Investigating the Effect of Adding Additional Costs on the Optimal Solution and Performance of a Wireless Sensor Network Optimisation Model

Steven Simpson

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Aberdeen.

Department of Computing Science

2021

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: [illegible]

Date: 2021

Abstract

In this paper, an optimisation model based on column generation for maximising the lifetime of a wireless sensor network (WSN) is adapted to consider scenarios that may be encountered in real-life applications. Related work is reviewed to identify variations of WSN optimisation models and the many costs that are taken into account in each of them, before the proposed changes are made. An experiment is conducted that measures the impact of two costs, on the performance and solution found of the models. It is found that increasing the energy consumption rate for inactive sensors has no significant impact on the performance of the models, but can significantly reduce the solution found. Moreover, the alteration of the models to take into account a cost for retrieving data from one or more targets on a sensor's battery not only had a notable impact on the solution value, but also greatly impacted the performance both in time taken to find a solution, and in the number of iterations the approach required to terminate. An approach for adding a cost for activating a sensor is proposed but is not experimented on in this paper. Finally, a tool for visualising the optimised WSN solutions is developed and presented, that could help researchers in future work, to understand the covers generated, the impact of changing parameters, and to assist in debugging the implementation of WSN optimisation models.

Acknowledgements

I would like to thank everyone who has assisted me in anyway throughout the completion of this project. In particular, thanks to my supervisor Dr. Bruno Yun for his guidance and advice, and to my fiancée Leah, for her constant encouragement and support. Finally, thanks to all in the Department of Computing Science at the University of Aberdeen for the past 4 years. I will always be grateful for the experience and knowledge each of you helped to provide me with.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Wireless Sensor Networks (WSN) have become increasingly popular over recent years due to their applicability to a wide range of real-world problems [2]. They are commonly used in problem areas such as natural disaster prevention and area monitoring amongst many others and are considered to be greatly effective overall despite the many challenges that can come with their design and implementation [23]. Therefore, it is worthwhile to explore how these sensor networks can be utilised to their full potential in order to cut the network operating costs and improve the accuracy of the data they retrieve.

One of the main challenges that is faced when implementing a sensor network is the limited lifetime that they can actively retrieve and relay information for. Each sensor that is involved in the role of the network is fitted with a battery containing a finite amount of energy and so it is crucial that the network considers this factor when assigning the role that the sensors will play and for how long [29].

Furthermore, this challenge is interesting because if there is a lack of consideration of this factor when designing a WSN solution, it could not only result in unnecessary spending on replacing the batteries when they are depleted, but may also mean that the problem is infeasible. This is since many of the applications of WSN are in dangerous scenarios where one cannot safely enter the area in which the network operates, for example, in monitoring areas where toxic gases are present as seen in [31]. In some other applications, wireless sensor networks can be used to track species of wildlife. The idea of replacing the batteries of a sensor can also be a problem here, as any human entering the area where the sensor network is deployed could be detrimental to the work being carried out should the wildlife be disturbed [30].

These same applications pose another challenge: how can these sensors be deployed in their required location in the first place if the area in which they operate is not safe? The most common answer to this question is by airdropping the sensors into the area [11]. There are, however; drawbacks to this approach. The unpredictability of this method means that we cannot choose the way in which the sensors communicate with each other before deployment as we cannot assume the exact distance between two sensors and consequently whether or not they will be within their communication range. We also cannot pre-determine the roles that the sensors will play at specific times and therefore, we must find a method that given the eventual coordinates of the sensors, can optimise the lifetime of the network by producing sets of active sensors that can adequately cover a number of required targets before allocating time to each optimally.

1.2 Research question

In [7], an approach to optimising the lifetime of a sensor network is modelled and solved using several linear programming techniques such as branch-and-cut based on Benders' decomposition that was first proposed in 1961 [3]. Specifically, the problem is solved three times by a column generation approach but each time with a different variation on the pricing problem for proposing new interesting covers where a cover is defined as a partitioning of the sensors into three defined roles such that all targets are adequately covered. Integer linear programming, branch-and-cut, and constraint programming methods are proposed before the subsequent performances of each of the techniques are compared and analysed. It is found that the branch-and-cut and constraint programming approaches perform best in reducing the computational solving time. Despite this, the integer linear programming method benefits from the many impressive commercial solvers that can be used for optimising the problem and is the most feasible for this research given the short time constraints and available resources. This is especially important since attempts at contact with the authors of [7] resulted in no replies, and so no information on the implementation of these models is provided and I must therefore research, learn and apply the techniques myself.

Whilst [7] features a thorough experiment with sufficient instances for testing, there are a few flaws in their research of how the lifetime of a sensor network can be optimised. Firstly, for their experiments, it is assumed that an inactive sensor consumes a negligible amount of energy. Whilst it may be fair to say that inactive devices in some settings consume little to no energy, it is often not the case [4]. It is a particularly unsafe assumption in a setting where a network may be operating in extreme temperatures or abnormal environments. Therefore it should be investigated how an inactive sensor that consumes energy from its battery, may affect the lifetime of the network. Thus, the first research question is how can differing rates of energy consumption for inactive sensors have an effect on the lifetime of a sensor network?

In this same paper [7], it is suggested in the conclusions that an interesting pursuit for future work is in modelling how a cost can be added to represent the energy consumption that may occur from a sensor establishing a connection with a target in order to retrieve data from it. I believe that this is an important issue since the model explained in the paper does not feature any additional cost for a sensor that is actively sensing any amount of targets. In a real scenario, it is clear that as a sensor becomes responsible for getting information from more targets, it will consume more energy since there will of course be more data that must be retrieved as evidenced by a study into denial of service attacks on wireless sensor networks that concluded that by simply giving a sensor too many tasks, the battery of that sensor could be drained [17]. For this reason, my second research question is; "how does an additional cost for establishing a connection between a sensor and target impact the lifetime of a sensor network?"

Finally, the model proposed in the paper is unclear on what state specifically an inactive sensor is in. In other words, it is not explicitly stated whether or not a sensor in an inactive state must reboot completely when made active and re-establish connections with the sensors around it in order to communicate as well as ready itself for possibly monitoring targets. In fact, in [27], the energy consumption of wireless multimedia sensor networks is analysed and it is found that the overhead of booting up a sensor can be reduced, but is still an inevitable cost. For this reason, it is a worthwhile pursuit to explore how this cost could be modelled and implemented.

Chapter 2

Background

2.1 Linear programming

2.1.1 Fundamental concepts

In many research areas for solving real-life problems, mathematical models are designed to represent a simplified version of the problem. This is done in order to run virtual experiments using these models and make predictions on how any proposed solution could fare if it were to be applied to the real-world scenarios. Moreover, mathematical models are often developed with some variable or function that the maximum possible value of which is desired (called the objective function), subject to some inequalities called constraints. These are called optimisation models and are especially useful in areas where the best possible solution must be found but must also respect some rules or constraints that limit what this optimal solution can be [16].

When a mathematical model is designed in such a way that all constraints and the objective function are linear, the problem can be classified as a linear programming problem. An example of linear programming problem's constraints can be seen below where the objective is to maximise $x+y$.

$$\begin{aligned} x+2y &\leq 10 \\ x-y &\geq 3 \\ x &\geq 0 \\ y &\geq 0 \end{aligned} \tag{2.1}$$

In this example, some varieties of constraints are shown. The first constraint is an example of a resource limitation constraint. That is, a constraint that limits the possible solution to not exceed the amount of resources that are available, in this case, 10. The next constraint is an example of a supply and demand constraint. This type of constraint describes a relationship between the variables where the value of one must be defined in terms of the other plus or minus some additional resource [25]. In this case, x must be at least the value of $y+3$. Finally, the last two constraints are examples of bound constraints. Specifically, they are non-negativity constraints that ensure both variables in the example are equal to or greater than 0. Non-negativity constraints are essential for many problem formulations where the variables represent real-life values that cannot be negative.

In many cases, linear programming problems are far more complicated than the example given and contain many variables but for this simple example, a graph can be created to show the effect the constraints have on the available solutions. The constraints of the problem can be

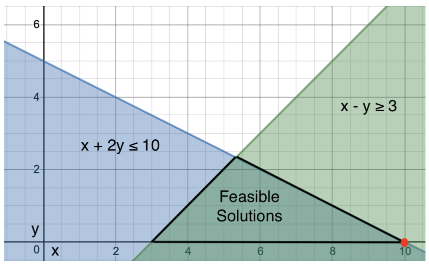


Figure 2.1: A graphical representation of an example linear programming problem.

displayed as lines on the graphs and the area of said graph in which all of these constraints are satisfied, is the area where the feasible solutions can be found. Furthermore, by what is known as the fundamental theorem of linear programming, it can be said that the optimal solution must be found on a corner or vertex of this enclosed region [33].

Thus, there are three possibilities for what is the optimal solution. The bottom left corner where $x = 3, y = 0$ gives an objective value of 3. The bottom right corner where $x = 10, y = 0$ gives an objective value of 10 so the bottom left corner can already be discarded as a potential optimal solution. Finally, the top corner coordinates are $x = 5.33, y = 2.33$ and so this objective value is 7.66. Therefore, the optimal solution is found when $x = 10$ and $y = 0$. This is essentially how optimisation algorithms work but often to a much larger scale with more variables and constraints, making graphical representations of the models difficult. There are some common algorithms that have been developed to efficiently solve linear programming problems such as the simplex method and barrier algorithm amongst others, each varying in their approach to finding the optimal solution [35].

2.1.2 Duality

The theory of duality comes from the idea that any linear programming model can be converted to another model (called the dual) such that all variables in the original program (in cases where dual programs are considered, this is called the primal program) are constraints in the dual and similarly, the constraints in the primal program are made to be variables in the dual problem. Finally, the objective of one problem should be the opposite of the other. In other words, if the primal problem's objective is to maximise some function, then the dual problem will aim to minimise its objective function.

The motivation for retrieving the optimal solution for the dual problem is that it obtains information about how each of the decision variables can be influenced by given constraints [24]. This means that each variable is given a value based on what improvement could be made to the solution should the constraint be relaxed somewhat. For example, if a resource constraint is defined (a constraint where the variables must be less than or equal to some value representing an available resource), the dual problem can be used to obtain what improvement to the variables could be made if that resource was increased by one unit. Moreover, it can be noted that these values represent how much of an impact a specific constraint of the primal problem can have on improving the optimal solution, and the lower the value, the less this resource was utilised by the optimal solution [12]. A variable of the dual problem is called a dual variable of shadow price [24].

For the example (2.1) in the previous section, the corresponding dual problem is given below in (2.2) where the optimisation function is to minimise $10a - 3b$. The variables in this problem represent the main constraints from the primal problem (that is, the constraints not relating to non-negativity) whilst it can be seen that the constraints of this dual problem are defined by inequalities from the values in the primal problem. For more information on this topic, see [24].

$$\begin{aligned} a - b &\geq 1 \\ 2a + b &\geq 1 \\ a &\geq 0 \\ b &\geq 0 \end{aligned} \tag{2.2}$$

2.1.3 Column generation

In many linear programming problems, a huge amount of decisions have to be made and possible solutions considered. This can result in unreasonable amounts of time being taken to find the optimal solution. There is, however, a way around this. Column generation is a technique that is regarded as a highly useful tool for solving linear programming problems and is used very often, especially in larger problems [10].

The idea behind column generation is that in problems with a large amount of decision variables, the optimal solution found often has many variables that do not contribute and could have essentially been discarded. In other words, many of the variables are set to 0 and are irrelevant to the optimal solution. Therefore, it could be interesting to create an algorithm that will generate only the variables that have a chance of improving the solution and are likely to be used in the final solution. This is what column generation does.

The process of column generation begins with solving the main linear program model (in column generation terms, known as the master problem) with only a small subset of all possible variables. Then, the dual variables from this solution can be used to determine how well the resources or constraints are being utilised in that solution. The dual variables are passed to what is called the subproblem, which is a model designed to produce interesting variables that should be considered in the master problem. This subproblem (sometimes called the auxiliary problem) aims to find variables that have some kind of reduced cost, the definition of which depends on the problem [1]. Regardless, the subproblem is solved and one or more solutions that reached a specific threshold that defines when a subproblem solution has potential to improve the master problem solution based on the current dual variables, are added to the subset of variables that are used in the master problem. This process will repeat until no interesting variables are found by the subproblem, and it can then be said that the optimal solution has been found [15].

2.2 Wireless sensor networks

Wireless Sensor Networks are composed of nodes that can monitor areas and communicate with each other when necessary. Each node (or sensor) is composed of a sensing subsystem, a processing subsystem and a wireless communication subsystem that when put together, can collect information before making effective decisions on what to do with that data. As mentioned previously, each node also contains a battery to supply the required energy and so the sensors are provided with a finite power source.

In my research, I consider the problem of target coverage within a specified area. This means that sensors are deployed to retrieve information from targets at some coordinate that is within the defined sensing range of a sensor. This information can then be communicated to other sensors within another specified range called the communication range. This range may not necessarily be equal to the sensing range. All information must eventually be transmitted to the base station through multi-hop communication, that is, information may be transmitted through multiple sensors before finally reaching a sensor that is within the communication range of the base station. It should be noted that whilst specified ranges are used in this project, in real-life scenarios, these ranges cannot always be defined accurately as sensors may have connectivity issues with other nodes for a variety of reasons, including external factors outwith the control of the network.

In this section, the variables and mathematics behind the problem will be introduced. Many of the elements used in [7] are used similarly here and so this section will aim to explain the techniques that were used in the cited paper clearly and how their formalisation of the problem is relevant for the research I am carrying out. The improvements and suggestions that I have for the model to better fit my research question will be discussed in the following sections as well as the technologies that are required to carry out the experimental study.

Let $S = \{s_1, s_2, ..., s_m\}$ be a set of all sensors where m is the total number of sensors, each of which is given random coordinates in an area of a specific size for monitoring. The goal of this sensor network is to retrieve the required information and communicate it to the base station $s_0 \notin S$ through multi-hop communication amongst other sensors if necessary. For this problem, we take n targets that are represented by random points in the area that must be monitored by the network. We can denote the set of targets by $K = \{k_1, k_2, ..., k_n\}$ where usually $n < m$. In order to properly sense and relay the data from the targets, we must set three specific roles that a sensor can play in the network. We say that a sensor is a source sensor if it is actively seeking and retrieving information from a target within its sensing range $R_s > 0$. The sensors that take up this role can not only retrieve the information from the targets but they can also relay data to other sensors and the base station. The drawback of having sensors carry out this role is that it consumes the most energy in most instances of the problem. A sensor can instead be given the role of being purely a relay sensor. This type of sensor cannot retrieve information like a source sensor does but can be used in communicating information towards the base station by transmitting data to nodes within its communication range R_c. The energy consumption of a sensor operating in this role varies depending on the type of problem. Finally, a sensor can set to be inactive. Sensors in this role cannot be used for sensing or relaying information but do consume much less energy. As mentioned earlier, [7] assumed that an inactive role consumes a negligible amount of energy but this will not be the case for this research.

The chosen energy consumption rates for each of the sensor roles are imperative in what specific lifetime problem we are dealing with. We can denote the energy consumption rates of a source, relay, and inactive sensor by E_s, E_r, E_i respectively and the set $\varepsilon = \{E_s, E_r, E_i\}$ holds each of these consumption rates. E_s is typically given the largest value in ε but there are some instances of the problem where this is not the case. If we take $E_s \leq E_r$, then the relay sensor role is made obsolete as there is no drawback to choosing that a sensor should be a source node that can perform more duties for less cost. These specific problem instances will be discussed further

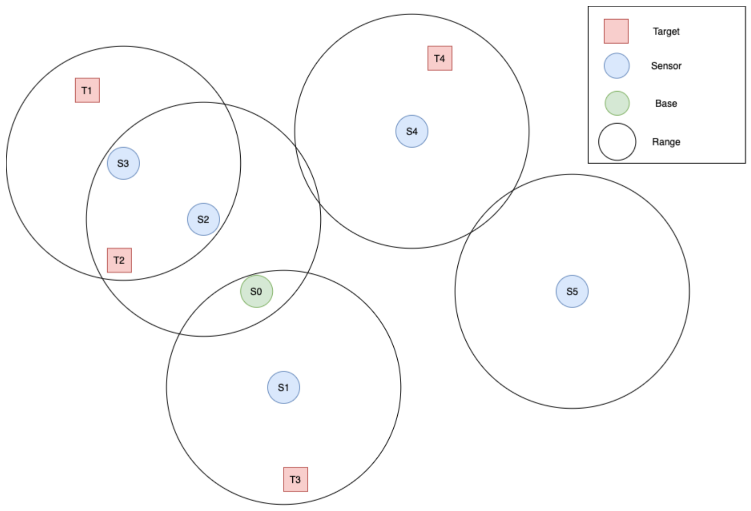


Figure 2.2: A simple example of a sensor network where the sensing and communication ranges are equal.

in this report and will be experimented on in order to test the proposed solution thoroughly.

