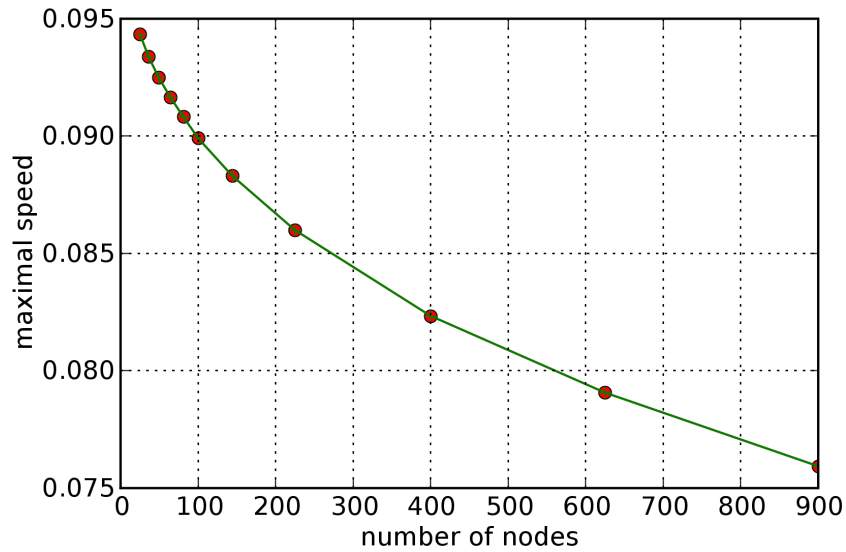


Figure 3: Maximum speed of Pareto points with using optimized tree (plus marker) with only changed parents compared to the initial tree (dot marker)

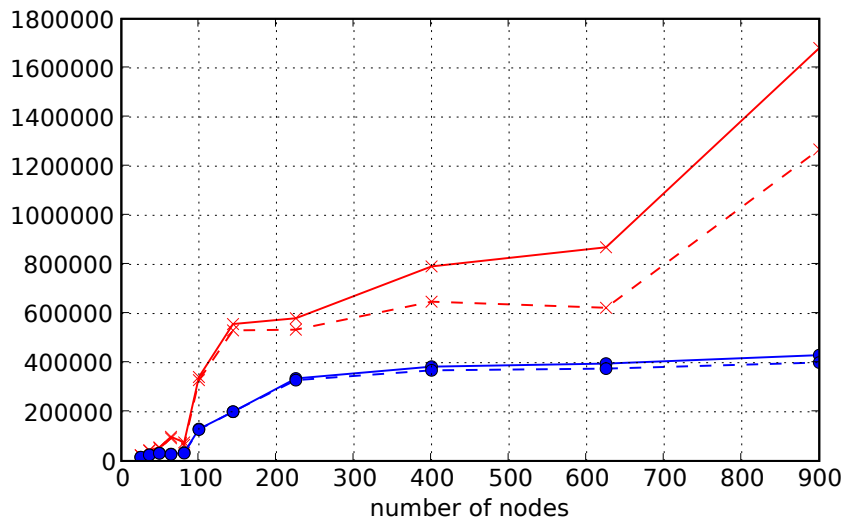


Figure 4: Run time analysis of optimized tree (dot marker) and initial tree (cross marker) for the centralized (solid lines) and distributed (dashed lines) approach.

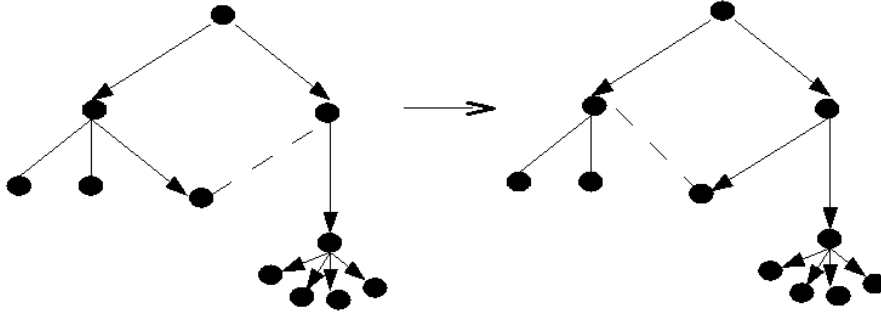


Figure 5: Scenario influencing critical path

(or number of nodes with maximum node degree) and therefore the run time needed to run the Pareto analysis according to our analytical formulas).

We see from this analysis that we expect the optimization step to perform better than the initial Dijkstra approach and that this effect is larger for larger network sizes.

What we observe is that with using the optimization the distributed and centralized run time of the algorithm are closer together.

To explain this observation we look back at the initial problem. We have constructed an SPST in which the hop-count from every node to the root is minimal. The run time for the distributed solution mainly depends on the node degrees critical path. These node degrees can be changed while optimizing the tree, because we only look at optimizing the node degree per level.

In figure 5 we see an example that shows that optimizing the node degree can result in a possible increase of node degrees on the critical path which makes the run time for the distributed execution of the algorithm larger using the analytical approach described in section 1.

What we can conclude is that the optimization step on the tree does not take the distributed environment in which the tree is used into account. In a distributed environment the run time needed for the critical path is most important. In this report we do not look at optimizing the distributed execution. This might be subject of future work.

**Experimental reduction in run time needed for the Pareto analysis** To view the average result of the optimizations we do an experiment by running the Pareto analysis approach with some optimized trees. Figure 6 shows the average run time of the Pareto analysis using the initial and optimized tree of 100 networks per network size, with 8 possible configurations per node. We expect the results to have the same behaviour as the analysis visualized in figure 4.

The first thing we can observe in figure 6 is the different behaviour of the lines compared to the analytically expected behaviour, especially the behaviour of the lines with the dot marker (Pareto analysis using the optimized trees). They do not follow the smooth curve like the lines with the cross marker does (Pareto analysis using the initial trees). This is due to the fact that the result of the optimization algorithm is strongly dependent on the given

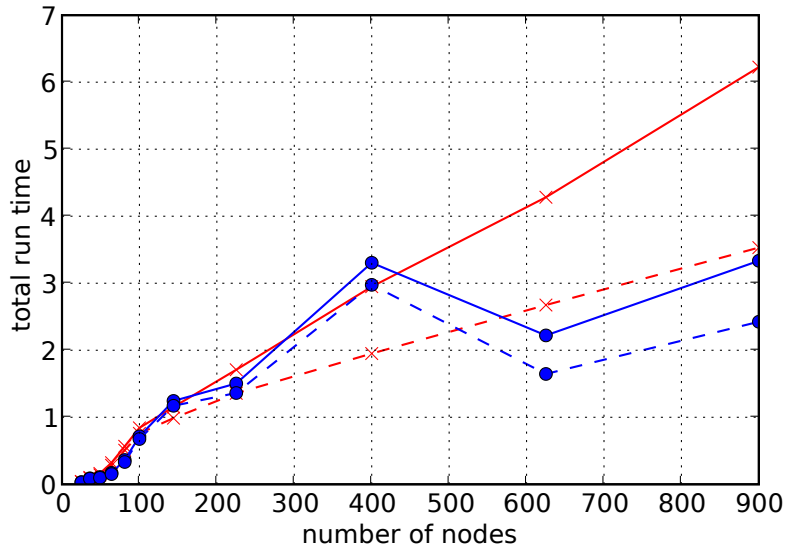


Figure 6: Run time of Pareto analysis using the optimized tree (dot marker) and initial tree (cross marker) in seconds for the centralized (solid lines) and distributed (dashed lines) approach.

network and the amount of improvement can vary a lot.