In order to find an optimal solution for any proposed model, the program must be provided with every connection that can be made between nodes. In other words, we must find a way to represent all of the possible connections with respect to the sensing and communication ranges for a particular instance. We can do this using a directed graph $G = (N, A)$ where the set of nodes is defined as $N = S \cup K \cup \{s_0\}$, the union of the set of sensors, the set of targets and the base station. We can then define the arcs, denoted by A, of the directed graph by the connections of nodes that fit at least one of the three following conditions. An arc denoted by $a(u, v) = (u, v) \in A$ exists if (1) $u \in K, v \in S$ where the coordinates of u are within the sensing range of R_s for v, (2) $u, v \in S$ and the coordinates of each sensor are within the communication range of the other or (3) $u \in S, v = s_0$.

If we look at Figure 2.2, we can create the list of arcs for this network. The set of arcs by the first arc rule is $\{(T1, S3), (T2, S3), (T2, S2), (T4, S4), (T3, S1)\}$. Then, the set of arcs that fit the second rule is $\{(S2, S3), (S3, S2)\}$ and then for the third rule $\{(S1, S0), (S2, S0)\}$. From these three sets, we have the complete set of arcs $A = \{(T1, S3), (T2, S3), (T2, S2), (T4, S4), (T3, S1), (S2, S3), (S3, S2), (S1, S0), (S2, S0)\}$. From this set, it can be seen that this network cannot feasibly complete its task as there is no arc from S4 to any other sensor or the base station. This is a problem as the only arc existing from T4 is to S4. Once the information from T4 is retrieved, it has nowhere else to go and cannot be transmitted to the base station as required.

We can represent a single cover set by partitioning S into three disjoint sets. We call this partition P_j and the three corresponding sets S_s^j, S_r^j, and S_i^j for the source, relay, and inactive sensors respectively. We then define $N_j = N \setminus S_i^j$, the set of all nodes except inactive sensors (the active network) in a partition P_j, and $G[N_j]$ as the subgraph of G induced by N_j [7]. We can say that for every target in K, there must exist a path from that target to s_0 in the subgraph $G[N_j]$ in order for P_j to be a feasible partition for the problem. In other words, for every target, there must be some way to reach the base station by transmitting data along active sensors that are within the range of the node sending the information. An example of a partition or cover of the sensor network is shown in Figure 2.3. From this cover, we can observe that $S_s^j = \{S1, S3, S4\}$, $S_r^j = \{S2\}$,

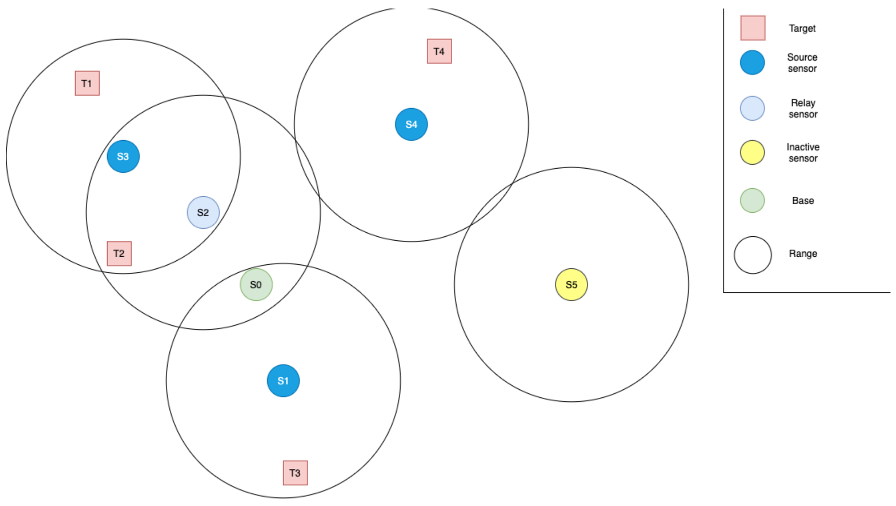


Figure 2.3: An example of a cover in a sensor network.

and $S_i^j = \{S5\}$. $N_j = \{S1, S2, S3, S4\}$ and so we can see that for each target, there must be a path to the base station through N_j for the cover to be feasible. Target 1 is sensed by S3, relayed to S2 and then to the base station S0. Target 2 follows the same path. Target 3 is sensed by S1 and directly transmitted to the base station. Finally, target 4 is sensed by S4 but does not have another vertex (sensor) that it can move to in the path. Therefore, this cover is not feasible.

As mentioned earlier in this section, there are different types of the problem that we can find solutions for. The two main types that we will tackle in this research are the maximum network lifetime problem with role allocation and connectivity constraints (referenced here on as CMLP-MR) and the connected maximum network lifetime problem (CMLP). CMLP-MR involves finding the maximum lifetime of a sensor network where there are specific roles that a sensor can be allocated. This is done by allocating a time $t_j \geq 0$ to each feasible partition $P_j \in \Omega$ where Ω is the set of all feasible covers for the given sensor network. The allocated times must also take into account the battery capacity for each sensor. CMLP is a specific case of CMPL-MR where $E_s = E_r$ and so it could be said that there is no role allocation for this type of problem, only the decision of whether a sensor is turned on or off.

We now have most of the information that we need to build a model that can optimise the lifetime of a sensor network but there is another set of key variables that will be required in order to do this. We have binary role variables $y_{s_u \ell j} \in \{0, 1\}$ for every sensor s_u in a partition $P_j \in \Omega$ where $\ell \in \varepsilon$, that represents whether or not a sensor has taken up a specific role in that partition, and consequently determining the amount of energy that a sensor will consume for that partition when multiplied by the amount of time allocated to P_j. If the value of $y_{s_u \ell j}$ is 0, then the sensor does not take up that role in the partition, but if 1, the sensor has been allocated that role for the partition.

If we assume a situation where we know all possible covers that are feasible for the problem, our binary role variables will also be completely known and we can find an optimal solution for the network by Model M1. Equation (1) in Table 2.1 aims to maximise the sum of the time allocated to every feasible partition, effectively maximising the lifetime of the network. This is subject to Equation (2) that enforces the constraint that no sensor s_u can exceed its specified battery capacity

Model M1:	Maximum network lifetime problem		
Maximize:	$\sum\limits_{j \mid P_j \in \Omega} t_j$		(1)
Subject to:	$\sum\limits_{j \mid P_j \in \Omega} \left(\sum_{\ell \in \varepsilon} \ell y_{s_u \ell j}\right) t_j \leq b_{s_u}$	$\forall u \in S$	(2)
	$t_j \geq 0$	$\forall j \mid P_j \in \Omega$	(3)

Table 2.1: Maximum network lifetime problem

b_{s_u} and Equation (3) that ensures non-negativity for each of the allocated times.

This model will eventually find an optimal solution for every instance, but there is the problem of performance, especially if there is a time constraint for a given application. In fact, for CMLP-MR problems, there are 3^n possible partitions. For sufficiently large instances with many sensors, it is simply not always possible to know every feasible partition at the time of deployment. It would take a very long time to calculate this, and thus an exceptional amount of total time would be needed to not only calculate every possible cover, but to allocate time to each of them optimally. For this reason, it is crucial to the success of the network that we can quickly produce the covers that are worth considering for the maximum network lifetime problem without considering covers that are unlikely to improve the solution. This is an especially interesting proposal because the final, optimal solution is unlikely to use many covers with non-zero time and it is wasteful to consider every possible partition when most will never be used in the optimal solution. For this research, I will implement a column generation algorithm based on the pricing subproblem described in Model M2 (Table 2.2) with some adjustments to carry out this process.

The pricing subproblem model will take the optimal set of dual variables $\pi_v \forall v \in S$ from Model M1 that represents the battery constraints defined in the master problem for every sensor. These dual variables are found by solving the dual problem of M1 that aims to make best use of the available battery life such that the amount left is minimised. M2 then uses these variables to produce new covers that can be added to the set of partitions that are allocated optimal times using M1. There is the issue that the pricing subproblem requires dual variables from the maximum network lifetime problem but the maximum network lifetime problem requires covers to work with. To resolve this, we initialise the pricing subproblem with $\vec{\pi}$ values set to 0, that will result in all possible covers being considered but we will only take a specific amount of them to begin with. This will give us a set $\Omega' \subset \Omega$. The role of the pricing subproblem will then be to add more interesting covers to our set Ω' before the process is restarted and M1 is called. If no potential improvements are found, then the program stops and we know that the solution given by the most recent solution found by M1 is optimal.

It can be seen in Table 2.2, Equation (4) that the objective of this model is to minimise the dual variables multiplied by the cost of their role in a proposed cover. It is clear that the maximum value of this function is 1 but any cover that results in a positive objective value is considered interesting. As described in [7], the constraints (5)-(7) define the rules for how flow is created by sensing targets and relayed to the base station through other sensors. Flow variables x_{uv} are defined for all node relationships and contain the total amount of data being passed from u to v. These

Model M2:	Pricing subproblem		
Maximize:	$1 - \sum_{v\in S}\sum_{\ell\in\varepsilon} \pi_v \ell y_{s_u\ell j}$		(4)
Subject to:	$\sum_{u\in S \mid \exists a(v,u)} x_{vu} = 1$	$\forall v \in K$	(5)
	$\sum_{u\in N \mid \exists a(v,u)} x_{vu} - \sum_{u\in N \mid \exists a(u,v)} x_{uv} = 0$	$\forall v \in S$	(6)
	$\sum_{u\in N \mid \exists a(u,s_0)} x_{us_0} = \lvert K\rvert$		(7)
	$x_{uv} \leq \lvert K\rvert \left(y_{s_vE_rj} + y_{s_vE_sj}\right)$	$\forall u,v \in S \mid \exists a(u,v)$	(8)
	$x_{vu} \leq \lvert K\rvert \left(y_{s_vE_rj} + y_{s_vE_sj}\right)$	$\forall v,u \in S \mid \exists a(v,u)$	(9)
	$x_{uv} \leq y_{s_vE_sj}$	$\forall u \in K, \forall v \in S \mid \exists a(u,v)$	(10)
	$\sum_{\ell\in\varepsilon} y_{s_v\ell j} = 1$	$\forall v \in S$	(11)
	$x_{uv} \in \mathbb{Z}^+ \cup \{0\}$	$\forall u,v \in N$	(12)
	$y_{s_v\ell j} \in \{0,1\}$	$\forall v \in S, \ell \in \varepsilon$	(13)

Table 2.2: Pricing subproblem

equations ensure that each target can only pass a unit of flow to one sensor (5), even if multiple active sensors are within the targets range, ensures that a sensor must pass every unit of flow that it receives (6), and ensures that the base station eventually receives $|K|$ units of flow (7). Next, equations (8)-(10) define the constraints for which sensor roles can receive and pass on flow. In order, Equation (8) ensures that the sensor receiving flow from another sensor is either a relay or source node whilst (9), in a similar way, ensures that a sensor passing on flow to another sensor is also one of these two roles. Equation (10) constrains flow being passed from a target to a sensor to only exist when that particular sensor is a source node. In order to ensure that a sensor only takes up one of the specified roles, constraint (11) is defined. This constraint ensures that the sum of the binary role allocation variables for a sensor is equal to one. Finally, constraints (12) and (13) ensure that the variables in the subproblem are of the correct types. The flow variables are constrained to be non-negative whilst the binary role allocation variables are of course constrained to be only 0 or 1.

2.3 Related Work

In this chapter, the work that has previously been conducted in optimising the lifetime of a sensor network is reviewed. The papers that are analysed feature many additional costs that have been implemented in their models for lifetime optimisation. Furthermore, work that has been completed on optimisation models designed for slightly different problem areas or with slightly different properties are described. Whilst few papers exist that directly investigate the impact of adding new costs to existing models, conducting a review of papers that aim to use the battery life of the sensors efficiently can still be beneficial to the project. By conducting this review of the existing literature, I can make decisions on how best to approach the alterations I wish to investigate and review what effect some other additional costs had on the performance of optimisation models for wireless sensor networks and the eventual solution they find.

In [37], authors Dimitrios Zorbas and Christos Douligeris review many of the problem areas and approaches associated with optimising how limited batteries can be utilised in wireless sensor networks. The distinction between centralised and distributed approaches for finding an optimal number of cover sets are discussed. The former approaches propose carrying out the necessary calculations before deployment and sending the resulting instructions to the sensor network to execute. This approach can be beneficial in that the sensors only have to focus on their specific tasks of sensing and relaying information but in order for this kind of approach to be successful, the exact locations of the nodes must be available for use in the optimisation algorithm. Contrarily, the latter approaches propose that calculations are performed after deployment by communication between sensors, benefiting from robustness as changes can be made when necessary if something goes wrong, but of course, at the expense of the battery life of the sensors. This paper also makes a distinction between disjoint and non-disjoint cover sets. In disjoint cover sets, a sensor can assume only one role whilst this is not a restriction in non-disjoint covers and it is concluded that in many cases, non-disjoint cover sets result in a better overall network lifetime but often with a higher time complexity [37]. Finally, the paper briefly discusses some other variations on the problem. The ideas of having sensors with adjustable ranges and mobile sensors are put forward as well as problem instances where not all targets need to be covered, the latter of which involves an objective of covering a specified percentage of the given targets whilst optimising the lifetime of the network. Zorbas and Douligeris conclude that in order to use the limited energy of the batteries efficiently in target coverage problems, the most feasible approach is to find a way to discover the maximum number of cover sets for the instance.

A paper written by the same authors that designed the optimisation models that this research is based on considers using multiple base stations to improve the total time that a sensor network can operate for [8]. The experimental design in this paper is slightly different to what is found in [7] as instances with smaller numbers of sensors are used to test their hypothesis. The paper focuses more on the performance of their approach of using column generation with a greedy search algorithm and find that it performs well. No clear conclusions are reached on the impact that multiple base stations had on the final solution but nonetheless, this is an interesting idea for an addition to the problem.

In research conducted by Muhammed Emre Keskin, the additional cost of transferring data to a base station (also known as sinks) is considered [20]. This paper criticises much of the work done on attempting to extend the optimal lifetime of sensor networks for proposing ideas without considering the expense of implementing such an idea. I tend to agree with this criticism as very few papers consider the feasibility of their model being applied to real applications of sensor networks but instead try to design new models that will increase the optimal solution but only because no cost is implemented to keep these new ideas restricted and fair. This paper examines problem instances with multiple base stations that are mobile in order to remove themselves from locations where the surrounding sensors have depleted batteries. The proposed model adds an additional cost in the objective function that takes into account the cost of using the base stations. Binary variables represent whether or not the base stations are used and are summed up before being multiplied by the additional cost rate and taken away from the total time that can be allocated.

In [5], the idea of one-time costs is presented. This paper investigates how expensive operations that must be carried out once by a wireless sensor network can reduce its potential lifetime. Some examples of one-time costs are given such as establishing a secure cryptography key between nodes or applying a software update to the network. The experiment in this paper applies public key cryptography as a one-time cost, citing its frequent usage in the research area and finds that contrary to the popular belief in previous research, the impact of a one-time cost is not something that can be discounted, especially in larger networks where this cost must be applied to all nodes in the network. This again, raises the important issue of ensuring that mathematical models do not underestimate costs or assume that they are negligible, if they are to be applied to optimising real networks.

In [36], a limitation is placed on the number of times information can be relayed amongst sensors before reaching the base station. In other words, the number of hops is restricted. This paper presents two models. The first model is used to minimise the number of hops that can be made in the network subject to the constraints that flow (or data) must reach the base station where each piece of information being relayed is one unit of flow. The second model aims to maximise the lifetime of the network with this calculated restriction on the number of hops taken into account. The results of the experiment that was conducted found a significant decrease in total lifetime when the minimum hop restriction was used compared to when no such restriction was in place. On the other hand, it should be mentioned that when this minimum hop restriction was increased slightly, the reduction in lifetime was found to be fairly small when compared with cases where there was no restriction.

It is clear from the papers examined that there is clearly a need for model accuracy to be considered more than it currently is, if the models are ever to be used for real applications of wireless sensor networks. Many papers design new models and algorithms for finding optimal solutions in the fastest possible time but the realism of these models are not always prioritised, rather the theoretical result achieved. It is therefore crucial that more work is put forward that takes existing real-world energy consumption costs into account. This project will aim to do this by experimenting with how more costs can be modelled and the impact they have on both the performance of optimisation algorithms but also the optimal solution that is found.

Chapter 3

Optimizing the Lifetime of a Sensor Network

3.1 Problem description

The original models that were created to optimise the lifetime of a sensor network in [7] sufficiently covered many different constraints that are crucial for applicability to real sensor networks. Each generated cover takes into account the battery constraint of the sensors, the different roles that a sensor can play in the network and the particular ways in which data must be transmitted to the base station. The model M1 accurately represents how a set of covers can be allocated appropriate durations in which they will operate to keep a sensor network active optimally. Despite this, there are some criticisms that could be made of their model and subsequent experimental study that this research aims to tackle.

Firstly, the pricing subproblem fails to recognise that there must be some cost for a sensor being given the task of retrieving large amounts of data from multiple targets. For example, if we take their current column generation and integer linear programming approach, and imagine two scenarios, both of which there are ten sensors that must cover ten targets. In the first scenario, lets say that one target lies directly in the sensing range of one sensor, that is, each sensor covers exactly one target each. In this scenario, every sensor can communicate directly with the base-station without the need for multi-hop communication. An example of the sensing duties of one sensor in this scenario can be seen in Figure 3.1. In the second scenario, all ten targets lie within the sensing range of a single sensor called S1 and outwith the sensing range of the other nine. In this scenario, S1 must retrieve and transmit an amount of information equal to ten times more than it must work with in scenario one. The role of sensing targets that is given to S1 can be visualised by Figure 3.2. It is fair to say that if these scenarios were applied to a real life example, this sensor's battery would be depleted far sooner in scenario two than in scenario one, either directly as a result of having to carry out more processes or from the sensor starting to overheat, however; the models designed in [7] would result in the exact same rate of depletion for this sensor's battery in both instances. The models must be adapted to take the number of targets that a single sensor must cover into account and how this can impact that sensor's battery life. It should be investigated what type of effect this additional cost can have on the optimal solution found by the models.

Furthermore, whilst the model in [7] appropriately considers the possibility of a non-negligible energy consumption rate for inactive sensors, this is never experimented with in their study. This lack of analysis is detrimental to the potential applications of the models to a real scenario as it cannot be guaranteed that inactive sensors will always consume negligible amounts

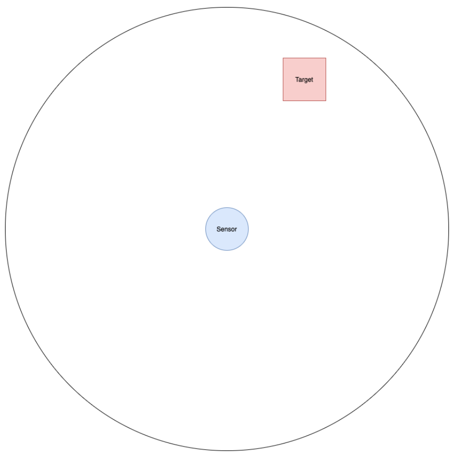


Figure 3.1: A sensor's role in the first scenario (base station not included in diagram).

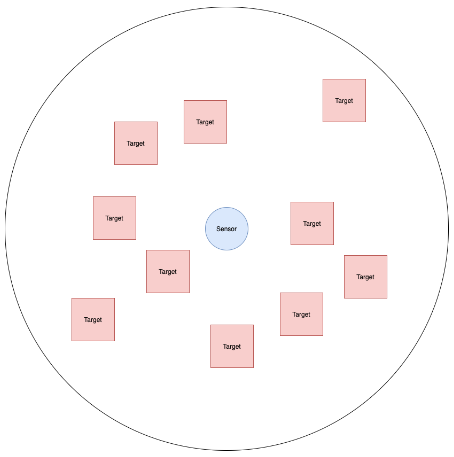


Figure 3.2: Sensor S1's role in the second scenario (base station not included in diagram).

of energy. The ability of a battery to hold capacity can be influenced by not only the role of the device it is powering but also by other factors such as the temperature that it is operating in and for some battery types more than others, the concept of self-discharge can have an impact over time [22, 13].

3.2 Proposed solution

Firstly, the idea of adding a cost to activating a sensor that was inactive in the previous cover should be explored. The first problem that I encountered when considering this additional cost is that it is not as simple as checking each sensor's state in the previous cover and in the current cover to evaluate whether or not it has been activated, as the list of covers are not necessarily ordered. If a perfectly ordered list of covers was given to the master problem in such a way that sensor activations were minimised, then this cost could be modelled by checking for each partition and then for each sensor, the role allocation variable for the inactive state in partition P_{j-1} and then the same for partition P_j. If the former value was equal to 1 and the latter equal to 0, then we know that an activation has taken place and some defined cost rate can be applied.

Unfortunately, as mentioned, the list of partitions is not ordered to minimise these activations. I decided to explore the option of adding a third component to the column generation approach, that would order the list of partitions each time new covers are added by the pricing subproblem, and before the next iteration of the master problem. This model would minimise the number of sensor activations subject to a number of constraints. The idea was that these constraints would set rules on how indices of the new array of covers were allocated to the covers from the unordered array. The problem with this approach, is that it is clear that this model would add to the overall complexity of the program a very large amount. The complexity of this model itself would be worst case $\mathcal{O}(n^2)$ where n is the number of covers at that stage in the program. Therefore, I decided not to implement and experiment on this possible change because in order to obtain meaningful results, the planned time for experimentation would have to be extended excessively and could make the entire project infeasible.

It is clear that the models M1 and M2, as they are, can be considered sufficient for testing the research question of how inactive sensor costs can impact the optimal solution. This can be experimented on simply by adjusting the value of the defined variable E_i, but the model must be adapted to carry out an experimental study to find the effect that an additional cost on a sensor's battery for retrieving data from targets can have on both the final solution found and also on the performance of the optimisation models. The way that a solution for this problem can be represented must be discussed as there are various ways it could be approached.

Firstly, it is important to note that the solution must consider a cost for sensing the targets rather than a limit on how many targets one sensor can cover. In other words, the goal is not to restrict the sensors by limiting them to covering a specified number of targets but simply applying a cost for doing so. If the results from the experiment do indicate that the additional cost for retrieving data from multiple sensors has a drastic impact on how long a sensor network can operate for, then a constraint in the pricing subproblem could be considered to restrict a sensor to only cover z targets where $1 \leq z \leq |K|$. This could be represented by the additional constraint below.

$$\sum_{v\in K|\exists a(v,u)} x_{vu} \leq z \quad \forall u \in S \tag{3.1}$$

It should also be clarified that this problem specifically aims to model a cost for retrieving data from targets but not for communicating this data to other sensors or to the base station. The additional cost should only apply to sensors that are allocated the source role and are actively receiving flow from one or more targets. For this reason, one cannot simply find the summation of the flow variables x_{uv} for each sensor v where $u \in N$. The solution must appropriately find the flow for each source sensor such that only nodes that have an existing arc with the relevant sensor are considered and only sensing flow should be considered, not flow that represents communication between sensors.

The first change that must be made is in the maximum network lifetime problem model (M1). There is no change that needs to be made to the objective function here but constraint (2) must be adapted slightly. Let α be the defined battery cost rate that is defined for monitoring one or more targets and $\lambda_{s_u j}$ be the number of targets that a sensor s_u is responsible for monitoring in a cover P_j. The value $\lambda_{s_u j} = \sum_{v\in K|\exists a(v,u)} x_{vu}$ is formed by using the flow decision variables between each target and the sensor, from the pricing subproblem for that cover. Then we can define the new constraint as shown in (3.2). Keep in mind that this constraint will add an additional cost to a sensor monitoring even just one target, as the idea is we want to separate the cost of supplying power to a sensor to stay active and the cost of retrieving data from targets. This could be adjusted to only add a cost when a sensor covers multiple targets simply by replacing $\alpha\lambda_{s_u j}$ with $\alpha\lambda_{s_u j} - 1$. The remaining addition is what must be modified in the pricing subproblem.

$$\sum_{j|P_j\in\Omega} \left(\left(\sum_{\ell\in\varepsilon} \ell y_{s_u \ell j} \right) + \alpha\lambda_{s_u j} \right) t_j \leq b_{s_u} \quad \forall s_u \in S \tag{3.2}$$

The original pricing subproblem constraints should suffice but the objective function here must be modified. The first solution that I proposed can be seen in (3.3). It can be seen that the idea is, for each sensor, to use a summation that for each target that is in the sensing range of that sensor, adds the value of flow between the relevant target and sensor. The result of this summation is then multiplied by α and added to the role allocation cost summation before the reduced cost criterion is carried out. The problem with this solution is that it does not actually consider the sensor that is carrying out the monitoring of the targets. In fact, this proposed solution can be simplified to what can be seen in (3.4). In this formula, all that is added to the original model is α multiplied by the number of targets in the instance. This is a flat rate that will have no impact on the decision of which cover is more interesting than another.

$$1 - \sum_{v\in S} \left(\left(\sum_{\ell\in\varepsilon} \pi_v \ell y_{s_u \ell j} \right) + \alpha \left(\sum_{u\in K|a(u,v)} x_{uv} \right) \right) \tag{3.3}$$

$$1 - \left(\sum_{v\in S} \left(\sum_{\ell\in\varepsilon} \pi_v \ell y_{s_u \ell j} \right) + \alpha|K| \right) \tag{3.4}$$

From this proposed solution that is unsuitable, it can be noted that the final proposal must

Model M3:	Maximum network lifetime problem with sensor-target connection establishment cost		
Maximize:	$\sum\limits_{j \mid P_j \in \Omega} t_j$		(1)
Subject to:	$\sum\limits_{j \mid P_j \in \Omega} \left(\left(\sum_{\ell \in \varepsilon} \ell y_{s_u \ell j} \right) + \alpha \lambda_{s_u j} \right) t_j \leq b_{s_u}$	$\forall s_u \in S$	(2)
	$t_j \geq 0$	$\forall j \mid P_j \in \Omega$	(3)

Table 3.1: Maximum network lifetime problem with target data retrieval cost

consider the specific sensor that is monitoring the given targets. The dual variable π_v from the maximum network lifetime problem that represents the battery constraint of the sensor can be used here to do this by simply multiplying the solution by this value. Thus, the final addition to the objective function is seen in (3.5) and since for this research, only instances where a sensor can carry out just one role at any one time, this can be simplified to the formula (3.6) however; we will stick with (3.5) in order to make future adaptations to the model easier. Now that the changes to both models have been made, the final models for this research are presented. The updated maximum network lifetime problem is presented in Table 3.1 and the pricing subproblem in Table 3.2. The variables discussed throughout this report are summarised in Table 3.3.

$$1 - \sum_{v \in S} \left(\left(\sum_{\ell \in \varepsilon} \pi_v \ell y_{s_u \ell j} \right) + \left(\pi_v \alpha \sum_{u \in K \mid \exists a(u,v)} x_{uv} \right) \right) \tag{3.5}$$

$$1 - \sum_{v \in S} \pi_v \left(\left(\sum_{\ell \in \varepsilon} \ell y_{s_u \ell j} \right) + \alpha \left(\sum_{u \in K \mid \exists a(u,v)} x_{uv} \right) \right) \tag{3.6}$$

3.3 Implementation

In order to test my proposed modelling of the problem, I must choose the technologies and methods that can be used effectively in completing this task. Firstly, a programming language must be selected. I opted to use Python to conduct this research as it is held in high regard for its flexibility in how it can be used in many domains and performs fairly well. This is supported by a study into the usage of Python by Abhinav Nagpal and Goldie Gabrani where it was found that Python is the language of choice for many of those working in numerical analysis and data science [28] and in [6] where the performance of Python was found to be sufficient for scientific domains whilst also boasting large amounts of supportive modules. In the area of optimisation specifically, it is mentioned in [19] that Python can be a very powerful tool for optimisation and that the language benefits from many popular commercial solvers providing a specific interface for using their optimisation software with Python.

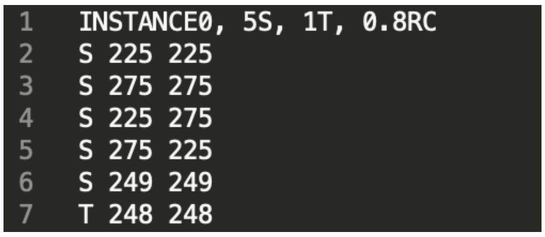
Next, an optimisation solver will be required so that the optimal lifetime can be found efficiently. The choice of this solver is also restricted to those that offer some kind of support of usage with Python, the programming language I have chosen to use. Thankfully, as mentioned previously, this is not much of a restriction as Python is a popular language to use in this domain

Model M4:	Pricing subproblem with sensor-target connection establishment cost		
Maximize:	$1 - \sum_{v \in S} \left(\left(\sum_{\ell \in \varepsilon} \pi_v \ell y_{s_u \ell j} \right) + \left(\pi_v \alpha \sum_{u \in K \mid \exists a(u,v)} x_{uv} \right) \right)$		(4)
Subject to:	$\sum_{u \in S \mid \exists a(v,u)} x_{vu} = 1$	$\forall v \in K$	(5)
	$\sum_{u \in N \mid \exists a(v,u)} x_{vu} - \sum_{u \in N \mid \exists a(u,v)} x_{uv} = 0$	$\forall v \in S$	(6)
	$\sum_{u \in N \mid \exists a(u,s_0)} x_{us_0} = \lvert K \rvert$		(7)
	$x_{uv} \leq \lvert K \rvert \left(y_{s_v E_r j} + y_{s_v E_s j} \right)$	$\forall u, v \in S \mid \exists a(u,v)$	(8)
	$x_{vu} \leq \lvert K \rvert \left(y_{s_v E_r j} + y_{s_v E_s j} \right)$	$\forall v, u \in S \mid \exists a(v,u)$	(9)
	$x_{uv} \leq y_{s_v E_s j}$	$\forall u \in K, \forall v \in S \mid \exists a(u,v)$	(10)
	$\sum_{\ell \in \varepsilon} y_{s_v \ell j} = 1$	$\forall v \in S$	(11)
	$x_{uv} \in \mathbb{Z}^+ \cup \{0\}$	$\forall u, v \in N$	(12)
	$y_{s_v \ell j} \in \{0,1\}$	$\forall v \in N, \ell \in \varepsilon$	(13)

Table 3.2: Pricing subproblem with target data retrieval cost

Notation	Meaning
m	Number of sensors
n	Number of targets
S	Set of sensors
K	Set of targets
s_0	Base station
R_s	Sensing range
R_c	Communication range
E_s	Energy consumption rate for source sensors
E_r	Energy consumption rate for relaying sensors
E_i	Energy consumption rate for inactive sensors
ε	The set of energy consumption rates
Ω	All feasible sensor network configurations that satisfy the constraints
j	An index used to loop through all partitions
S_s^j	The set of source sensors in partition j
S_r^j	The set of relay sensors in partition j
S_i^j	The set of inactive sensors in partition j
P_j	Partition of S into S_s^j, S_r^j and S_i^j
N	Set of nodes $S \cup K \cup \{s_0\}$
N_j	Subset of nodes $N \setminus S_i^j$
A	Set of arcs between nodes N
$G(N,A)$	Graph containing nodes N and arcs A
$G[N_j]$	Subgraph of G induced by N_j
t_j	The decision variables of the model denoting the time allocated to each partition
b_{s_u}	The battery capacity of each sensor s_u
$\vec{\pi}$	Dual variables with values π_v associated to each sensor's battery constraint
x_{uv}	Flow passing between nodes $u, v \in N$
$y_{s_v \ell j}$	Binary role allocation variable for a sensor in a partition
z	The maximum number of targets a sensor can retrieve data from in a cover, unused in final model
$\lambda_{s_u j}$	The number of targets that a sensor s_u is responsible for retrieving data from in a cover P_j
α	The cost rate defined for every sensor that is retrieving data directly from targets in a cover

Table 3.3: A table showing all of the variables involved in the problem.



```
INSTANCE0, 5S, 1T, 0.8RC
S 225 225
S 275 275
S 225 275
S 275 225
S 249 249
T 248 248
```

Figure 3.3: The format of an example instance text file.