We also see that the run time for the centralized and distributed execution of the algorithm are closer together which is also observed when analytically calculating the behaviour of the run time.

Even though we expect the optimization step to improve the solution in all cases, we observe that in some cases the optimization algorithm gives a tree that makes the Pareto analysis run even longer than the initial tree.

Further investigating the results revealed that the problem lies in the inability to change the root's degree and the fact that there is a great uncertainty in the number of Pareto points that are outputted each cluster step.

The optimization algorithm tries to reduce all the nodes in the tree except of the root. This is because we want an SPST in which the paths need to be the shortest and in the case of the root, each neighbour of the root can only be connected to the root in the shortest path case and no optimization of the degree of the root is possible.

Also due to experiments we saw that the root had a significantly larger product set than in the initial tree in most of the cases and because of the high degree and the fact that the run time almost completely depended on the root's run time, the total run time actually increased.

In this particular experiment, for 172 out of the 1100 networks (11 times 100 networks) this increase was even more than the total reduction in run time, resulting in an optimized tree requiring more run time for the Pareto analysis than the initial tree.

What we can conclude from this is that different trees, even though they are all SPSTs, can result in different sized product sets for the same nodes. And if we cannot change the root

this could result in more run time needed for the optimized tree compared to the original implementation for some cases.

We expect that this phenomenon becomes less when the degree of the root is low. With a low degree the root could still need more run time but the improvement obtained from the node degree reduction of the rest of the tree compensates the extra run time needed in the root, in most cases.

If we now limit the maximum node degree allowed in the tree and also fix the root's degree to a small number, we expect a better result where we do not have large variations in run time due to the limited maximum degree and a lot less cases in which the optimizations perform worse than the initial solution due to the low degree of the root.

Limiting the node degree limits the number of networks on which we can run the optimization algorithm, but might give us more insight in the results until now.

As a final conclusion of this part, before looking at the limited degree case, we could compare the results to find out if the effort we need to optimize the tree is less than the improvement we get. So, is the run time needed to optimize the tree less than the reduction in run time for the Pareto algebra?

Until now we can only look at the behaviour of the results and we can not compare the absolute values. This is due to the incomparable implementation of the algorithm that optimizes the tree and the algorithm that runs the Pareto analysis. The first is written in Python, which is an interpreted language and the latter is written in C++ and compiled which is significantly faster than a python implementation.

Even though we cannot use the absolute values we can make some observations about the usefulness of this algorithm. This is to be done in section 2.4.

Writing the tree optimization algorithm in C++ might be subject of future work.

In the rest of this section we look at optimizing a tree with limited node degree. In this case this means that the all nodes have a maximum of six children and the root has exactly four children. We made this choice because taking this constraint for the root any higher results in the same observations as done in the beginning of this section and taking the constraint lower would result in a very low run time for the Pareto analysis.

The networks with these limited degrees are constructed by selecting the networks that comply to the node degree constraints from random networks.

**Increase in run time needed to construct the tree (limited node degree)** In figure 7 we see the average run time needed to create the initial and optimized tree for 100 networks per network size, both with limited degrees.

As can be seen in figure 7 the run time needed for the construction does not differ a lot from the non-limited node degree case.

**Quality loss of the Pareto points (limited node degree)** For the same reason as discussed for the non-limited node degree case we also assume no loss of quality of the Pareto points.

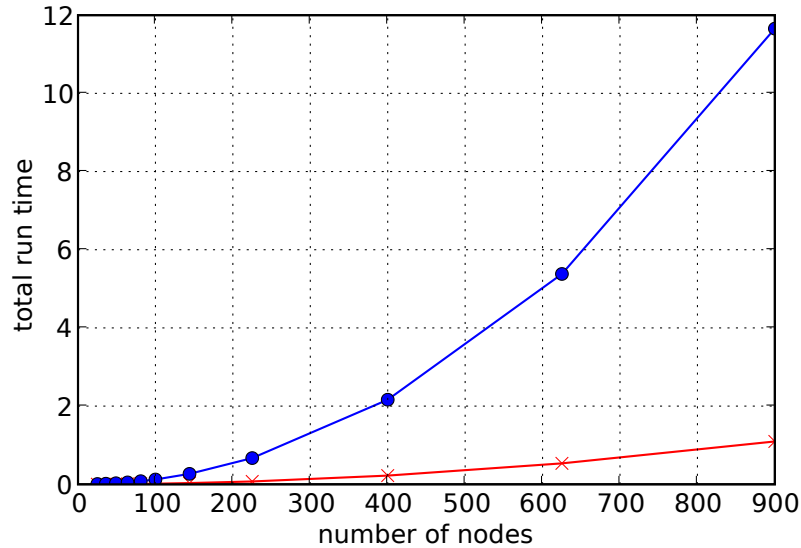


Figure 7: Run time needed to create optimized limited tree (dot marker) and initial limited tree (cross marker) in seconds

**Analytical reduction in run time needed for the Pareto analysis (limited node degree)** If we analyse the behaviour of the run time we expect we get the behaviour as can be seen in figure 8.

In this figure we see the average run time we analytically calculate needed to run the Pareto analysis for the initial and optimized tree for 100 networks per network size, both with limited degrees.

As we have not changed the parameters of our analysis and assumed the same number of Pareto points for each configuration set, this behaviour is the same as the non limited case. Only the absolute values of the analysis differ, because we now optimize trees with a limited degree.

**Experimental reduction in run time needed for the Pareto analysis (limited node degree)**

More interesting results might be found in the reduction in run time needed for the Pareto analysis.

In figure 9 we see the behaviour of the Pareto analysis for the initial and optimized tree for which the maximum degree is limited to seven and the degree of the root is fixed to four for the 100 networks per network size. We now observe a smoother curve and when looking at the results none of the 1100 networks (11 times 100 networks) has an improvement that is worse than the initial solution.

What we can conclude from figure 9 is that with a limited node degree the average improvement of the optimization step is large. We can also conclude that the problem of the optimizations being worse than the initial solution becomes less if we limit the root's degree.

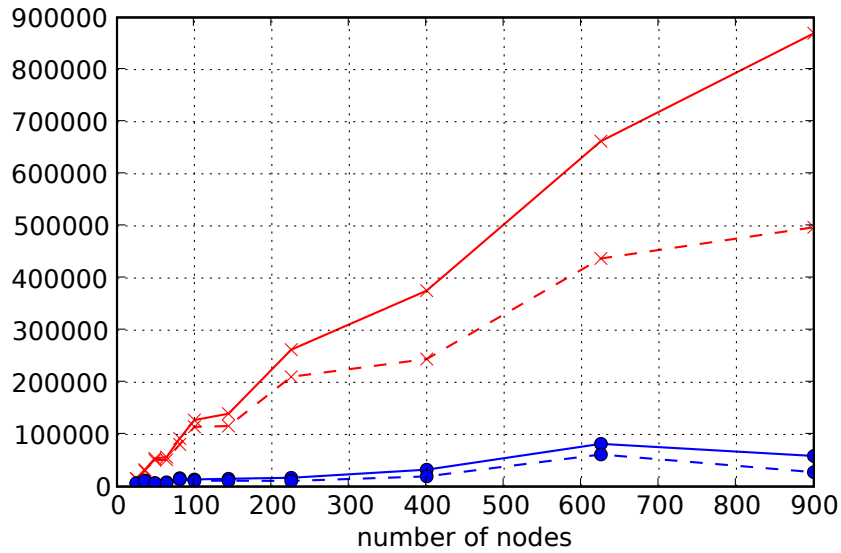


Figure 8: Run time analysis of optimized limited tree (dot marker) and initial limited tree (cross marker) for the centralized (solid lines) and distributed (dashed lines) approach.

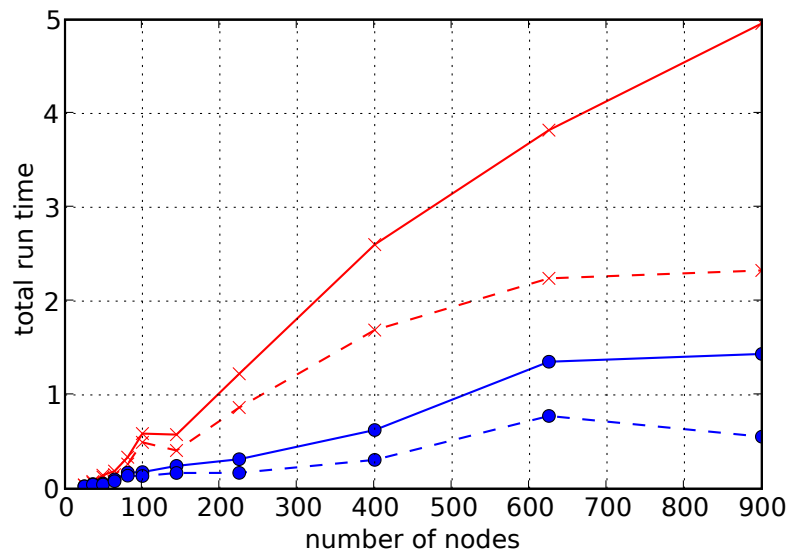


Figure 9: Run time of Pareto analysis using the optimized limited tree (dot marker) and initial limited tree (cross marker) in seconds for the centralized (solid lines) and distributed (dashed lines) approach.

## 2.4 Conclusion

In this section a method for optimizing the tree construction part of the algorithm for calculating the configuration in a wireless sensor network without affecting the final quality metrics is discussed.

The optimization algorithm given in literature is adapted to solve the problem of this section. The algorithm optimizes the tree by balancing the node degree in the tree. Using analysis we expect a large improvement in run time of the Pareto analysis due to the optimized degree. But what we observe is that the fact that we cannot reduce the degree of the root makes the improvement less than expected and this effect is even more visible due to the unpredictability of the size of the Pareto sets involved in the clustering steps.

This observation is also confirmed by the experiments with limited node degrees. In those cases, we actually get a good improvement if we limit the node degrees and especially if we limit the degree of the root.

The usefulness of this algorithm in the general case can be discussed by looking at the trade-off between the extra run time needed for the tree construction and the reduction in run time for the Pareto analysis. As discussed in this section the absolute times cannot be compared, but that is not needed in this case. We can conclude that even though we spend time on optimizing the tree and the degrees actually get balanced we cannot guarantee in all cases that the run time of the Pareto analysis is reduced due to the fact that the root, from which the degree is the main factor in the run time needed for the Pareto analysis, cannot be optimized. This effect is more visible due to the unpredictability of the size of the Pareto sets involved in the clustering steps.

### 3 Optimizing the tree with trade-offs

#### 3.1 Introduction

What we have seen in the previous sections is that the node degree is of great importance when optimizing the run time.

When we stick to an SPST some degrees cannot be reduced due to the lack of alternatives, for example because a path needs to be the shortest and in the case of the root, each neighbour of the root can only be connected to the root in the shortest path case.

To reduce the node degree even further by overcoming the problems described above we need some kind of trade-off.

In this section we look at trading of speed and completeness for the reduction of run time needed for the Pareto analysis. We only look at these metrics as these are the most interesting when it comes to changing the routing tree.

A similar problem of delay constraint least cost routing is described in [12] which is NP complete. In this and the rest of the literature the problem where we look at multiple constraints is not treated and that is why we first start to define the problem to find an own algorithm.

#### 3.2 Observations

The following observations can be made to define the problem:

- When using an approach which tries to reduce the node degree every iteration of the algorithm, increasing the path length should only be done when no other choice, which does not affect the metrics, is possible. The best choice influences the final quality metrics as little as possible.
- If there is no choice to reduce the node degree without affecting the final quality metrics, the only way to reduce the degree of a node is to connect a child to another node and increase the hop-count of that child (and all nodes on all the paths from that child to the leafs).
- Connecting a child of a node with maximal degree to a new parent results in a better maximal node degree (or reduction of the number of nodes with maximal degree) if the following condition holds:
  - The child has a new parent which has a degree at least two less than the old parent.

If the above condition holds the parent of the child can be replaced if and only if one of the following steps are possible:

- Without changing the structure of the rest of the tree, the increased path length caused by changing the parent does not violate the completeness or speed constraints. (This is later called step 1 and is explained in more detail in section 3.4)
- If the above step is not possible, so if changing the parent of a child violates the constraints, further changing the tree can ensure that the paths that violate the constraint are made smaller. This can be done by changing the parent of a node

on the path that is too long such that it does not violate the constraints any more. (This is later called step 2 and is explained in more detail in section 3.4)

- If multiple neighbours of a node have a degree at least two less than the current parent, then we need a criterion to decide to which neighbour to connect. We could for example choose the one with the lowest difference in distance (meters or hops) compared to the current distance from the parent to the root to minimize the increase in path length and consequentially the final metrics. Another choice would be to connect the node to the new parent that has the smallest degree of all, but this could result in unnecessary increase of the path length.

Before taking a look at an algorithm made using these observations, we take a look at an important step in dealing with the trade-offs. We do not want to trade off a particular metric in such a way that we cannot fulfil the QoS constraints anymore for all the Pareto points we find after running the Pareto analysis. So we need a way to find out to which extent we can trade off a particular metric without violating the constraints.

### 3.3 Checking the constraints

In this section we look at trading off speed and completeness for run time reduction of the Pareto analysis. But first we look how to check the speed and completeness of a tree to make trade-off choices.

The constraints are checked based on the current model used in the implementation of the Pareto analysis, which is based on a TelosB mote [1].