[28, 19]. One study into optimisation solvers, specifically for linear and mixed integer programming problems, investigated the performance of three commercial solvers [18]. Namely, this study looked at how CPLEX, Gurobi and Xpress compared with respect to how long each took to find the optimal solution for each instance in the dataset. The experiment found that Gurobi performed best in most cases with CPLEX following closely behind. Furthermore, another study that compared both commercial and free-to-use solvers in their ability to solve linear problems found that Gurobi had the lowest average run time for the experiment's data set and successfully solved the most instances of all solvers in the given time [26]. Despite this, in this same paper, Gurobi performed worse than CPLEX on average for the second experiment that applied the solvers to the secondary cell suppression problem. I decided to use Gurobi for this project as it boasts impressive performance and encourages the use of Python with a specific environment designed for combining the modelling techniques of the solver with the programming constructs of the Python programming language [14]. Furthermore, this decision was solidified by the opportunity offered by Gurobi of obtaining a free license for their software if it is to be used for academic purposes only. Clearly, it is beneficial to this research that I use a commercial solver if possible as evidenced by [26].

The main drawback to using Gurobi for this research is that column generation is not an implemented feature of the solver. The column generation approach is possible only through iteratively redefining and calling each model and upon each model's termination, returning the required variables that will then be used for the next iteration. Specifically, in a single iteration, M3 should return the optimal dual variables that will then be passed as a parameter to M4. Then, from M4, the new covers must be returned using the binary role allocation variables to form new elements that will be added to the partitions array for use in M3 again. Alongside this, the flow variables x_{uv} must be used to generate the variables $\lambda_{s_u j}$ for every sensor in every new cover.

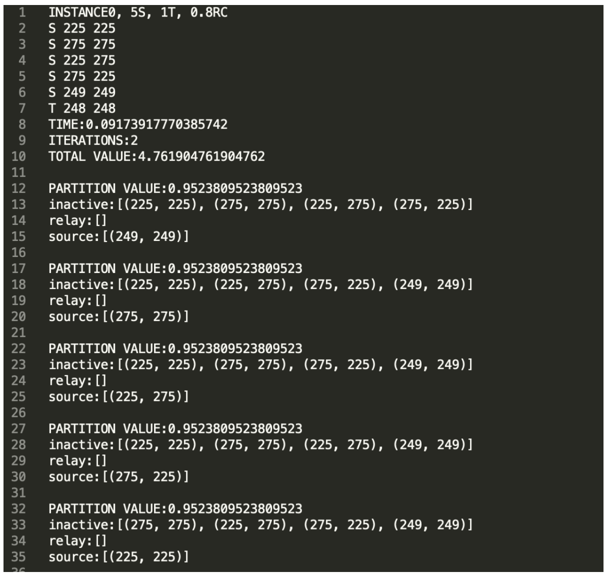
The first task that must be performed by the program is reading the instance information that the optimal solution is to be found for. I chose to read this in from a text file. The file format consists of the first line including key information about the instance such as the instance number, number of sensors, number of targets, and for this research specifically, the particular relay energy consumption rate (the source energy consumption rate is constant). The following lines consist of the coordinates of both the sensors and targets in the instance, prefixed by an S or T to distinguish the two node types in the file. More information on the file format and how these files are created can be found in section 4.2 and an example of how this file would look is seen in Figure 3.3.

For the models to work, the set A of arcs must be known. This can be calculated by looping through the set of sensor coordinates, and then through the set N containing the coordinates of all sensors, targets and the base station. The distance between the coordinates can be calculated using the Euclidean distance formula as shown in (3.7) where $(x_1,...,x_n)$ and $(y_1,...,y_n)$ are the x and y coordinates respectively and n is the number of dimensions in each [32]. Once the distance, is calculated, a conditional block is used to check whether the node is another sensor, a target or the base station to determine which of the two ranges are relevant to compare the distance with. If the distance is within the respective range, then an arc between the sensor and node (the order of the arc depends on which type the node happens to be) can be added to the arcs list.

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2} \tag{3.7}$$

It is also important that for the sake of the experiment, some kind of instance feasibility check is carried out to ensure that there is at least one feasible solution and there are no connectivity restraints that are unsatisfied. Alongside this, a starting set of covers is needed to begin the process. The former could be carried out by checking there is a path from every target to the base station, however; I chose another approach that will allow us to hit two birds with one stone. Both of these requirements can be fulfilled by calling the pricing subproblem with π_v values of 0 for every sensor. This will result in every possible cover being interesting and so some random covers can be taken to start with. The number of covers that can be returned from each iteration of the pricing subproblem is capped at 1000 covers as it was seen from initial testing that when set any higher, the performance of the algorithm when retrieving new covers was severely impacted and when set any lower, the objective value failed to improve much until after several iterations. The instance feasibility check is implemented by checking if some covers were returned by this first instance of the subproblem. If no feasible covers were found, then there is no solution for the instance. For a feasible instance, the covers returned here are added to the main set of covers for the first call to the master problem and from there, the iterative process begins.

After no more interesting covers can be found, and the optimal solution is known, the solution file must be created. This is important not only so that the covers and their allocated times are written permanently in storage but also, for this project, so that the instance can be used for visualising the results using my visualisation tool. The first thing that is carried out when writing the solution to a file is the duplication of the instance file. It is crucial that the key variables are written along with the coordinates of all sensors and targets. Following this, the time taken to find an optimal solution is written before the number of iterations and the final objective value are noted in the file. Finally, a for loop is used iterate over the covers, writing the sensor roles and allocated time for each element of the partitions array. The format of this solution file can be seen in Figure 3.4.



```
INSTANCE0, 5S, 1T, 0.8RC
S 225 225
S 275 275
S 225 275
S 275 225
S 249 249
T 248 248
TIME:0.09173917770385742
ITERATIONS:2
TOTAL VALUE:4.761904761904762

PARTITION VALUE:0.9523809523809523
inactive:[(225, 225), (275, 275), (225, 275), (275, 225)]
relay:[]
source:[(249, 249)]

PARTITION VALUE:0.9523809523809523
inactive:[(225, 225), (225, 275), (275, 225), (249, 249)]
relay:[]
source:[(275, 275)]

PARTITION VALUE:0.9523809523809523
inactive:[(225, 225), (275, 275), (275, 225), (249, 249)]
relay:[]
source:[(225, 275)]

PARTITION VALUE:0.9523809523809523
inactive:[(225, 225), (275, 275), (225, 275), (249, 249)]
relay:[]
source:[(275, 225)]

PARTITION VALUE:0.9523809523809523
inactive:[(275, 275), (225, 275), (275, 225), (249, 249)]
relay:[]
source:[(225, 225)]
```

Figure 3.4: The format of an example instance solution file.

Chapter 4

Experiments

4.1 The experimental setting

When creating the experimental design for this project, it was important that I kept the key parameters at a similar value to the experiments carried out in [7]. This approach means that I can compare the results with that of their research to record the impact of implementation differences on similar (albeit nonidentical) instances. Whilst the Gurobi solver is known to be very quick, it should be noted that it does not feature built-in functionalities for column generation. Therefore, it is possible that my manual implementation of the algorithms could result in slower speeds. This possibility did affect some of the design decisions for the experiment.

One such experimental design decision is the number of sensors I decided to test with. In [7], their main goal is to compare three methods of solving the pricing subproblem in performance. They do this by setting a time limit of one hour for each run of the optimiser and comparing the algorithms by how many instances they managed to find the optimal solution for. This time limit will still be used in this research because of the time constraints of the project, however; in order to make clear conclusions on the results that are obtained, I chose to use smaller amounts of sensors that will result in a large reduction in how many decisions the optimiser must make. Consequently, the time it takes to find the optimal solution should be far lower and so, usable data for comparisons of the additional costs will be found. In the case that the time limit is exceeded for a specific instance, it is still possible to obtain interesting results as the impact of additional costs on the performance of the program can be measured.

As is the case in [7], I experimented on different amounts of sensors and targets, tested two different sensing ranges and two relay consumption rates (in order to include both instances of CMLP-MR and CMLP in our experiment). To build on this design and to answer my research question, I will be optimising every instance with three different inactive consumption rates. I chose to test with E_i values of $0.0, 0.1$, and 0.3 to ensure that I can make clear conclusions on the effect it has on the optimal solution and performance. Furthermore, each instance will be tested for my second research question. This involves using different values of α, the cost for a sensor establishing a connection with every target it is allocated for sensing.

I considered an area of $500x500$ in which the sensors will be deployed. The number of sensors was chosen as $m = |S| = \{50, 100, 150, 200\}$ and the number of targets $n = |K| = \{15, 30\}$. These parameters were chosen with the time limit in mind but were also chosen to obtain a wide range of objective values. In instances where the number of sensors is the lowest chosen value and the targets is the highest, we should obtain final solutions that are fairly low. The opposite will be seen for a higher number of sensors and lower number of targets. This means that the impact of additional costs can be analysed for how the difference in the solution changes when the parameters are altered. The base station was chosen to be fixed at the coordinates of $(250,250)$. All experiments were carried out on a machine with Windows 10, an Intel i7-7700 CPU 4 core 3.6GHz processor and 16GB of RAM.

4.2 The generation of instances

I chose to experiment on a total of 80 instances that are generated to test various combinations of variable values. The first 40 instances consider CMLP-MR problems where the relaying consumption rate is set to 0.8 whilst the other 40 cover CMLP problems where $E_r = E_s$ or in other words, where there is no benefit to giving a sensor the relaying role over the source role. Within each of these 40 instances, 10 instances are allocated to the 4 differing values of m, the number of sensors and then within each of those 10, 5 instances are allocated to the 2 values of n, the number of targets. This design will result in a thorough experimentation of the research questions by testing every combination of the parameter values.

These instances would of course take a long time to write into a file manually so some script must be created that can do this quickly and in a way that can be read later for optimising. This can be done by calling a function to generate an instance for a defined number of instances, in this case, 80. For every instance, a variable is defined to contain the sensor coordinates, another for the target coordinates and a third variable is defined to hold both the sensor coordinates and target coordinates as well as the specified coordinates of the base station in the instance. The purpose of the latter coordinates variable is to be used for checking a newly generated coordinate for a sensor or target is not already taken by another node, thus preventing duplication. For this research, sensor and target coordinates are chosen completely at random.

Another important part of generating the instances is choosing the correct parameters for the instance using only the instance number. This is done by the creation of dictionaries for each instance variable specifying the possible values that can be chosen as the dictionary keys and the lower and upper bounds, that represent the range of instance numbers that the corresponding key should be chosen for, as the dictionary values. For example, the target dictionary implemented in Python can be seen as $\{15:[0,4],30:[5,9]\}$. To understand the values contained in this dictionary, one more value must be defined. Let p be the number of instances that are generated before a particular parameter is reset to its first possible value. In the target example, p_n can be defined as 10 as the 10th instance will contain 30 targets but the 11th instance should again contain 15. For the number of sensors by this same logic, p_m can be defined as 40. Therefore, the dictionary values are defined with respect to p, For every parameter, the instance number modulo p is calculated and found in the parameter dictionary in order to choose the correct value.

Finally, the information generated for each instance must be written to a text file for future use. This is done in a similar way to the described method in Chapter 3. The first line of the instance file contains the key information for that instance such as the value of the parameters. The following lines describe the coordinates of the sensors and targets in the instance, prefixed by an S or T to make that distinction.

4.3 Results

4.3.1 Inactive costs with the original model

Firstly, lets tackle the first research question that aims to explore different energy consumption rates of inactive sensors. The full results of the original model experiments before target data retrieval was taken into account can be seen in Tables 4.1 and 4.2. If we first break the comparisons up into the number of sensors in the problem, as this is the most influential parameter in the performance of the program as well as the optimal solution it finds. In CMLP-MR instances where $m = 50$ (instances 1-10), the average time taken for $E_i = 0.0$ was 15.36 seconds, the average number of iterations was 7.3 and the average optimal solution was 2.00. For $E_i = 0.1$, the average time taken was 15.87 seconds, 6.9 iterations were performed on average and 1.81 was the average optimal solution. Finally, the average time, number of iterations, and optimal solution for $E_i = 0.3$ was 13.93, 6.6 and 1.54 respectively. For CMLP instances, when $E_i = 0.0$, the average time taken was 11.38, the average iterations was 5.4, and the average final solution was 1.80. When $E_i = 0.1$ for CMLP instances, the average time taken was 11.72, the average iterations was 5.7, and 1.64 was found as the average final solution. For the final inactive rate $E_i = 0.3$, the average time taken was 11.36, 6.1 was found as the average number of iterations and 1.41 is the average final solution.

For CMLP-MR instances where $m = 100$ (instances 11-20), when $E_i = 0.0$, the average time taken was 787.72 seconds, 40.8 iterations were needed on average and the average solution was 5.40. When $E_i = 0.1$, 739.31 seconds was taken to terminate on average, the average iterations was 39.1, and average solution on termination was 3.67. On average, the time taken, number of iterations and final solution was 641.05 seconds, 36.2, and 2.29 respectively for an inactive consumption rate E_i of 0.3. Then, for CMLP instances, the following results were obtained. When $E_i = 0.0$, the average time taken was 422.88 seconds, the average iterations was 38.7 and 5.10 was the average solution found on termination. When $E_i = 0.1$ for CMLP, 455.96, 37.3, and 3.54 were found to be the average values for time taken, iterations, and final solution respectively. For $E_i = 0.3$, these instances result in average values of 471.93 seconds for time taken, 39.5 for number of iterations, and 2.24 for the solution found.

Next, when $m = 150$, $E_r = 0.8$, and $E_i = 0.0$, the averages for time taken, iterations and final solution are 2870.11 seconds, 76.5, and 9.57 respectively. When $E_i = 0.1$ for these instances, the time taken on average is 2907.30, the average number of iterations is 74.1 and the average final solution on termination is 5.14. Lastly, for $E_i = 0.3$, the average for time taken was 2789.28 seconds, the average number of iterations was 74.9, the average final solution was 2.68. For CMLP instances, the time taken in seconds was on average 3298.28, 3166.08 and 3226.68 for inactive consumption rates E_i of 0.0, 0.1, and 0.3 respectively. The average number of iterations carried out for these instances was 87.7, 82.4, and 86.8, again for E_i values of 0.0, 0.1, and 0.3 respectively. Finally, for these instances, the average solution found for E_i rates of 0.0, 0.1, and 0.3 was 9.09, 5.0, and 2.64 respectively.

For the largest number of sensors $m = 200$ in CMLP-MR instances, all instances took the entire hour and did not find the guaranteed optimal solution. Despite this, some key results can be taken from instances where the time limit was reached. The solution that can be obtained within an hour of optimising is still important as to how the program performs and what this means for its applicability to real world scenarios where time may be constrained and an optimal solution is needed urgently. For these instances, when $E_i = 0.0$, the average number of instances was 60.7 and the average final solution was 7.13. For $E_i = 0.1$, the average number of iterations was 60.0 and the average final solution was 4.50. For $E_i = 0.3$, the average number of iterations was 61.0 and the average final solution was 2.49. Similarly for CMLP instances, the program hit the one hour time limit in every case. When $E_i = 0.0$, the average number of iterations was 61.1 and the average solution found was 5.43. For $E_i = 0.1$, the number of iterations carried out in the hour on average was 62.3 and the average solution found in the hour was 3.84. Finally, for $E_i = 0.3$, the number of iterations carried out on average was 61.6 whilst the average solution found was 2.36.

A summary of these results in table form can be seen in Table 4.3.

4.3.2 Inactive costs with target data retrieval costs

Here, the main results will be presented from the experiments on the newly proposed model for taking into account a cost for retrieving data from targets on a sensor's battery. The full results can be seen in Tables 4.4 and 4.5. As was the case for presenting the original model experiment results, the averages will be broken into number of sensors, relay consumption rate (CMLP-MR vs CMLP), and finally the inactive consumption rate. In section 4.4, not only will the first research question on inactive sensors be discussed, but also comparisons of the two models and the impact of the addition of the new cost will be analysed.

We start with instance of 50 sensors for CMLP-MR. When $E_i = 0.0$, an average execution time of 887.93 seconds was found, whilst the number of iterations made on average was 50.9 and the average solution was 1.43. For $E_i = 0.1$, the average time taken was 1154.51 seconds, the average iterations was 64.8, and the average solution found was 1.31. For $E_i = 0.3$, the average time taken was 1208.26 seconds, the average iterations was 67.1, and the average solution found was 1.11. Next, the CMLP instances with 50 sensors. For E_i values of 0.0, 0.1, and 0.3, the average times taken were 567.11, 604.95, and 548.53 seconds respectively. The number of iterations carried out on average were 37.3, 38.1, and 37.2 respectively. The solution found on average was 1.28, 1.20, and 1.06 respectively.

Now, we look at instances with 100 sensors, starting again with CMLP-MR instances. For these instances, for E_i values of 0.0, 0.1, and 0.3, the average times taken in seconds were 2090.52, 1993.95, and 2034.47 respectively, the average number of iterations were 54.9, 48.8, and 52.0 respectively, and the average solution value found on termination was 3.24, 2.51, and 1.77 respectively. When it comes to CMLP instances, the results show for E_i values of 0.0, 0.1, and 0.3, average times of 1822.82, 1844.61, and 1865.64 seconds respectively, an average number of iterations of 44.7, 44.0, and 43.1 respectively, and average final solution values of 3.27, 2.54, and 1.78 respectively.

For instances with 150 sensors, the following key results were obtained. For CMLP-MR instances, the average time taken for $E_i = 0.0$ was 3374.34 seconds, whilst the average number of iterations was 52.6, and the average solution value was 4.83. For E_i values of 0.1 and 0.3, the average times taken in seconds were 3360.67 and 3265.95 respectively, the average number of iterations were 50.7 and 49.3 respectively, and the average solution on termination was 3.40 for $E_i = 0.1$ and 2.12 for $E_i = 0.3$. For CMLP, the average termination times for inactive energy consumption rates of 0.0, 0,1, and 0.3 were 3504.64, 3484.91, and 3448.82 seconds respectively, the average number of iterations were 52.3, 51.5, and 48.2 respectively, and the average solution

			$E_i = 0.0$			$E_i = 0.1$			$E_i = 0.3$		
Instance	m	n	Time	Iters	Sol	Time	Iters	Sol	Time	Iters	Sol
1	50	15	6.87	5	1.20	4.95	3	1.20	6.09	4	1.18
2	50	15	47.12	18	2.77	53.67	19	2.39	50.17	20	1.88
3	50	15	32.59	18	2.50	22.03	10	2.22	17.96	10	1.82
4	50	15	16.13	8	2.50	17.28	8	2.22	14.52	8	1.82
5	50	15	4.74	3	2.00	6.50	4	1.82	4.71	3	1.54
6	50	30	7.39	4	1.25	7.53	4	1.25	4.17	2	1.25
7	50	30	12.93	6	2.50	17.06	8	2.22	11.97	6	1.82
8	50	30	4.30	2	1.25	7.97	4	1.25	6.01	3	1.25
9	50	30	1.15	0	1.00	1.18	0	1.00	1.17	0	1.00
10	50	30	20.39	9	3.00	20.50	9	2.50	22.53	10	1.88
11	100	15	1373.28	71	8.00	997.66	61	4.71	673.41	52	2.58
12	100	15	337.63	35	5.40	366.67	37	3.77	229.00	28	2.35
13	100	15	48.75	8	2.40	55.78	9	2.14	96.10	14	1.75
14	100	15	3600.00	96	7.41	3600.00	90	4.52	3600.00	92	2.57
15	100	15	518.57	44	7.20	772.89	55	4.45	403.56	40	2.52
16	100	30	147.28	18	4.00	128.85	16	3.08	146.07	18	2.11
17	100	30	1215.66	67	6.00	754.68	48	4.00	614.28	46	2.40
18	100	30	171.13	20	4.20	222.53	24	3.19	184.04	21	2.16
19	100	30	172.99	20	4.00	196.55	22	3.08	157.56	19	2.11
20	100	30	291.88	29	5.40	297.52	29	3.77	306.50	32	2.35
21	150	15	751.35	35	7.20	1004.54	43	4.45	988.59	44	2.52
22	150	15	1520.65	58	9.20	1584.13	58	5.06	1389.61	54	2.66
23	150	15	3600.00	90	11.15	3600.00	86	5.46	3600.00	90	2.77
24	150	15	3600.00	93	10.31	3600.00	88	5.34	3600.00	92	2.73
25	150	15	3600.00	92	10.12	3600.00	85	5.30	3600.00	88	2.73
26	150	30	3556.72	88	11.00	3600.00	83	5.48	3600.00	92	2.75
27	150	30	3488.72	88	9.20	3083.94	76	5.09	2195.50	61	2.68
28	150	30	1383.68	48	8.00	1800.36	56	4.71	1719.11	57	2.58
29	150	30	3600.00	93	9.31	3600.00	85	5.14	3600.00	89	2.68
30	150	30	3600.00	80	10.24	3600.00	81	5.33	3600.00	82	2.74
31	200	15	3600.00	62	8.38	3600.00	62	4.73	3600.00	61	2.67
32	200	15	3600.00	63	7.10	3600.00	62	4.65	3600.00	64	2.55
33	200	15	3600.00	62	7.95	3600.00	60	4.72	3600.00	61	2.63
34	200	15	3600.00	65	8.85	3600.00	64	5.06	3600.00	67	2.56
35	200	15	3600.00	64	7.44	3600.00	62	4.91	3600.00	64	2.49
36	200	30	3600.00	58	6.33	3600.00	60	4.40	3600.00	61	2.48
37	200	30	3600.00	58	6.16	3600.00	58	4.27	3600.00	58	2.45
38	200	30	3600.00	58	7.00	3600.00	57	4.56	3600.00	58	2.38
39	200	30	3600.00	61	5.67	3600.00	59	3.81	3600.00	60	2.28
40	200	30	3600.00	56	6.38	3600.00	56	3.88	3600.00	56	2.45

Table 4.1: Experiment results for CMLP-MR with the original models without data retrieval costs, as shown in Tables 2.1 and 2.2 ($E_r = 0.8$).

			$E_i = 0.0$			$E_i = 0.1$			$E_i = 0.3$		
Instance	m	n	Time	Iters	Sol	Time	Iters	Sol	Time	Iters	Sol
41	50	15	0.96	0	1.00	1.01	0	1.00	0.98	0	1.00
42	50	15	31.77	14	3.00	27.35	13	2.50	27.05	14	1.87
43	50	15	3.25	2	1.00	3.57	2	1.00	3.13	2	1.00
44	50	15	8.60	5	2.00	6.80	4	1.82	12.64	8	1.54
45	50	15	6.48	4	2.00	8.53	5	1.82	5.97	4	1.54
46	50	30	3.84	2	1.00	3.80	2	1.00	3.93	2	1.00
47	50	30	26.14	11	2.00	19.81	9	1.82	17.34	9	1.54
48	50	30	14.22	7	2.00	23.58	11	1.82	14.65	8	1.54
49	50	30	7.60	4	2.00	11.76	6	1.82	16.96	9	1.54
50	50	30	10.89	5	2.00	10.97	5	1.82	10.91	5	1.54
51	100	15	340.11	35	6.00	308.41	33	4.00	225.15	28	2.40
52	100	15	900.50	64	7.00	819.70	56	4.38	882.53	63	2.50
53	100	15	725.93	51	7.00	759.65	54	4.38	1000.51	63	2.50
54	100	15	184.68	24	4.00	177.72	23	3.08	137.49	20	2.11
55	100	15	633,85	51	7.00	657.61	51	4.38	847.38	67	2.50
56	100	30	805.41	55	6.00	931.77	57	4.00	758.75	56	2.40
57	100	30	265.69	28	4.00	212.88	24	3.08	271.21	29	2.11
58	100	30	264.76	30	3.00	161.36	20	2.50	204.69	24	1.88
59	100	30	153.96	19	3.00	216.64	24	2.50	183.01	22	1.88
60	100	30	292.68	30	4.00	313.82	31	3.08	208.59	23	2.11
61	150	15	3600.00	95	10.31	3600.00	92	5.27	3600.00	95	2.71
62	150	15	3600.00	93	9.32	3600.00	89	5.07	3600.00	87	2.68
63	150	15	3600.00	94	9.72	3600.00	92	5.20	3600.00	98	2.69
64	150	15	3600.00	100	10.44	3600.00	93	5.44	3600.00	96	2.73
65	150	15	3600.00	89	10.58	3600.00	91	5.38	3600.00	102	2.72
66	150	30	2551.89	77	8.00	1926.71	59	4.71	2383.95	73	2.58
67	150	30	3600.00	89	9.45	3600.00	84	5.19	3600.00	92	2.68
68	150	30	1630.95	56	5.00	934.10	38	3.57	1082.82	40	2.27
69	150	30	3600.00	95	8.74	3600.00	96	4.99	3600.00	94	2.63
70	150	30	3600.00	89	9.38	3600.00	90	5.17	3600.00	91	2.66
71	200	15	3600.00	64	6.00	3600.00	65	4.41	3600.00	65	2.55
72	200	15	3600.00	64	6.71	3600.00	63	4.64	3600.00	62	2.49
73	200	15	3600.00	63	4.81	3600.00	63	3.84	3600.00	64	2.31
74	200	15	3600.00	61	6.94	3600.00	64	3.94	3600.00	63	2.47
75	200	15	3600.00	64	6.14	3600.00	64	3.81	3600.00	64	2.43
76	200	30	3600.00	59	4.83	3600.00	60	3.17	3600.00	61	2.15
77	200	30	3600.00	59	4.98	3600.00	60	3.91	3600.00	60	2.29
78	200	30	3600.00	56	4.34	3600.00	58	3.31	3600.00	57	2.18
79	200	30	3600.00	61	4.62	3600.00	66	3.60	3600.00	60	2.36
80	200	30	3600.00	60	4.97	3600.00	60	3.80	3600.00	60	2.35

Table 4.2: Experiment results for CMLP with the original models without data retrieval costs, as shown in Tables 2.1 and 2.2 ($E_r = 1.0$).

		$E_i = 0.0$			$E_i = 0.1$			$E_i = 0.3$		
m	E_r	Time	Iters	Sol	Time	Iters	Sol	Time	Iters	Sol
50	0.8	15.36	7.3	2.00	15.87	6.9	1.81	13.93	6.6	1.54
50	1.0	11.38	5.4	1.80	11.72	5.7	1.64	11.36	6.1	1.41
100	0.8	787.72	40.8	5.40	739.31	39.1	3.67	641.05	36.2	2.29
100	1.0	422.88	38.7	5.10	455.96	37.3	3.54	471.93	39.5	2.24
150	0.8	2870.11	76.5	9.57	2907.30	74.1	5.14	2789.28	74.9	2.68
150	1.0	3298.28	87.7	9.09	3166.08	82.4	5.00	3226.68	86.8	2.64
200	0.8	3600.00	60.7	7.13	3600.00	60.0	4.50	3600.00	61.0	2.49
200	1.0	3600.00	61.1	5.43	3600.00	62.3	3.84	3600.00	61.6	2.36

Table 4.3: A summary of the mean results for the original models.

found for each of them was 4.85, 3.36, and 2.10 respectively.

For instances with 200 sensors, as was the case with the original model, all instances failed to reach the optimal solution before the hour time limit was reached but again, this does not mean that observations cannot be made on the difference that the cost has made, on what type of solution we can expect the program to reach within the hour for all cases. For CMLP-MR instances, the average number of iterations for E_i values of 0.0, 0.1, and 0.3 were 37.8, 37.3, and 37.5 respectively whilst the average solution found for each of these values were 5.04, 3.51, and 2.22 respectively. For CMLP instances and for the same three values of E_i, the average number of iterations were 38.1, 38.0, and 37.7 respectively. The average solution upon termination was 4.27 for $E_i = 0.0$, 3.30 for $E_i = 0.1$, and 2.13 for $E_i = 0.3$.

A summary of these results can be seen in Table 4.6.

4.4 Discussion

When we look at the results, the observations that we can make about changing the energy consumption rate for inactive sensors are clear. Changing the rate in both the old and new model resulted in nearly no difference in the average time it took for the program to find an optimal solution. Similarly, the number of iterations that were carried out on an instance in order to terminate at the optimal solution (or best solution in an hour if time expired) did not change a statistically significant amount. Despite this, the final solution changed drastically with a change in the inactive sensor energy consumption rate. In fact, on average for instances with larger amounts of sensors, the final solution was reduced by over 50% when E_i was changed from 0.0 to 0.3. From this, we can conclude that in applications where sensor networks are required to operate in extreme environments when inactive sensors may still consume energy, we can rely on the models to perform just as well, if not better. However, the impact an inactive sensor cost has on the final solution is clear, and this should absolutely be considered when designing a network for a real scenario, it may be worthwhile to invest in hardware and technology that will either reduce the amount of energy a sensor consumes when inactive, such as incorporating some kind of temperature regulator, or in adding more sensors to cover crucial areas where sensor batteries are depleted quickly.

The next research question involved exploring the effect of a cost for retrieving/sensing data from a target on the source sensor's battery. It should be mentioned that the new model proposed in this paper did have a significant impact on the performance of the program. It is noteworthy that the total time taken sometimes increased by over 100 times the original model's time taken. The

			$E_i = 0.0$			$E_i = 0.1$			$E_i = 0.3$		
Instance	m	n	Time	Iters	Sol	Time	Iters	Sol	Time	Iters	Sol
1	50	15	11.03	2	0.80	11.33	2	0.79	11.54	2	0.77
2	50	15	760.45	83	2.01	733.89	77	1.77	655.61	73	1.43
3	50	15	527.98	60	2.27	2628.50	168	1.99	2611.77	167	1.54
4	50	15	50.00	10	1.73	78.04	15	1.54	81.26	16	1.26
5	50	15	18.07	4	1.00	18.87	4	0.96	26.71	6	0.87
6	50	30	151.89	17	1.13	159.85	18	1.08	81.88	10	0.94
7	50	30	3600.00	168	1.72	3600.00	166	1.53	3600.00	165	1.25
8	50	30	134.15	15	1.25	688.54	56	1.25	1382.21	95	1.10
9	50	30	25.69	3	0.67	26.10	3	0.67	31.66	4	0.67
10	50	30	3600.00	147	1.76	3600.00	139	1.56	3600.00	133	1.27
11	100	15	1601.64	61	4.47	1238.97	49	3.20	1089.26	48	2.05
12	100	15	1141.86	51	3.96	1065.47	44	3.00	1960.62	70	2.02
13	100	15	147.84	10	1.96	165.18	11	1.76	105.50	7	1.47
14	100	15	3600.00	72	4.44	3600.00	70	3.24	3600.00	81	2.10
15	100	15	3600.00	79	4.58	3600.00	62	3.26	3600.00	71	2.11
16	100	30	317.87	13	2.00	352.43	14	1.74	343.69	14	1.38
17	100	30	2892.39	77	2.67	2584.43	68	2.19	2381.81	64	1.61
18	100	30	1135.80	41	2.60	950.44	34	2.17	1005.19	35	1.63
19	100	30	2867.81	80	2.44	2782.53	76	2.05	2658.66	71	1.55
20	100	30	3600.00	65	3.24	3600.00	60	2.53	3600.00	59	1.78
21	150	15	1343.38	33	4.30	1440.52	34	3.14	1141.80	29	2.04
22	150	15	3600.00	71	5.61	3600.00	68	3.74	3600.00	66	2.25
23	150	15	3600.00	54	6.32	3600.00	54	4.01	3600.00	59	2.35
24	150	15	3600.00	65	5.84	3600.00	63	3.87	3600.00	63	2.30
25	150	15	3600.00	64	5.82	3366.14	62	3.81	2717.73	54	2.25
26	150	30	3600.00	53	4.30	3600.00	53	3.09	3600.00	51	2.00
27	150	30	3600.00	48	3.39	3600.00	44	3.12	3600.00	46	2.01
28	150	30	3600.00	48	3.95	3600.00	43	2.91	3600.00	43	1.94
29	150	30	3600.00	47	4.39	3600.00	46	3.18	3600.00	40	2.04
30	150	30	3600.00	43	4.37	3600.00	40	3.15	3600.00	42	2.03
31	200	15	3600.00	41	6.39	3600.00	40	4.14	3600.00	41	2.34
32	200	15	3600.00	43	6.02	3600.00	42	3.83	3600.00	43	2.42
33	200	15	3600.00	42	6.01	3600.00	42	3.99	3600.00	42	2.39
34	200	15	3600.00	44	6.16	3600.00	43	4.12	3600.00	42	2.36
35	200	15	3600.00	44	5.77	3600.00	42	3.96	3600.00	42	2.38
36	200	30	3600.00	33	3.88	3600.00	32	3.02	3600.00	33	2.07
37	200	30	3600.00	33	4.02	3600.00	33	2.93	3600.00	33	2.03
38	200	30	3600.00	33	4.34	3600.00	33	3.04	3600.00	33	2.12
39	200	30	3600.00	33	3.99	3600.00	34	3.02	3600.00	34	2.02
40	200	30	3600.00	32	3.80	3600.00	32	3.00	3600.00	32	2.05

Table 4.4: Experiment results for CMLP-MR with the new proposed models defined in Tables 3.1 and 3.2 ($E_r = 0.8, \alpha = 0.5$).