**Effect of algorithm on constraints in current model** Until now the completeness and speed are determined after establishing a routing tree and running the Pareto analysis to find the Pareto points. The formulas for calculating the completeness and speed can be found in [6]. What we want now is to determine, based on the routing tree, what the completeness and speed metric values are, to allow us to make a good trade-off decision.

Only if the values of the metrics are better than the constraints required we may want to decide to trade off a particular metric.

We do not want to run the Pareto analysis algorithm every time to check the completeness and speed metric and the corresponding constraint as this would take a lot of time.

If we want to determine the completeness and speed when using a particular tree before we run the Pareto analysis we do not know all the values of the parameters, like transmission power for each node, which are known after running the Pareto analysis.

The calculation of the completeness and speed is based on these parameters and to make sure that we do not overestimate the metrics we need to take the worst case assumption for the parameters.

When we take the worst case assumptions while calculating the metrics we ensure that all Pareto points we find comply to the constraints (if using the initial SPST also complies to the constraints). What we observe from initial experiments is that this is indeed the case but there are also a lot of Pareto points that have metric values which are much better than the constraints given. If we want to reduce the run time needed for the Pareto analysis as much as possible we want to give as much freedom as possible to the tree optimization algorithm,

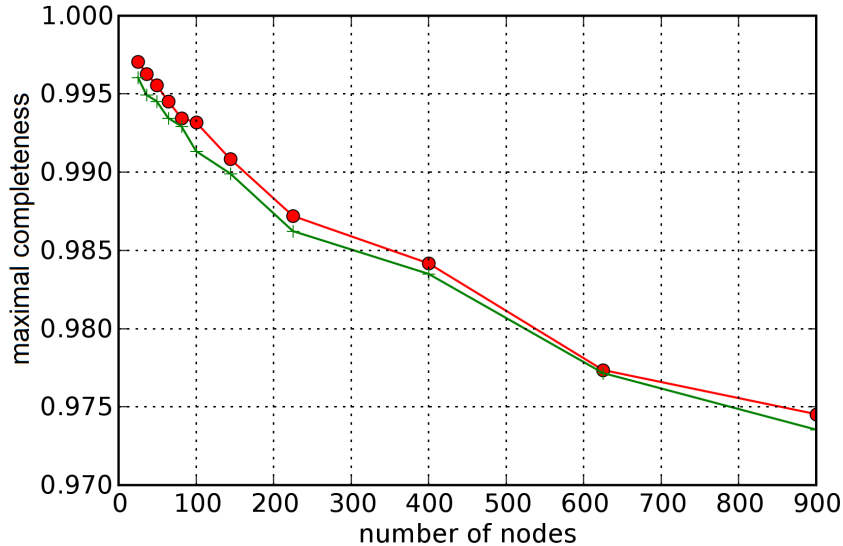


Figure 10: Maximum completeness of Pareto points with using full optimized tree (plus marker) compared to the initial tree (dot marker)

which is now greatly reduced by taking the worst case assumptions. What we want is to optimize the tree as much as possible such that there is at least one Pareto point that complies to the constraints. If we try another approach than taking the worst case assumption we have to show experimentally in how many cases the new predictions result in not finding any Pareto point fulfilling the constraints due to over optimization.

Before looking at a better prediction we take a look at what happens to the metrics if we do not use any prediction at all. This means that we can optimize a tree as much as possible without looking at the constraints. In this case we optimize the tree as much as possible which means that we get the greatest reduction in run time for the Pareto analysis when using this algorithm. But the disadvantage is that all the Pareto points we find using this tree might not comply to the constraints.

We can experimentally investigate this effect.

We can experimentally get the reduction in information completeness of a network after optimization by looking at the difference between the maximal completeness of the Pareto points obtained when using the initial tree compared with the maximal value for the completeness metric of the Pareto points obtained when using the optimized tree, for a number of networks. This is done in figure 10 for 100 networks per network size.

We observe from these figures that the completeness is not significantly changed on average when using the current model. This effect is explained later in this section.

The same experiments are done for the speed metric. The effect of using an optimized tree on this metric is visualized in figure 11.

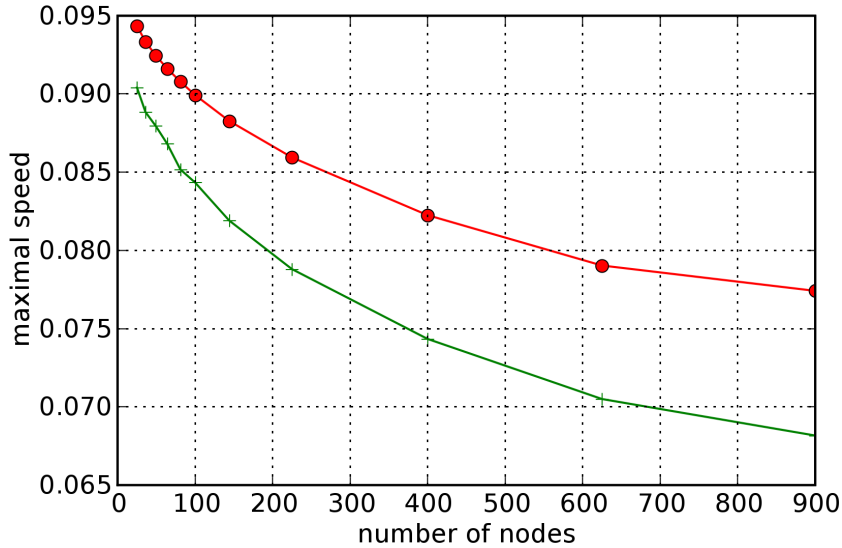


Figure 11: Maximum speed of Pareto points with using full optimized tree (plus marker) compared to the initial tree (dot marker)

The speed of the Pareto points using the initial tree is a bit higher, but still not significantly reduced when using the optimized tree if we keep in mind that the path length is 1.75 times the original path length by optimizing as much as possible for the particular experiment done now. The speed is reduced by around 10%, while we see a great reduction in run time needed for the Pareto analysis as is seen in section 3.5. In the remainder of this section we try to explain this effect and the influence of the current model in this.

To explain the results from figure 10 and figure 11 we look at the individual actions in the algorithm that can influence completeness and speed.

There can only be a major difference between the completeness and speed metric of the Pareto points obtained when using the initial tree and the completeness and speed metric of the Pareto points obtained when using the optimized tree by changing the parent while keeping an SPST (as in the non trade-off part) or increasing the path length (as in the trade-off part) is of a significant influence to the completeness or speed metric.

**Influence of changing the parent of a node on the completeness metric without changes in path length** Changing the parent of a node while maintaining the SPST, can result in slight changes in completeness. This effect is also discussed in section 2.3 and is visualized in figure 2.

**Influence of changing the parent of a node on the speed metric without changes in path length** Only changing the parent has even less influence on the speed metric. This effect is also discussed in section 2.3 and is visualized in figure 3.

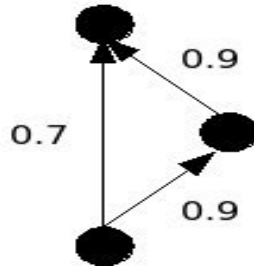


Figure 12: Situation with more reliable longer path

Changing the parent of a node to optimize the node degree has a neglectable influence on both the completeness and the speed metric.

More influence on these metrics is expected when increasing the path lengths in the tree.

**Influence of increasing the path length on the completeness metric** Intuitively increasing the path length should reduce the reliability of a path and therefore the information completeness because the packet needs to travel further.

In figure 10 we see the influence of using a fully optimized tree with changing parents and increasing path lengths compared to using the initial tree for the Pareto analysis for 100 networks per network size. The completeness is reduced just a bit more than when only changing the parents, what might be a bit counterintuitive as we increase the path length with 75% on average.

But further looking into the influence of increasing the path length shows that the situation in figure 12 is also possible.

The reliability can actually improve when we increase the path length. In figure 12 we see an example that if we take the direct link from the lower to the upper node we have a reliability of 0.7, but if we take the route via the intermediate node we have a reliability of  $0.9 * 0.9 = 0.81$ . So the longer route actually is more reliable.

Further looking at the experiments leading to the result shown in figure 10 shows that this effect almost cancels out the reduction in completeness we get when increasing the path length.

**Influence of increasing the path length on the speed metric** In figure 11 we see that increasing the path length has some influence on the speed metric. But this effect is minor if we take the fact that the path length is increased by 75% on average.

This effect is only minor due to the current model. The delay is calculated by adding the worst case delay of detecting an event after its occurrence near a sensor to the time needed to send the packet over the network. In the current model most of the time is needed for the worst case delay to detect an event after its occurrence near a sensor and increasing the length of the path has no significant influence in the calculation of the delay.

**Effect of algorithm on constraints with different models** The model until now focusses on a implementation using a TelosB mote. This has influence on the results we get in the experiments. An obvious influence is in the delay calculation. Increasing the path length has no significant influence because in the current model sending a packet over a path is only a

small portion of the delay. In another model this might be completely different. Also in the case of completeness the model can be changed such that changing parents or increasing path lengths has a large influence on the metrics. Whether the model stays realistic in this way is a different question. Doing the same experiments like is done in this section with other models might be subject of further work.

**Conclusion** What we can conclude from this subsection about constraint checking is that constraint checking takes time and reduces the freedom we have for optimizing the tree, because we need to have a conservative prediction of the metrics.

Experiments show that in the current model completely optimizing the tree has only a small influence on the metrics due to reasons explained in this subsection.

We assume that if the Pareto points we get from using the initial, non-optimized, tree fulfill the constraints, also the Pareto point we get after fully optimizing the tree fulfills the constraints. The only exception in this is when the Pareto points of the initial SPST already violate the constraints or are only nearly fulfilling them.

In the experiments of this section no check for the constraints is done anymore and the trees are all fully optimized.

### 3.4 Algorithms

If we combine the observations from section 3.2 and the fact that we do not check the constraints as discussed in section 3.3 we can conclude that the second step of our algorithm becomes superfluous. The second step is only performed when the first step cannot reduce the degree of the node due to constraints limitations.

In this section we look at the algorithm performing the first step and take a quick look at the second step.

Using the observations done in section 3.2 we can construct a general algorithm which consist of the following steps. The input for this algorithm are all the nodes that want to be optimized (which are the nodes that have at least two children. More information on choosing the nodes to optimize in section 3.5).

0. Use the approximation algorithm described in section 2 to reduce the node degree without trade-offs and if no reduction of the degree of the node is possible without trade-offs:
1. Try to reduce the node degree with step 1 as described in section 3.2 and if the degree of the node cannot be reduced due to constraint violations:
2. Change the original path lengths to allow more increase of path length (step 2 as discussed in section 3.2).

If none of the steps can be done for a node, it has reached its lowest degree possible when using this algorithm.

As said before the second step is not needed when we do not check any constraints in our case. The idea behind step two is that the tree can be changed to allow some paths to be

enlarged without violating the constraints. This is done by trying to find a node on the path violating the constraint which can be connected to a new parent such that the path length is reduced. The step is not discussed any further, but we look at step one which due to the fact that we don't check constraints becomes less complex:

**Algorithm** *Step1(Node x)*

1. **for** all children  $y$  of  $x$
2.       **for** all neighbours  $i$  of  $y$
3.             **if**  $i$  'better' parent than  $x$
4.             **then** change parent of  $y$  from  $x$  to  $i$

The input for this algorithm is a dynamic set of nodes which is based the node degrees. The initial set consists of all nodes (or all nodes with a degree higher than a predefined minimum, see section 3.5). Currently the highest degree node from this set is taken as input for the algorithm. If the degree can be reduced after running algorithm *Step1* the node is inserted in the input set to be optimized again if possible, otherwise it is removed from the input set. These steps repeat until the input set is empty.

The 'better' parent has a degree that is at least two less than the current parent. Compared to the non trade-off approach of section 2 we do not have to look at the hop-count anymore.

Also in this algorithm some implementation choices have to be made, especially the choice of order of searching children and neighbours.

In the current implementation the children are visited according to their distance (in meters not hops) from node  $x$ , starting with the closest one. The same criterion is used for visiting the neighbour of a child  $y$ . With the criterion we focus on finding a 'better' parent that is close to  $x$  to keep the decrease in completeness as little as possible.

### 3.5 Experiments

With the experiments of this section we want to find the reduction in run time needed to run the Pareto analysis and the costs of doing that in terms of extra run time needed to construct the tree and quality of the Pareto points found after running the Pareto analysis using this tree.

**Increase in run time needed to construct the tree and the reduction in run time needed for the Pareto analysis** In this paragraph we first look at the run time we analytically and experimentally need for the Pareto analysis in the cases we use the initial and optimized tree.

For the analysis of the reduction of the run time needed for the Pareto analysis we expect the same behaviour as in figure 4, but with even a larger difference, as we optimize the degree even more.

To view the average result of the optimizations we do an experiment by running the Pareto analysis approach with some optimized trees. Figure 13 shows the average run time of the Pareto analysis using the initial and optimized tree of 100 networks per network size, with 8

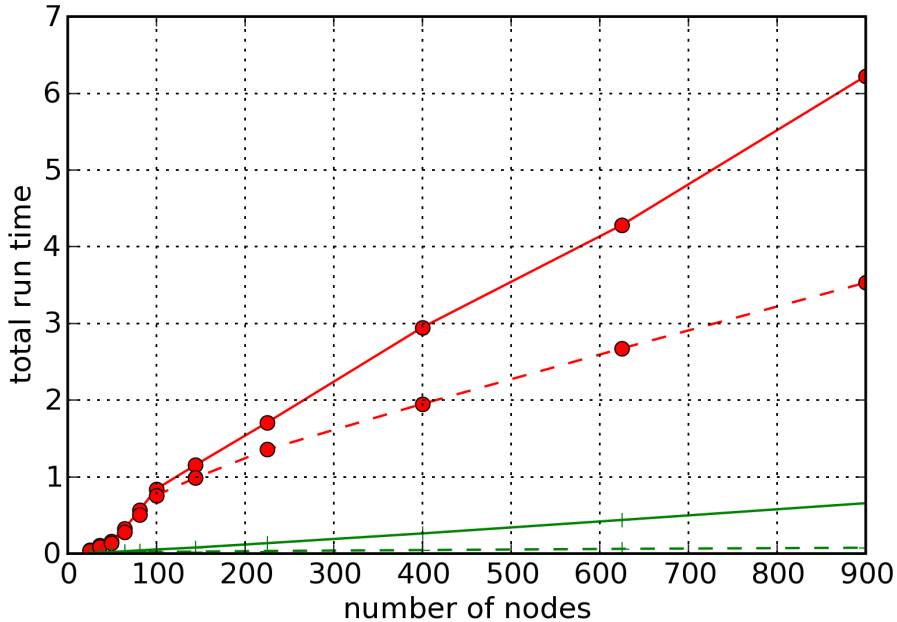


Figure 13: Run time of Pareto analysis for optimized tree with trade-offs (plus marker) and initial tree (dot marker) in seconds for the centralized (solid lines) and distributed (dashed lines) approach.

possible configurations per node. We expect the results to have the same behaviour as the analysis.