			$E_i = 0.0$			$E_i = 0.1$			$E_i = 0.3$		
Instance	m	n	Time	Iters	Sol	Time	Iters	Sol	Time	Iters	Sol
41	50	15	3.34	0	0.50	3.44	0	0.50	3.46	0	0.50
42	50	15	83.71	16	1.83	85.54	16	1.64	95.60	17	1.35
43	50	15	16.29	3	1.00	19.11	4	1.00	24.53	5	0.97
44	50	15	75.33	15	1.25	29.00	6	1.17	34.98	7	1.05
45	50	15	33.37	7	1.17	23.20	5	1.10	23.32	5	1.00
46	50	30	36.03	4	1.00	49.00	6	1.00	91.18	10	1.00
47	50	30	3600.00	174	1.64	3600.00	170	1.48	2519.00	134	1.23
48	50	30	279.75	29	1.33	402.65	38	1.25	433.63	40	1.07
49	50	30	429.90	42	1.42	463.83	42	1.31	573.08	50	1.13
50	50	30	1113.33	83	1.67	1373.74	94	1.50	1686.47	104	1.25
51	100	15	824.91	40	4.00	931.13	44	3.00	1102.54	44	2.00
52	100	15	3600.00	64	4.73	3600.00	61	3.35	3600.00	56	2.12
53	100	15	3600.00	68	4.33	3600.00	64	3.17	3600.00	60	2.05
54	100	15	319.07	20	2.67	264.91	17	2.22	323.11	20	1.67
55	100	15	3600.00	90	4.53	3600.00	89	3.25	3600.00	82	2.07
56	100	30	3600.00	65	3.24	3600.00	63	2.53	3600.00	66	1.78
57	100	30	491.30	20	2.33	558.57	21	1.99	555.06	22	1.53
58	100	30	969.11	35	1.97	1062.94	36	1.73	929.86	33	1.39
59	100	30	387.37	16	2.22	473.84	19	1.93	530.16	21	1.54
60	100	30	836.45	29	2.67	754.71	26	2.22	815.68	27	1.67
61	150	15	3600.00	68	5.09	3600.00	67	3.49	3600.00	67	2.14
62	150	15	3600.00	59	5.74	3600.00	56	3.85	3600.00	53	2.28
63	150	15	3600.00	56	5.84	3600.00	58	3.88	3600.00	55	2.28
64	150	15	3600.00	60	5.99	3600.00	62	3.91	3600.00	55	2.29
65	150	15	3600.00	54	5.96	3600.00	56	3.89	3600.00	51	2.31
66	150	30	3600.00	47	4.35	3600.00	41	3.07	3600.00	42	2.01
67	150	30	3600.00	44	4.30	3600.00	45	3.11	3600.00	42	2.01
68	150	30	2646.39	43	2.65	2449.12	40	2.19	2088.23	34	1.62
69	150	30	3600.00	44	4.21	3600.00	46	3.10	3600.00	42	2.01
70	150	30	3600.00	48	4.37	3600.00	44	3.12	3600.00	41	2.02
71	200	15	3600.00	43	4.67	3600.00	43	3.85	3600.00	42	2.25
72	200	15	3600.00	43	4.80	3600.00	42	3.69	3600.00	41	2.26
73	200	15	3600.00	42	5.20	3600.00	42	3.58	3600.00	43	2.22
74	200	15	3600.00	42	5.36	3600.00	43	3.78	3600.00	43	2.33
75	200	15	3600.00	43	4.61	3600.00	43	3.60	3600.00	43	2.22
76	200	30	3600.00	34	3.59	3600.00	33	2.86	3600.00	33	2.03
77	200	30	3600.00	33	3.69	3600.00	34	2.94	3600.00	33	2.04
78	200	30	3600.00	33	3.58	3600.00	32	2.99	3600.00	32	1.98
79	200	30	3600.00	34	3.61	3600.00	34	2.83	3600.00	34	2.07
80	200	30	3600.00	34	3.59	3600.00	34	2.83	3600.00	33	1.87

Table 4.5: Experiment results for CMLP with the new proposed models defined in Tables 3.1 and 3.2 ($E_r = 1.0, \alpha = 0.5$).

		$E_i = 0.0$			$E_i = 0.1$			$E_i = 0.3$		
m	E_r	Time	Iters	Sol	Time	Iters	Sol	Time	Iters	Sol
50	0.8	887.93	50.9	1.43	1154.51	64.8	1.31	1208.26	67.1	1.11
50	1.0	567.11	37.3	1.28	604.95	38.1	1.20	548.53	37.2	1.06
100	0.8	2090.52	54.9	3.24	1993.95	48.8	2.51	2034.47	52.0	1.77
100	1.0	1822.82	44.7	3.27	1844.61	44.0	2.54	1865.64	43.1	1.78
150	0.8	3374.34	52.6	4.83	3360.67	50.7	3.40	3265.95	49.3	2.12
150	1.0	3504.64	52.3	4.85	3484.91	51.5	3.36	3448.82	48.2	2.10
200	0.8	3600.00	37.8	5.04	3600.00	37.3	3.51	3600.00	37.5	2.22
200	1.0	3600.00	38.1	4.27	3600.00	38.0	3.30	3600.00	37.7	2.13

Table 4.6: A summary of the mean results for the new models.

performance for the new model experiments varied greatly, as some instances seemed to terminate in a similar amount of time to its original model counterpart, but on the other hand, this was not always the case. It is difficult to pinpoint what was causing this inconsistency but it is clear that overall, the performance of the program did suffer with the inclusion of this new cost. This is likely due to the objective function in the pricing subproblem, where a new summation is included. This summation specifically involves the flow decision variables, which of course will add a lot of time onto the pricing subproblem as the decision variables summation will clearly have a substantial impact on the objective value for each proposed cover.

Aside from the performance difference of the models, the solutions found also differed greatly. Similar to the impact of the inactive sensor cost, the target data retrieval cost was seen to impact the optimal solution often by around half. This is a substantial decrease. However, there were some instances that this new cost had little to no impact on the final solution. This could be explained in a few ways. First, it could be that no sensors were responsible for retrieving data from more than one target. The new model would still add a cost for this, but it would be far less substantial than if a sensor was sensing multiple targets in a single cover. This could result in such a small change in the solution that it could be disregarded in rounding. Second, it could be that for the changes for instances with smaller amounts of sensors, the solution was so low, that the difference could potentially be unnoticeable. The pricing subproblem is still able to produce covers as effective in the new model as in the old model and so eventually, it is not unreasonable to believe that the solutions could be so close that by rounding to two decimal places, the difference is minuscule.

Chapter 5

Visualisation Tool

5.1 Background

Wireless sensor networks have many applications and often can have significant consequences should there be impreciseness in the data or mistakes in the implementation. For this reason, it is important that users of wireless sensor networks are given an opportunity to visualise their sensor network in how they operate and for how long to ensure that there are no issues during and after deployment. Therefore, I decided that it would be useful to design a tool that can allow users to do that easily.

The visualisation tool that I am creating will fit the specifications of the problem researched in this project and will not, for the purposes of this project, be designed to be used in other problem types for wireless sensor network such as area coverage problems rather than target coverage. When considering existing software for this precise purpose, there is no existing related work.

5.2 Requirements

5.2.1 Functional requirements

- User should be able to choose a sensor network solution file from their files.
- User can view the sensors and their roles, the targets and the base station for the selected cover in an instance.
- User should be able to move between sensor network covers for their instance.
- User should be able to search for a specific cover number
- User should be able to see the time allocated to the currently viewed cover.
- User must be able choose at any time whether or not they want to see the communication and sensing range in the viewer.
- User can choose to see the currently active network connections between sensors and/or between sensors and targets.
- User can view progress bars that show them how far through the covers in an instance they are by the current instance they are viewing. This should be shown both in number of covers and in total time allocated.

5.2.2 Non-functional requirements

- Changing between covers in an instance should re-plot the required data with very little delay.

- Buttons for movement and choosing options should be understandable and contain fairly large and relevant text to the action that is performed.

- Data viewer should take up a large amount of the window to show the cover information clearly.

- File validation should be in place to ensure the file that is chosen for the program is suitable both in format and content.

5.3 Architecture & technologies

The clear choice of architecture for this tool is a three-layer architecture, that is, where three layers called the presentation layer, the application layer and the storage layer are defined. These layers are separated and each perform a specific task. The presentation layer deals only with what is viewed by the user, the application layer performs the main logical tasks of the program and the storage layer deals with the data involved in the problem and the persistence of this data. A closed-layer approach is suitable here where each layer can make use of only the layer immediately below itself and provide a service to the layer immediately above itself. This architecture allows tasks in the program to be split in such a way that supports information hiding whilst reducing coupling [9].

To keep the sub-tasks carried out throughout the project consistent in their implementation, I have again used the Python programming language for creating this tool. I will require two packages that can assist me in developing this software. Firstly, some kind of data visualisation package will be needed to create the viewer. I decided to use matplotlib, an open-source package designed for plotting graphs and creating figures in Python. This library is very popular and is listed by many as one of the most useful and powerful packages available openly for use with Python [34]. This library can be used to create scatter plots where points in the plot represent the nodes in our instance and the plot itself is the defined area that the sensors have been deployed in.

The second package that will be required for the development of this tool is some kind of graphical user interface module that can be used to create the required buttons and option boxes as well as display the matplotlib figure that is created within the window. It is worth noting that the matplotlib library does have GUI elements available for use but after some initial testing, the performance of these elements were poor as the actions performed were far slower than required to create a good experience for the user. Thankfully, it is also designed to be easily integrated into some well known Python GUI packages.

One of the GUI packages that matplotlib has functionalities for embedding plots in Python is TkInter. This is an interface to Tk for Python and is considered the conventional GUI for the programming language [21]. The reputation that has cemented TkInter as the traditional choice for creating graphical user interfaces along with the fact that matplotlib figures can be embedded in TkInter interfaces, this makes it an easy choice for developing the visualisation tool.

5.4 User interface

The user interface should be simple and intuitive, as it is not intended to be complicated software with many functionalities but a visualisation tool. The use case for this software is simply someone that wants to view their sensor network and how it operates. Therefore, as mentioned in the requirements, it is crucial that the main figure component takes up a large portion of the window. The rest of the window will contain the different options that the user has for moving around and choosing which information they can see in the viewer.

Whilst the choice to take up a large portion of the window with the figure was necessary to fit the requirements, it does bring with it the drawback of having little room to fit all of the required options and buttons for the user in whilst ensuring the layout remains aesthetically pleasing. I chose to use the top of the window and part of the left side of the window to fit these in. I also chose to reserve the top right specifically for the exit button. Incorporating such a button is important to ensure that when a user wishes to close the program, they can do so properly and in a way that won't crash or affect the performance of the user's device.

The progress bars that will show the user the time allocated to the covers they have viewed so far and the proportion of the total covers they have viewed at their current cover will be placed at the top centre of the program. This was a natural decision as keeping these bars close to the viewer allows for all displayed information to be kept close together so the user can intuitively look between the two main display components of the program. This leaves us with the next and previous cover buttons, and the options for showing node connections and ranges. These can be fitted to the left side of the window. The top left will display the two buttons and then the options can be listed below as check boxes.

5.5 Implementation

In order to develop this tool effectively, I took an agile approach. This involved continuously making improvements and adding new features whilst testing each as they are implemented. This approach allowed me to constantly reflect on the progress that was being made and make any adaptations as necessary.

5.5.1 Reading the solution file and plotting the covers

The absolute basic tool that I focused on implementing first was plotting and embedding a sensor network cover in the GUI and allow the user to move between these covers, re-plotting the sensor role data as a result. In order to keep the properties and methods of the program accessible and tidy, I chose to define one class to represent the main program (called 'VisualiseSensorNetwork'), another for the main page design, and a third to define the buttons used in the program.

The decision to create a class specifically for the buttons used in the GUI was made as the standard button design did not let the user know when they were hovering over a button. In other words, I wanted the button colour to change slightly when a user hovered over a button and so I defined a class for this with methods that would bind event handlers to update the background of a button when hovered over. Whenever a button was required in the main program, an instance of the 'GUIButton' class was created. Similarly, a class was created for the main page and the contents that generally will be found on this page.

The 'MainPage' class defines the components that will be displayed when the user is using the software. This page will eventually define components such as the progress bars and options checkboxes but for now, a previous cover, next cover, and exit button are defined. When the buttons for moving between covers are pressed, the method of the VisualiseSensorNetwork class 'change_cover' is called. This method takes the parameter of the current cover number, plus or minus 1 depending on the button pressed. This parameter is assigned to the current cover number member variable. Then, this parameter is checked to see if either of the buttons should be disabled. In other words, if the parameter is less than or equal to 1, then the previous cover button should be disabled, however; if the parameter is greater than or equal to the length of the covers in the solution, then the next cover button should be disabled. If neither of these conditions are true then both are enabled. The member method that will be described in the following paragraphs 'scatter_data' is then called after the old figure from the previous cover has been closed.

The first main operation that must be carried out to display a cover to the user is the retrieval of the information from the relevant solution file. This is done in the 'generate_data' method At this stage in the implementation, there was no functionality for prompting the user with a file manager GUI to choose the file they wish to use so instead, the user is asked to provide a path for the desired file through the command line that they are using. This file is then opened and the information is retrieved by calling a function defined in a separate file that was created for reading the solution file. Within this function, the file is read and the lines are appropriately split and stripped of any unnecessary characters. As mentioned previously, the solution files are designed so that the first line of the file contains the parameter information and that is then followed by sensor and target coordinates. After this information is stored in the corresponding variables, three more variables are defined to store the total time taken to find the optimal solution, the number of iterations completed, and the total value or time that was found to be optimal.

The next section of this function for reading data loops through the remaining lines in the file. The first 9 characters of the line are checked to see if they match the word 'PARTITION' and if this is the case, it is known that another cover is contained within the file. This line can be split by the colon character and the information after the colon is stored as the time allocated to that partition. The format of the file means that when this match is found, three more lines can be read to find the sensors that play inactive, relay and source roles in that particular cover. This information is in the form of the coordinates of the sensors as is consistent throughout the implementation of this project as this can be matched to an element of the sensors array that was created earlier. After the end of the text file is reached, the variables created to hold the sensor coordinates, target coordinates, base station coordinates, the time taken to find the optimal solution, the number of iterations, the final solution value, the covers and their respective allocated time are returned as they will all be used for this tool.

After these variables have been returned, the information for every sensor in every cover must be split into x and y coordinates to for use in the plot function later. This also must be carried out for the targets but as this tool deals only with stationary targets, this can be done just once rather than for every cover. Whilst it could be an expensive operation for large instances, this task is completed simply by iterating over the covers, and then over the nodes before the type of node in question is checked the coordinate splitting is performed.

At this point in the implementation where extra features were not yet considered, the only

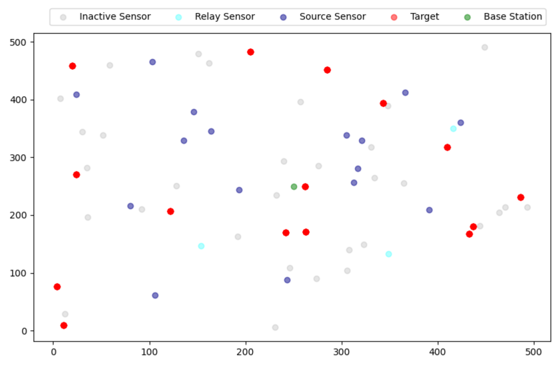


Figure 5.1: The grid that displays the nodes to the user.

thing left to do to show the first cover is scatter the data. The suitably named method 'scatter_data' executes this action. A member variable stores the current cover number that the user is viewing and this can be incremented or decremented by using the buttons created earlier. This member variable is used to find the correct element of the covers array. A figure and plot is created with ticks on both axes in increments of 100 from 0 to 500, in other words, the area that the sensors are deployed in. The data is then scattered using the scatter method included in the package and the split coordinates defined earlier and the final figure is displayed using the TkInter built-in 'grid' method for placing components in the window. This grid can be seen in Figure 5.1.

5.5.2 Including options to show the ranges of the sensors

After testing this feature worked for at least 10 of the instances chosen at random from the experiment solution files, I began implementing the next feature, options boxes for showing sensing and communication ranges. This mainly changed two methods. First, the components had to be defined in the main page class. When it comes to using checkboxes, a TkInter binary variable must be defined for each of them to essentially represent whether or not the box has been selected. When either of the boxes variables are changed, the method for scattering data is called. This means that some changes in the 'scatter_data' method have to be made.