The behaviour of the run times in this figure follows the same curve as we expect and is not as unpredictable as the algorithm without trade-offs (figure 6).

Further investigating the results shows that for only 3 out of 1100 the optimized tree requires more run time for the Pareto analysis than the initial tree.

So the problem as seen in section 2 is still there in the trade-off case, but has a lot less effect. This is mainly due to the fact that also the root degree can be reduced in most cases.

Until now the algorithm for optimizing the tree is executed for every node that has at least two children and therefore potentially can be optimized. The run time needed to optimize the tree in this case is high (almost 60 sec for 900 nodes using a python implementation). The run time needed for the trade-off algorithm to optimize a node is much more than is needed to optimize a node in the non trade-off case of the previous section. It is worth looking at the stop condition of the algorithm to prevent spending too much time on just minor improvements. Even though we cannot compare times as discussed in section 2.3 we still can investigate if running the algorithm for a particular time has a significant effect on reducing the run time needed.

A way to describe the stop condition of the algorithm is to define the nodes it needs to optimize. For every network size we can make the following table as we did below for 900

nodes (we still cannot compare absolute values but the relative values gives us insight in the usefulness of optimizing some nodes). First we look at the run time needed to create the SPST followed by optimize the nodes having a particular number of children:

900	construct tree (s)	Time extra	Pareto analysis (s)	Time saved
Create SPST	1.11	0	5.96	0
Nr of children > 6	1.11 + 0.34	+ 0.34	4.77	1.19
Nr of children > 5	1.11 + 0.40	+ 0.40	3.62	2.34
Nr of children > 4	1.11 + 0.65	+ 0.65	2.40	3.56
Nr of children > 3	1.11 + 1.70	+ 1.70	1.36	4.60
Nr of children > 2	1.11 + 5.71	+ 5.71	0.70	5.26
Nr of children > 1	1.11 + 56.55	+ 56.55	0.62	5.34

What we already observe from this table is that optimizing all nodes with more than one child requires a lot more run time to optimize the tree but does not reduce the run time needed for the Pareto analysis compared to only optimizing the nodes more than two children. From experiments we can conclude that this observation is true for all network sizes.

We can conclude a lot of run time is needed to optimize the nodes with more than one child and only minor improvement is gained. The same can be said for the nodes with more than two children. Relatively seen the small improvement gained still requires a lot of run time, but in this case it is less visible as optimizing all nodes with more than one child.

The choice of stop condition might also depend on the implementation of the algorithm, but with the current implementation it is obvious not to optimize nodes with two children.

Investigating this effect with a C++ implementation might be subject of future work.

**Reduction in quality** In section 3.3 we have already seen the influence on completeness and speed of fully optimizing the tree without looking at any constraints. In general we can use a technique to calculate the loss of quality for all the metrics of the Pareto point, by looking at the average distance between Pareto points of the different sets. (More information on this can be found in [7].)

We can make this calculation for the optimization step by averaging the quality loss of 100 networks per network size after optimizing the tree. The results of this experiment are as follows:

Network size	Loss of quality in %
25	2.0434
36	7.7061
49	9.2128
64	8.4989
81	9.7369
100	6.3011
144	3.6029
225	5.9207
400	6.9598
625	8.0436
900	7.0755

What we observe here is that the average quality loss is within 10% even for a network as

large as 900 nodes, but we can also observe that there is no correlation between the network size and the quality loss visible in this experiment.

### **3.6 Conclusion**

In this section we have seen that allowing trade-offs results in a reduction in run time needed for the Pareto analysis, which is more than the algorithm of section 2 allows. But we have to keep in mind that the run time needed to optimize the tree does not exceed the reduction in run time we gain using this optimization or that we do not lose too much quality of the Pareto points for the optimization to be useful.

We could not compare the run time needed for the creation of the tree and the run time needed for the Pareto analysis, but investigated the stop condition of the algorithm to make the reduction in run time as high as possible for as little effort as possible.

For all the experiments we did not check the constraints for the Pareto points because the optimizations did not significantly reduce the metrics. Not checking the constraints also saves run time and gives much more freedom to the optimization algorithm.

In the next section we take a look at making the algorithms discussed until now distributed.

## 4 Distributed execution of the algorithms

### 4.1 Introduction

In the previous sections we discussed centralized versions of algorithms. In a WSN setting it is also a possibility to run the algorithm in a distributed way which is usually faster and more convenient (for example when looking at dynamic behaviour in which we have to frequently update parts of the tree for example due to dying nodes).

If we run an algorithm in a distributed way we only use messages to communicate between nodes and we want to make sure that the communication is as local as possible to avoid much traffic on the network. A node needs just enough information to calculate its own output. It is possible to run the centralized version of the algorithm in a distributed way by gathering all the information of the network in every node, but this eliminates the advantage of a distributed algorithm and that is why we have to develop a distributed version of the centralized algorithms.

### 4.2 Algorithms

**Dijkstra's algorithm** There is already done a lot of research on the distributed execution of a shortest path algorithm. Some of them are discussed in [2]. For our purpose these algorithm are complex, because the links have unit length in our case. Our problem reduces to a breadth first search or so called distributed minimum hop problem.

This problem is solved in [4] and the algorithm we use to make the SPST is based on this report. The algorithm together with the observations we make while implementing it for a wireless sensor network, are discussed below.

For the algorithm implementing Dijkstra's algorithm we can make the following assumptions and observations:

- Every node in the network needs to know its own ID which is unique in the network and needs to keep track of its optimal hop-count until now. This hop-count indicates the number of hops needed to reach the root with the current parent and this needs to be optimized.
- We assume that an underlying mechanism exists for queueing incoming messages to a node and for queueing outgoing messages until they can be transmitted on the outgoing edges and that the number of retries to send the message is high enough that we have a high probability that a message arrives.
- We cannot simply flood the network and assign the node from which a node received a message first, as a parent. This is mainly because we use an unreliable medium to send the messages and it might be possible that a node receives a message from a node more hops away from the root than optimal first due to some problems with the messages on the shortest path. In our algorithm we could specify a time a node waits on messages from neighbours. After the period of waiting we make a choice of parent and communicate our new hop-count. This approach saves sending messages compared to communicating a new hop-count every time a message from a node with a lower hop-count is received. More improvements might be subject of future work.

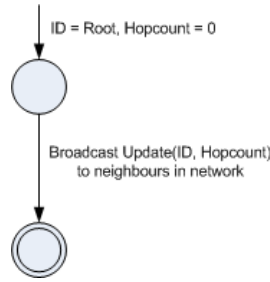


Figure 14: State machine of root of network

- If a message does not arrive we can end up with a tree that is not the optimal SPST, further research could be done in guaranteeing that the algorithm results in the SPST. This could for example be done with introducing acknowledgement messages.
- Important factors of a distributed algorithm are the number of messages sent and the run time needed. These are not analyzed in this section but this could be a subject of future work.

The distributed version of setting up the SPST with Dijkstra’s algorithm works as follows: The root initiates the construction of the tree by informing his neighbours.

If a node receives a message indicating a hop-count, it compares it with its current hop-count and changes the parent if a message from a new parent indicates a better hop-count. After a change of hop-count a message needs to be broadcasted to inform the neighbours of the improved hop-count. To prevent multiple broadcasts after each other due to a longer arrival time of messages from nodes with a lower hop-count, a delay is build in. So when a node receives a hop-count update it waits for a specified time for better hop-count offers from different neighbours. This technique possibly reduces the number of messages needed but increases the time needed to create the tree.

Figure 14 and figure 15 show the state machines indicating how to react and send messages between nodes and can be used to construct the algorithm.

**Algorithm optimizing without trade-offs** Recently, research has been done on the distributed execution of algorithms solving the minimum degree spanning tree ([10]).

For the algorithm on which we based the centralized algorithm without trade-offs ([9]) there is no distributed algorithm yet.

For our algorithm based on the algorithm discussed in [9] we construct a distributed algorithm using some extra assumptions and observations:

- The tree that is optimized is the SPST obtained after running Dijkstra. It might be possible to combine the Dijkstra and non-trade-off optimizing algorithm, but this might be subject of future work.
- In the algorithm we have to make a decisions based on the degrees of neighbours. After Dijkstra a node does not yet know its degree, only its parent. Before being able to balance the node degree by changing parents, nodes need to know the degree of itself and its neighbours.

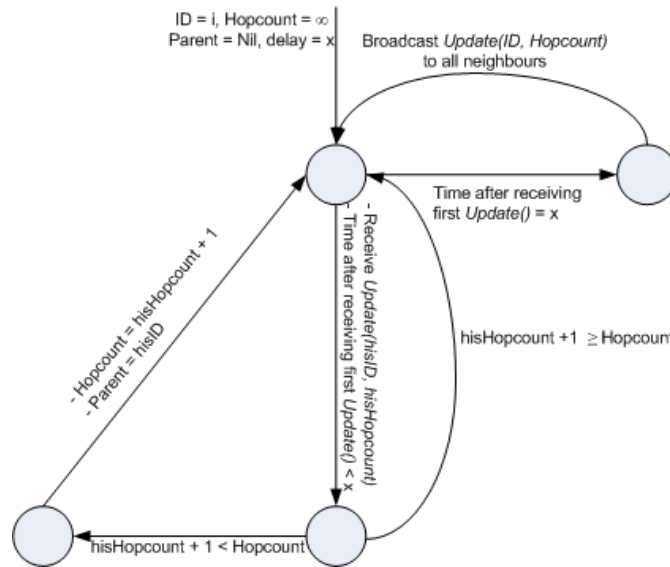


Figure 15: State machine of every node in network except root with a delay of  $x$  seconds

- If the parent of a node needs to be changed, the old and new parent need to be informed to allow them to change their degree, which needs to be done for other neighbours to be able to make optimization decisions.
- With sending messages there is the problem of losing packets, due to for example bit errors and collisions. If we lose for example a packet stating the change of parent, nodes keep optimizing with the wrong information. This is more of a problem compared with Dijkstra's algorithm because in that case we usually have more redundancy of packets due to the flooding approach. Until now we did not look at structures in the algorithms to cope with losing packets. This might be subject of future work.

Based on these observations we can construct an initial algorithm. After that a problem of this algorithm is discussed.

The algorithm is based on different phases. The first phase after creating an SPST is to make sure every node knows its degree. This can be done by every node sending its parent a message notifying itself as his child.

After this phase every node needs to know the degrees of its neighbours to be able to make optimization decisions. This can be done by every node sending its degree to all neighbours. Nodes need to store this information for all neighbouring nodes.

Now a node can compare the degree of its parent with the rest of the neighbours. If there is a neighbour with a degree at least two less than the current parent, the parent can be changed for the same reasons as discussed in the centralized version of the algorithm.

If a node changes its parent an update needs to be send to the old and new parent to reduce and increase the node degree respectively.

This new degree needs to be broadcasted to update a neighbouring nodes of this new degree.

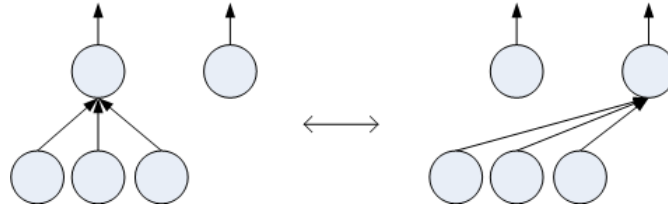


Figure 16: Unstable situation in distributed algorithm execution

In this approach timing is important. The phases assume the information of the previous phase is present.

Another problem that needs to be solved by correct timing of sending messages is the problem of an unstable situation. In this situation a node keeps switching between parents. An example can be seen in figure 16.

This is no problem in the centralized case because nodes are optimized one by one. But in the distributed case the three children see that there is a node that has a degree that is three less than the degree of the parent and they all decide to change. This problem can be resolved by making sure only one child is changed and that the new degree of the new parent is broadcasted before another child makes a decision. So in the case of figure 16 one child needs to be changed and the new degree needs to be broadcasted so that every node knows there is a node of degree two and one of degree one and no optimization is needed. This problem is not fully researched yet and might be subject of future work. A possible solution is to give time slots or another criteria which describe when a node can change its parent such that the solution does not end up in an unstable solution.

**Algorithm optimizing with trade-offs** The most important difference between the trade-off part and the non-trade-off algorithm discussed before is the presence of constraints and that we have to make sure these constraints are not violated. If we do not do it for reasons discussed in section 3.3 there might be more possibilities to construct an algorithm then if we have to check constraints. If we want to check constraints this has to be done in a distributed way. But usually a distributed algorithm only has access to information that is as local as possible while the constraints are network wide properties. In literature most research on creating trees with constraints is done in the area of networking. An example can be found in [11].

In literature algorithms usually try to give a result that is as far away from the constraint as possible, for example a delay that is as low as possible. What we actually want in our case is that the constraint is just met such that the tree can be optimized as much as possible.

Without checking the constraints the algorithm only differs from the previous algorithm in the fact that we can now make paths longer. This gives more freedom in choosing a new parent, but also the timing of applying the changes is important in this case. Further investigating and implementing this algorithm might be subject of future work.