I planned to have it work by calling the scatter method for the nodes that are relevant to the options box that had been checked but this was not plausible as the marker size was defined purely by pixels and had no relation to the plot size and so the actual range that a sensor has. For this reason, I used the matplotlib functionality for drawing shapes, specifically circles. Whenever the method is called, each of the box variables are checked to see if the box is selected. If this is the case, the corresponding sensor role coordinates in that cover are looped through and a circle is defined and appended to a circles list. The circle is defined with a radius equal to the respective range value for that option. These circles are added to the figure using the 'add_patch' method. The options can be seen in Figure 5.2, and some variations of selected options can be seen in Figures 5.3 and 5.4.

5.5.3 Including the progress bars

After again testing this new feature and finding that it was working correctly, I chose to implement the progress bars. Similar to the options boxes, they are defined in the main page using the functionality provided by the TkInter ttk module for styling. They are given a horizontal orientation and a initial value of 0. When the 'generate_data' method is called at the beginning of the

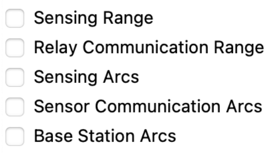
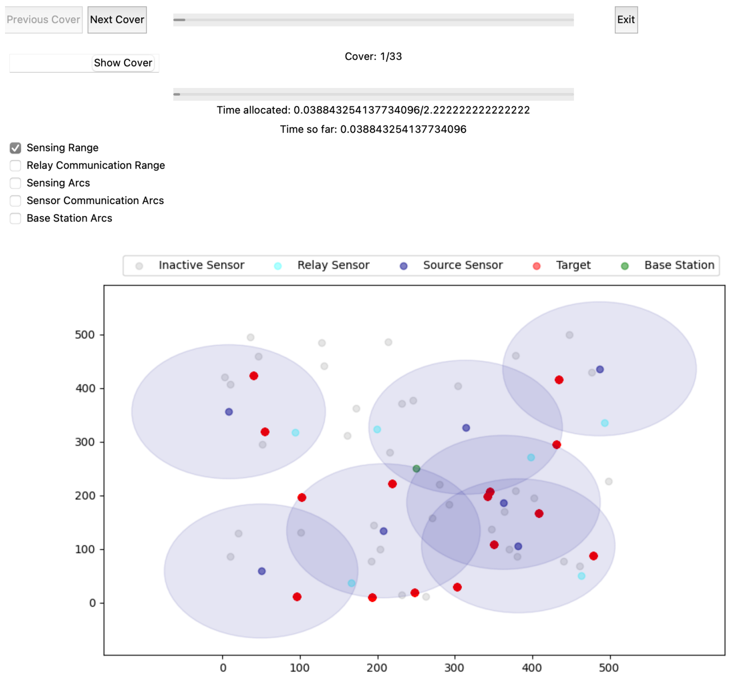


Figure 5.2: The options that the user can select.



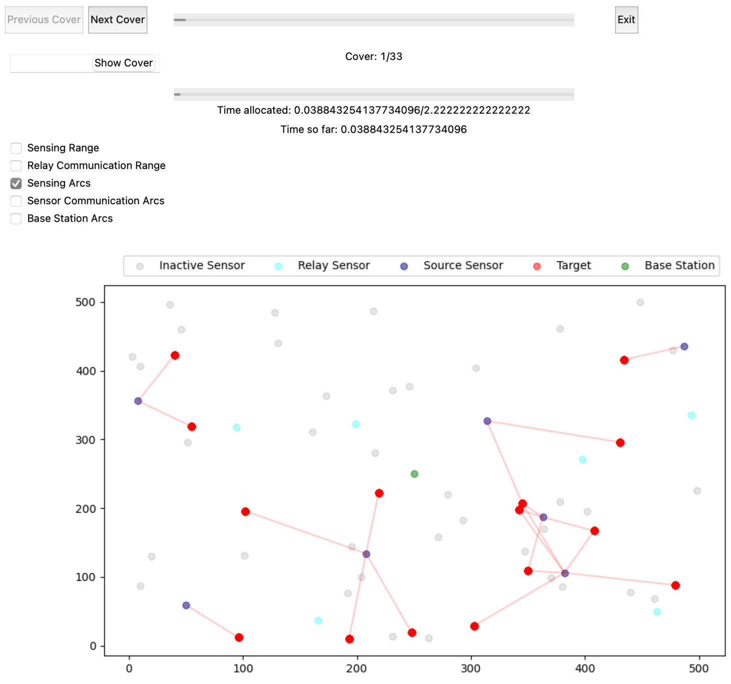
Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

[✓ Sensing Range]
Relay Communication Range
Sensing Arcs
Sensor Communication Arcs
Base Station Arcs

Figure 5.3: The interface with the Sensing Range option selected.



Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

Sensing Range
Relay Communication Range
[✓ Sensing Arcs]
Sensor Communication Arcs
Base Station Arcs

Figure 5.4: The interface with the Sensing Arcs option selected.

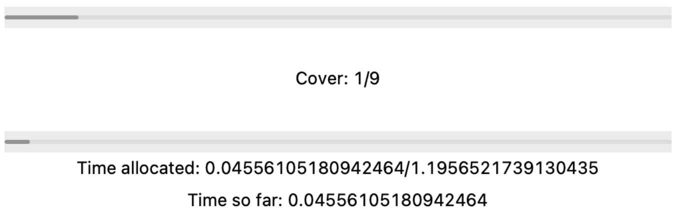

Cover: 1/9
Time allocated: 0.04556105180942464/1.1956521739130435
Time so far: 0.04556105180942464

Figure 5.5: The progress bars used in the tool.

Figure 5.6: The search bar in the visualisation tool.

program, the maximum value of these progress bars can be set. That is, the length of the covers array for the covers progress bar and the total time in the optimal solution for the time progress bar. The final change that must be made for this feature is to update the value. This can be done in the scattering method. The cover progress bar value can be set to the current cover number value whilst the time progress value can be updated by finding the sum of the time allocated to every cover from the first cover up until the currently selected cover. This will give the value of the time that has 'passed' in the network up until that point. These progress bars can be seen in Figure 5.5

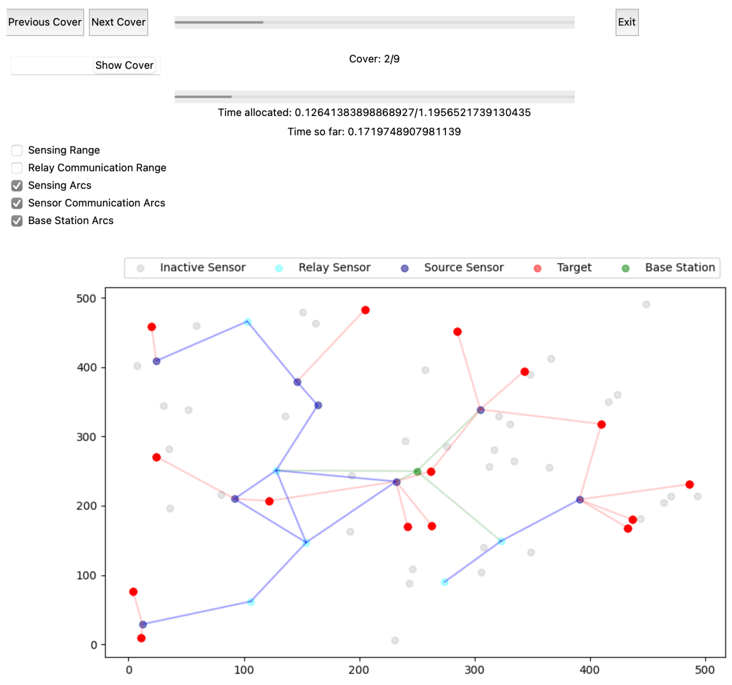
5.5.4 Adding a search bar

This feature was an important one, especially as users may be dealing with instances that are very large and hundreds of covers could exist in the final solution. Therefore, a user should be able to effectively find a cover they are looking for rather than just pressing the next button until they eventually arrive there. As is the case with all the feature up until now, the search bar and submit button are defined using the built in functionalities for these GUI components in the main page class. When a user presses the search button, the method for changing cover is called with the parameter taken from the search bar (defined in TkInter as an 'Entry') and converted to an integer. The search bar can be seen in Figure 5.6

5.5.5 Showing the active network connections

The option to view the sensing and communication ranges can be useful to see how the data gets from the targets to the base station, but in large instances, it can get messy. Therefore, I decided that an option to see the actual connections between nodes represented by lines could be a nice addition to the program.

To develop this feature, the method for calculating the arcs in the optimisation program is re-used here. This only needs to be calculated once and does not change between covers. The options for target-sensor connections and sensor-sensor connections are defined in the main page class and again, the 'scatter_data' method must be changed to incorporate these new components. In this method, the arcs are iterated over and for each arc, the type of arc is determined by an if statement that checks the type of nodes in the arc and the appropriate checkbox variable is retrieved. If this retrieved value is 1, then a newly defined 'connect_points' method is called, passing the arc and a colour (depending on the type of connection) as parameters. This method simply plots a new line graph between the two points. Originally, I had planned to just plot one line graph with all



Cover: 2/9

Time allocated: 0.12641383898868927/1.1956521739130435

Time so far: 0.1719748907981139

☐ Sensing Range
☐ Relay Communication Range
☑ Sensing Arcs
☑ Sensor Communication Arcs
☑ Base Station Arcs

Figure 5.7: The tool when each of the network arc options are selected.

the specified arcs but of course, the line graph drawn would automatically connect all points on the graph with a line rather than only the connections desired. The interface with these options selected can be seen in Figure 5.7.

5.5.6 Adding the introduction page and file chooser

Finally, I decided that passing a file path to the command line is not an ideal way to choose a solution file. I decided instead to create an introduction page where the user can click a button to open their file manager and specify a file that they wish to open. When they choose a file, the path is automatically returned and can be used to retrieve the required information. To do this, I defined a new class called ‘IntroPage’ that works the same way as the ‘MainPage’ class. In this class, a welcome message and button for choosing a file is created and placed in the window. On clicking the button, the new ‘get_filename’ method for the main program object is called. This method will call the TkInter method ‘askopenfilename’ and store the path in the file name member variable. The main page can then be displayed by calling the ‘show_main’ method where the introduction page is destroyed and the program can continue as normal.

Chapter 6

Conclusion

6.1 Summary of the contributions

The first main contribution that this project has made is the instance generation program. This software features various methods that can be used to design and write specific instances of a sensor network. The program can be altered to define many variables such as how many sensors and targets are required, specific ranges for sensing and communicating, how many total instances are required, how often parameters should change when generating multiple instances and more. Whilst originally designed purely for creating the test data for my experiment, I have adapted this file so that one-off instances can be created and written to a text file for any user that wishes to create sensor network instances to be recognised by my optimisation program.

In this project, I have investigated and experimented with two possible costs that could have a potential impact on the solution found by and performance of the optimisation program, namely inactive sensor energy consumption and target data retrieval costs. Furthermore, I have studied the possibility of implementing a cost for activating sensors that were previously inactive. Whilst I chose not to implement and test this additional cost, I have still provided ideas and formulae as to how this can be achieved should future work be conducted on this matter with less of a time constraint.

The experiment I carried out allowed me to make some clear conclusions. Firstly, it can be said that increasing the inactive consumption rate will result in a lower solution value in most cases. However, it is interesting that for some instances, the optimisation models can produce a similar or even identical solution for different inactive consumption rates by proposing covers with differing amounts of inactive sensors depending on the given rate. It can also be concluded that increasing the inactive sensor energy consumption rate does not have a significant impact on the performance of the program. On the other hand, it can be said that the new model proposed in this project for taking into account target data retrieval costs did perform slower than the original model. This is very likely down to the addition of the summation of the flow decision variables for each sensor in the pricing subproblem. Despite this, the new model was still able to find the optimal solution for many of the instances and therefore, some conclusions could be drawn. The additional cost resulted in a drastic decrease in the optimal solution for most instances regardless of the inactive consumption rate. This was also consistent for all number of sensors, targets and for both CMLP and CMLP-MR instances.

The final contribution I have made in this project is the visualisation tool I have created for viewing an instance of a sensor network that has been solved for an optimal time allocated to each cover. This tool can allow a user to see clearly how the network will function over time as well as comprehend the ways in which the nodes in the instance are able to interact using the show active network connections and show network sensing and communication ranges options. Whilst this tool is fairly basic and limited in the number of features it boasts, it has the potential to be very useful in real world situations should it be expanded on in the future to fit a specific problem area.

6.2 Future work

There are many possible adaptions and additions that could be made to the progress in this project that could be interesting. These considerations for future work can be placed into groups, the first of which is target related additions. Changes to the way that targets are regarded in the optimisation models were not considered for this project but that is not to say there are not some interesting ideas that could be explored in this area. The first idea is having moving targets instead of the stationary ones considered in this project. This could be especially interesting for applying sensor networks to domains such as monitoring wildlife where of course, some species may move often and at high speeds. The sensors must be able to adapt to this. Furthermore, moving targets could be considered relevant for problem areas where the landscape is uncertain. For example, when applying wireless sensor networks to sub-sea areas, possibly for monitoring debris or litter, it is important to consider the possibility of waves and storms affecting the position of the targets that must be monitored.

Another study that could be interesting to test these models with is adapting the models so that the goal of the sensors is to actively cover as much of the area as possible for as long as possible rather than sufficiently cover the specified targets that are placed in the area. This may be suitable for domains such as area monitoring for security purposes. The idea is that instead of covering particular points of interest, a minimum percentage of the area that the sensors must cover could be specified and the models could optimise how the lifetime of the sensor network whilst they are actively covering that percentage of the area.

Aside from ideas that involve changing the way that targets are represented and used, some ideas involving the sensors could be considered. Firstly, similar to the idea of moving targets, what if moving a sensor was possible? If the situation arises that a target is not able to be covered by a sensor because the battery of that sensor has depleted, a nearby sensor could move closer so that the target would sit within the sensing range of the active sensor. This could be considered through the idea of physically movable sensors or possibly more appropriately, sensors that can adjust their sensing range with an additional cost to the battery for doing this.

In this project, the idea of including a cost for activating a sensor that was inactive in the previous cover was introduced. I described the way that this could be implemented and the motivation behind it, however; the implementation and experimentation of this idea was not feasible given the time constraints of this project. This could be explored further in future work where there is no such constraint or if adaptions that could reduce the complexity of the proposed inclusion could be made.

6.3 Project evaluation

Throughout this project, I have achieved many small goals on top of the main task of investigating the impact of additional costs on the lifetime of a sensor network. Whilst the idea of this project was to simply make small adjustments to a model created for the optimisation of a sensor network's lifetime, the process was not so simple. In this section I will evaluate how this project was carried out and the hurdles that were faced throughout.

Firstly, it should be mentioned that the area of wireless sensor network was mostly unknown to me before starting this project, as were linear programming techniques and some of the advanced concepts of optimisation. This meant that a considerably large portion of the timeline was spent on learning and understanding the many concepts of these topics. Furthermore, in order to fully comprehend the related work in this area and to eventually implement a solution, some time had to be consumed to practice programming the techniques of various solvers and testing each to decide on which would be used for this experiment. Over the course of the semester, my knowledge of the subject has grown greatly and I am satisfied with the progress I have made in understanding an area that I was previously mostly uneducated on.

A large achievement in this project was successfully adjusting and implementing the final models used for the experimental work. This part of the project took a lot of time with many errors and frustrating problems encountered before the working solution was reached. Whilst the eventual adjustments to the model may seem small, it is worth noting that many attempts were made at how the additional costs could be modelled properly.

Whilst I believe this project has been successful overall, there are some things that in hindsight, I would have liked to have done differently. For example, I would have chosen to start basic testing and experimentation with smaller instances earlier than was carried out in this project. A large portion of the testing of my new model was carried out with large instances similar in size to what was used in the main experiment. This resulted in a lengthy process of testing an instance and waiting up to an hour to receive a result that may or may not be accurate. Consequentially, the main experiment was delayed and if this had not occurred, more clear results could have been obtained for analysing in the discussion.

Despite this, I am pleased with the contributions I have made in this project and the work that I have carried out. I believe the importance of the skills I have developed at university is how transferable they are and how well they can be applied to new topics and problems. This project has showed that this is the case, as I was able to use many of the competencies I have gained over the last four years in what was previously an unfamiliar area to me.

Bibliography

[1] AlainChabrier. Column generation techniques, Jan 2020.

[2] Th Arampatzis, John Lygeros, and Stamatis Manesis. A survey of applications of wireless sensors and wireless sensor networks. In *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005.*, pages 719–724. IEEE, 2005.