### 4.3 Conclusion

In this section we discussed some initial ideas about changing the algorithms discussed in sections 2 and 3 into a distributed algorithm. In a distributed algorithm we do not want a node to have all the information of the network, but to make decisions based on information that is as local as possible.

Implementing Dijkstra's algorithm is done with the use of flooding, but the optimization algorithms are more of a challenge to implement. The initial observations and possible implementation approaches are discussed in this section.

Implementing, optimizing and analyzing distributed algorithms still has a lot of possibilities for future work.

## 5 Conclusion

In this report optimizations on the routing tree construction part of the Pareto analysis approach as suggested in [6] are investigated guided by the research questions of section 1.

**1. How do we construct a shortest path spanning tree (SPST) that has the lowest maximal node degree?** The initial approach of [6] uses an SPST as the basis of the cluster steps in the Pareto analysis. A routing tree construction algorithm that also keeps the node degree in mind could result in a faster execution of this Pareto analysis, because the node degree is an important factor in the run time needed for the Pareto analysis.

In section 2.2 we constructed an algorithm to reduce the node degree of a given SPST, with the use of related work in literature.

**2. What is the influence of constructing such SPST, compared with the original Dijkstra method?** In section 2.3 we have done some experiments with the algorithm that optimized a given SPST. We have seen some unexpected results with optimizing an SPST for random network (with unlimited node degrees). Some optimized trees actually required more run time for the Pareto analysis compared to the tree created by the original Dijkstra method. The main observation we made was that the root could not be optimized with the current algorithm and actually has a large influence on the final run time needed for the Pareto analysis.

Then we decided to drop the requirement of being an SPST and continue to optimize the tree given some constraints.

**3. How can we construct a routing tree that minimizes the node degree given information completeness and detection speed constraints?** In section 3 we looked at constructing an algorithm that optimizes an SPST given some speed and completeness constraints. First we focussed on checking the constraints and concluded that giving a good prediction of the constraints takes much run time and is difficult. A bad (in this case overestimated) prediction of the metrics reduces the freedom of optimizing the tree.

When fully optimizing the tree with the given algorithm without checking any constraint, we observed that the quality reduction was less than 10% while we got much more freedom to optimize the tree.

In section 3 we also took a more detailed look into the stop condition of the algorithm to check if the run time needed to optimize some nodes was worth the effort, in terms of reduction in run time needed for the Pareto analysis and the lost quality of the obtained Pareto points. We concluded that the lower the degree of the nodes we optimize the more time it took to optimize, while the reduction in run time needed for the Pareto analysis became less. Deciding which nodes we want to optimize depends on the run time we want to spend on optimizing and the implementation of the algorithm.

**4. What is the influence of using such minimized node degree routing tree compared with using a tree created with the original Dijkstra method and the SPST with the lowest maximal node degree?** Compared to the optimized SPST we see even more reduction of the run time needed for the Pareto analysis compared to the original method. We also do not have the problem of optimized solutions requiring more run time than the original solution

as was the case in the first algorithm that keeps the SPST.

What we lose with the method that fully optimizes the tree without keeping the SPST is some quality of the final Pareto points and the extra run time we need for the creation of the tree.

In section 4 we started with constructing a distributed version of the centralized algorithms discussed in sections 2 and 3. Suggestions are done on possible implementations and some problems are identified. The problems that arise in the distributed way we want to implement the algorithm for a wireless sensor network, mainly have to do with losing packets and making decisions based on not enough information. One problem we have seen is a situation which can lead to an unstable solution. More on the future work for the distributed implementations in the next subsection.

For the research done on the tree construction part we can conclude that the tree has a large influence on the run time of the Pareto analysis and that improvements on the current approach are possible to reduce this run time. Due to these optimizations more insight is gained in the influence of the tree on the Pareto analysis.

## 5.1 Future work

Throughout this report some suggestions for future work are given.

Starting from the beginning of this report we focus on implementing an optimization algorithm which improves an already given SPST. Further research could be done on creating an optimized tree given the initial network instead of running the Dijkstra algorithm first.

In this report we focus on optimizing the tree which results in improvements for the centralized and distributed run time of the Pareto analysis. What we observed in section 2 is that the algorithm mainly focusses on the centralized version and further research could be done to find more optimization possibilities especially for the distributed execution of the Pareto analysis.

All the algorithms for which experiments are done are implemented in Python. Implementing the algorithm from an interpreted code like python to a compiled code like C++, potentially gives a great reduction in run time needed to run the suggested algorithms. New observations about the usefulness of the algorithm can be made when both the optimization algorithms and the Pareto analysis are implemented in C++.

In section 4 we start defining distributed algorithms and identifying the problems involved. Some suggestions can be made for future work in this area.

First we have to notice that we based the distributed algorithms on an ideal model. Further research needs to be done on dealing with realistic situations in wireless sensor networks like losing packets.

The algorithms we suggested need to be implemented and analyzed. This is not done until now. There might also be more improvements for reducing the number of packets in the network and consequently the run time needed for the distributed algorithm.

We also identified the problem of an unstable solution, for which we suggested a solution, but further research needs to be done for this.

## References

- [1] Crossbow TelosB Datasheet:  
[http://www.xbow.com/products/product\\_pdf\\_files/wireless\\_pdf/telosb\\_datasheet.pdf](http://www.xbow.com/products/product_pdf_files/wireless_pdf/telosb_datasheet.pdf).
- [2] B. Awerbuch. Distributed Shortest Paths Algorithms. *ACM Symposium on Theory of Computing*, 1989.
- [3] P. Crescenzi and V. Kann. *A compendium of NP optimization problems*. 1998.
- [4] R. G. Gallager. Distributed minimum hop algorithms. *M.I.T. Laboratory for Information and Decision Systems*, 1982.
- [5] M. Geilen, T. Basten, B. Theelen, and R. Otten. An algebra of Pareto points. *Fundamenta Informaticae*, 78(1):35–74, 2007.
- [6] R. Hoes, T. Basten, C.-K. Tham, M. Geilen, and H. Corporaal. Analysing QoS Trade-offs in Wireless Sensor Networks. *MSWiM*, pages 60–69, 2007.
- [7] R. Hoes, T. Basten, C.-K. Tham, M. Geilen, and H. Corporaal. Configuring Wireless Sensor Networks Under QoS Constraints. *To Appear*, 2008.
- [8] S. Khuller, K. Lee, and M. Shayman. On degree constrained shortest path. *University of Maryland*, 2005.
- [9] R. Krishnan and B. Raghavachari. The Directed Minimum-Degree Spanning Tree Problem. *FSTTCS*, pages 232–243, 2001.
- [10] C. Lavault and M. Valencio-Pabon. A distributed approximation algorithm for the minimum degree minimum weight spanning trees. *Laboratoire d’Informatique de Paris-Nord*, 2007.
- [11] M. Terrovitis, S. Bakira, D. Papadias, and K. Mouratidis. Constrained Shortest Path Computation. *Advances in Spatial and Temporal Databases*, pages 181–199, 2005.
- [12] J. Zhanfeng and P. Varaiya. Heuristic methods for delay constraint least cost routing using shortest paths. *IEEE Automatic Control*, 51:707–712, 2006.
- [13] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.