[3] Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2(1):3–19, 2005.

[4] P. Bertoldi, B. Aebischer, Charles Edlington, Craig Hershberg, B. Lebot, J. Lin, Tony Marker, A. Meier, H. Nakagami, Yoshiaki Shibata, H. P. Siderius, and C. Webber. Standby power use: How big is the problem? what policies and technical solutions can address it? *Lawrence Berkeley National Laboratory*, 2002.

[5] Kemal Bicakci, Hakan Gultekin, and Bulent Tavli. The impact of one-time energy costs on network lifetime in wireless sensor networks. *IEEE Communications Letters*, 13(12):905–907, 2009.

[6] Xing Cai, Hans Petter Langtangen, and Halvard Moe. On the performance of the python programming language for serial and parallel scientific computations. *Scientific Programming*, 13(1):31–56, 2005.

[7] Fabian Castaño, Eric Bourreau, Nubia Velasco, André Rossi, and Marc Sevaux. Exact approaches for lifetime maximization in connectivity constrained wireless multi-role sensor networks. *Eur. J. Oper. Res.*, 241(1):28–38, 2015.

[8] Fabián Castaño, André Rossi, Marc Sevaux, and N. Velasco. On the use of multiple sinks to extend the lifetime in connected wireless sensor networks. *Electronic Notes in Discrete Mathematics*, 41:77–84, 06 2013.

[9] Ernesto Compatangelo. Introduction to architectural patterns (1), October 2019.

[10] Ayhan Demiriz, Kristin P Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1):225–254, 2002.

[11] Abiola Fanimokun and Jeff Frolik. Effects of natural propagation environments on wireless sensor network coverage area. In *Proceedings of the 35th Southeastern Symposium on System Theory, 2003.*, pages 16–20. IEEE, 2003.

[12] Robert O Ferguson and Lauren F Sargent. *Linear programming*, volume 19. McGraw-Hill, 1958.

[13] Jurgen Garche, Chris K Dyer, Patrick T Moseley, Zempachi Ogumi, David AJ Rand, and Bruno Scrosati. *Encyclopedia of electrochemical power sources*. Newnes, 2013.

[14] Gurobi. The gurobi python modeling and development environment. *Gurobi*, 2021.

[15] Tomas Gustafsson. *A heuristic approach to column generation for airline crew scheduling*. Citeseer, 1999.

[16] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Mathematical Modeling and Optimization*, pages 15–27. Springer International Publishing, Cham, 2017.

[17] Philokypros P Ioulianou, Vassilios G Vassilakis, and Michael D Logothetis. Battery drain denial-of-service attacks and defenses in the internet of things. *Journal of Telecommunications and Information Technology*, 2019.

[18] Josef Jablonskỳ et al. Benchmarks for current linear and mixed integer optimization solvers. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 63(6):1923–1928, 2015.

[19] Eric Kelso. Optimization modeling: Everything you need to know, 2021.

[20] M. Keskin. Lifetime maximization of wireless sensor networks with sink costs. *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, 25:4602–4614, 01 2017.

[21] Cameron Laird. The python wiki - tkinter.

[22] Seyed Saeed Madani, Erik Schaltz, and Søren Knudsen Kær. Thermal analysis of cold plate with different configurations for thermal management of a lithium-ion battery. *Batteries*, 6(1), 2020.

[23] M.A. Matin and M.M. Islam. Chapter 1 - overview of wireless sensor network. In *Wireless Sensor Networks - Technology and Protocols*. 2012.

[24] Jiří Matoušek and Bernd Gärtner. *Duality of Linear Programming*, pages 81–104. 01 2007.

[25] Bruce A McCarl and Thomas H Spreen. Applied mathematical programming using algebraic systems. chapter 10. 1997.

[26] Bernhard Meindl and Matthias Templ. Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 20, 2012.

[27] Tenager Mekonnen, Pawani Porambage, Erkki Harjula, and Mika Ylianttila. Energy consumption analysis of high quality multi-tier wireless multimedia sensor network. *IEEE Access*, 5:15848–15858, 2017.

[28] Abhinav Nagpal and Goldie Gabrani. Python for data analytics, scientific and technical applications. In *2019 Amity International Conference on Artificial Intelligence (AICAI)*, pages 140–145, 2019.

[29] Harsh Kupwade Patil and Thomas M. Chen. Chapter 18 - wireless sensor network security: The internet of things. In John R. Vacca, editor, *Computer and Information Security Handbook*. Morgan Kaufmann, Boston, 3rd edition, 2017.

[30] Daniele Puccinelli and Martin Haenggi. Wireless sensor networks: applications and challenges of ubiquitous sensing. *IEEE Circuits and systems magazine*, 5(3):19–31, 2005.

[31] Lei Shu, Yuanfang Chen, Zhihong Sun, Fei Tong, and Mithun Mukherjee. Detecting the dangerous area of toxic gases with wireless sensor networks. *IEEE Transactions on Emerging Topics in Computing*, 8(1):137–147, 2020.

[32] R. Strichartz. The way of analysis. 1995.

[33] F. Tardella. The fundamental theorem of linear programming: extensions and applications. *Optimization*, 60(1-2):283–301, 2011.

[34] Shaun Taylor-Morgan. The 7 most popular ways to plot data in python, 2020.

[35] Robert J Vanderbei et al. *Linear programming*, volume 3. Springer, 2015.

[36] Huseyin Ugur Yildiz, Murat Temiz, and Bulent Tavli. Impact of limiting hop count on the lifetime of wireless sensor networks. *IEEE Communications Letters*, 19(4):569–572, 2015.

[37] Dimitrios Zorbas and Christos Douligeris. *Power Efficient Target Coverage in Wireless Sensor Networks*. 12 2010.

Appendix A

Visualisation Tool - User Manual

A.1 Choosing a file

When the program is first opened, the page shown in Figure A.1 is presented. Click the button to choose a solution file, and you will be met with your default file explorer. Double click the appropriate file to open it in the visualisation tool or click the file and press the ‘open’ button. An example of the file chooser can be seen in Figure A.2. If the file chosen causes an error when read, or is of an incorrect type, the original page is shown again until a suitable file is chosen.

A.2 Viewing the solution

The main viewer can be seen below the options, taking up most of the window. This viewer shows a scatter graph where the points are nodes. The legend that explains what each of the point colours mean is placed across the top of the viewer. At the top of the window, there are two progress bars. The top bar shows how far through the set of covers you are whilst at the current cover. The bottom bar shows the same but with the solution time that has passed after that cover has been active. The main viewer grid and progress bars can be seen in Figures A.3 and A.4 respectively.

A.3 Navigation

There are two options that can be used to navigate between covers in a solution. The first of which is the ‘next’ and ‘previous’ buttons in the top left corner. When the active cover is the first cover in the set, then the ‘previous’ button will be inactive and will appear faded. Similarly, the ‘next’ button will be inactive when the final cover is selected. The second option that can be used for navigating between covers, is the search bar. This bar can be seen below the buttons mentioned earlier. It works by entering a cover number in the bar and then pressing the button labelled ‘Show Cover’. If you enter a number less than one, the first cover will automatically be selected, whilst when a number greater than the amount of covers is searched for, the final cover is selected.

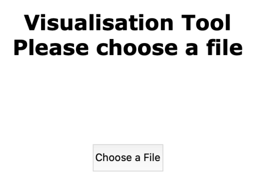

Visualisation Tool
Please choose a file

Figure A.1: The introduction page that is shown on opening the tool.

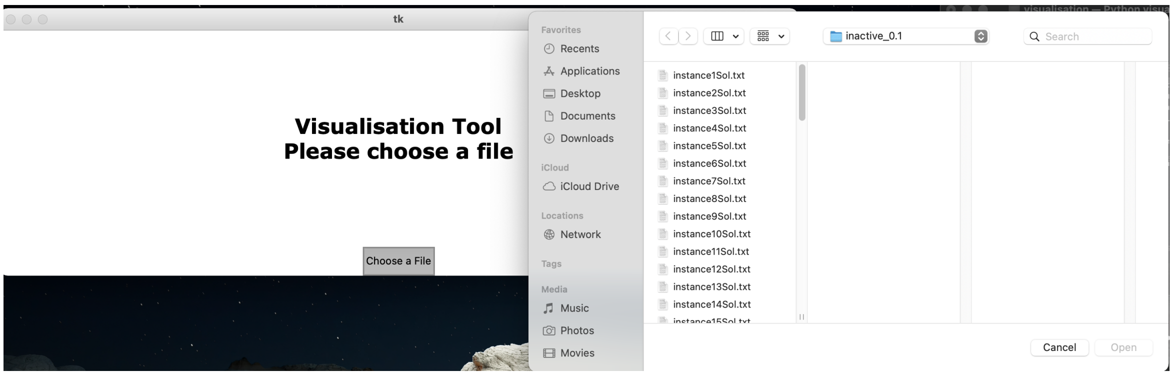


Figure A.2: The menu that is opened when the button to choose a file is pressed.

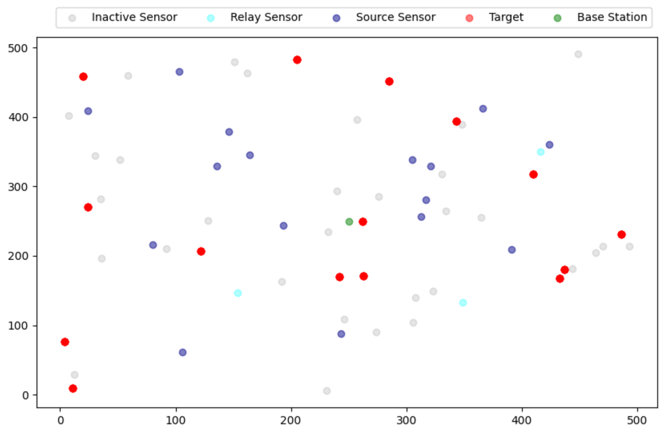
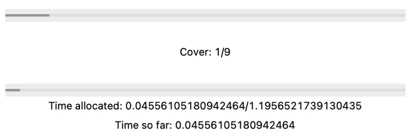


Figure A.3: The main viewer showing the solution data for a cover.

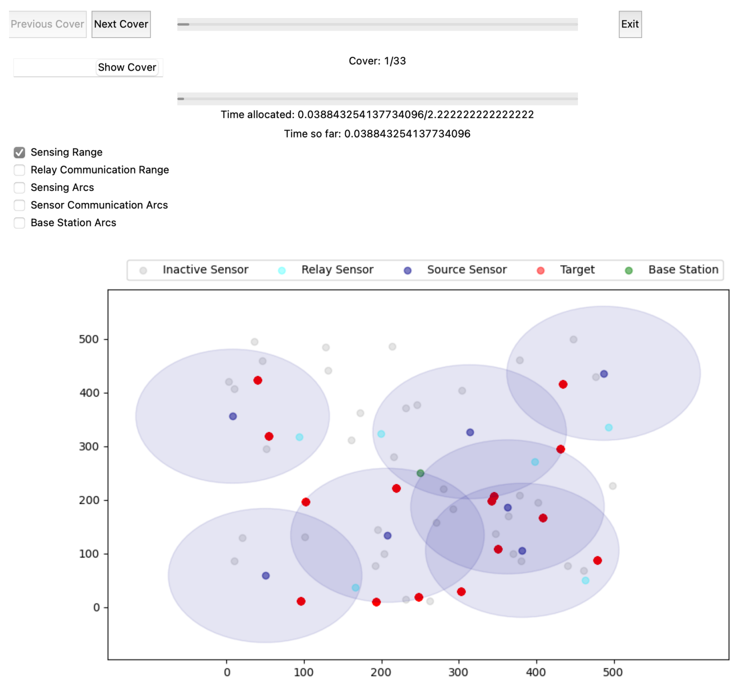


Cover: 1/9

Time allocated: 0.04556105180942464/1.1956521739130435

Time so far: 0.04556105180942464

Figure A.4: The progress bars that are placed at the top of the window.



Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

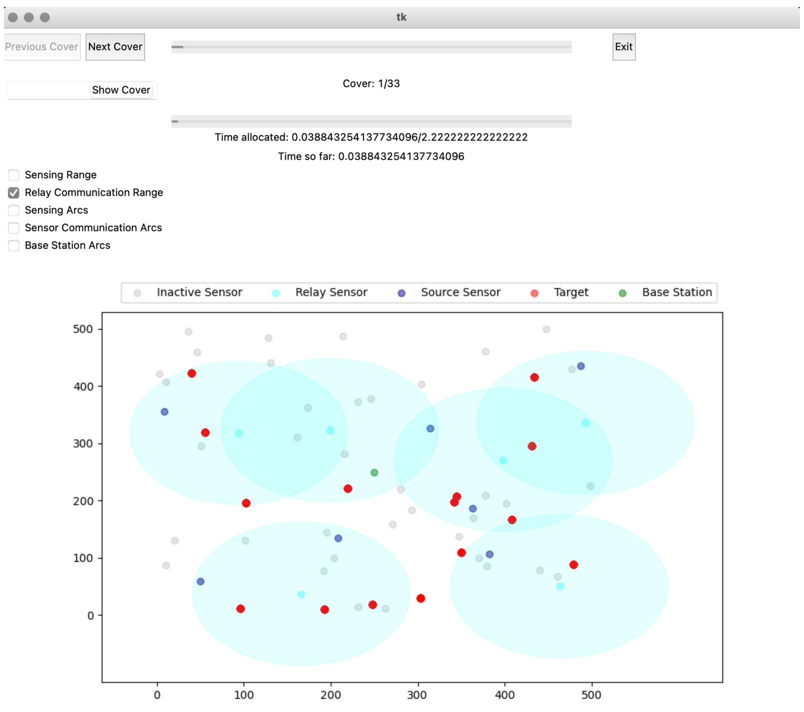
- [✓] Sensing Range
- [ ] Relay Communication Range
- [ ] Sensing Arcs
- [ ] Sensor Communication Arcs
- [ ] Base Station Arcs

Figure A.5: The visualisation tool with the Sensing Range option checked.

A.4 Options

It The options for what information can be viewed are found at the left hand side of the window, below the search box. These options are in the form of checkboxes and can all be toggled on and off. The options are as follows:

- Sensing Range - the sensing range of source sensors that are retrieving data from targets within that range (see Figure A.5).
- Relay Communication Range - the communication range of relay sensors that are purely communicating information to other sensors or the base station (see Figure A.6).
- Sensing Arcs - all arcs that exist between the active source sensors and the targets (see Figure A.7).
- Sensor Communication Arcs - all arcs that exist between sensors that are not inactive (see Figure A.8).
- Base Station Arcs - all arcs that exist between sensors that are active, and the base station (see Figure A.9).



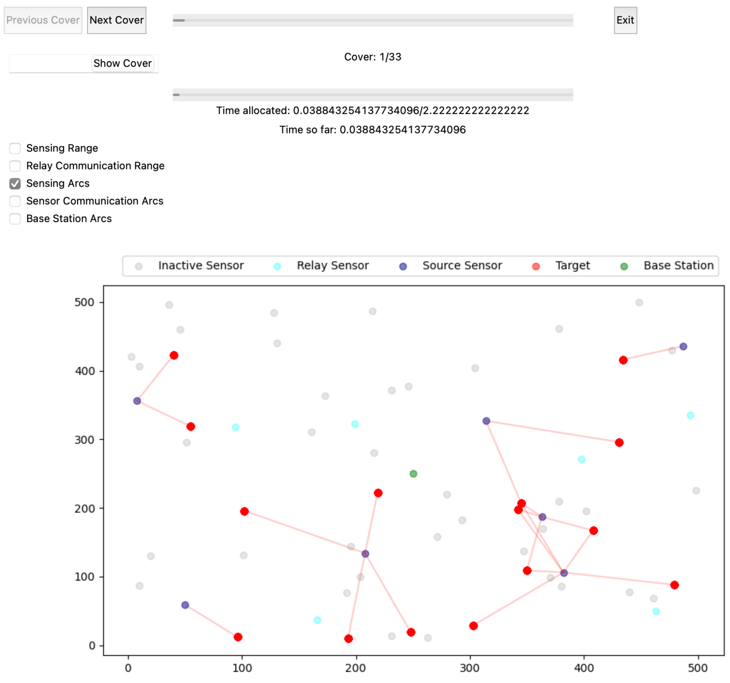
Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

- [ ] Sensing Range
- [✓] Relay Communication Range
- [ ] Sensing Arcs
- [ ] Sensor Communication Arcs
- [ ] Base Station Arcs

Figure A.6: The visualisation tool with the Relay Communication Range option checked.



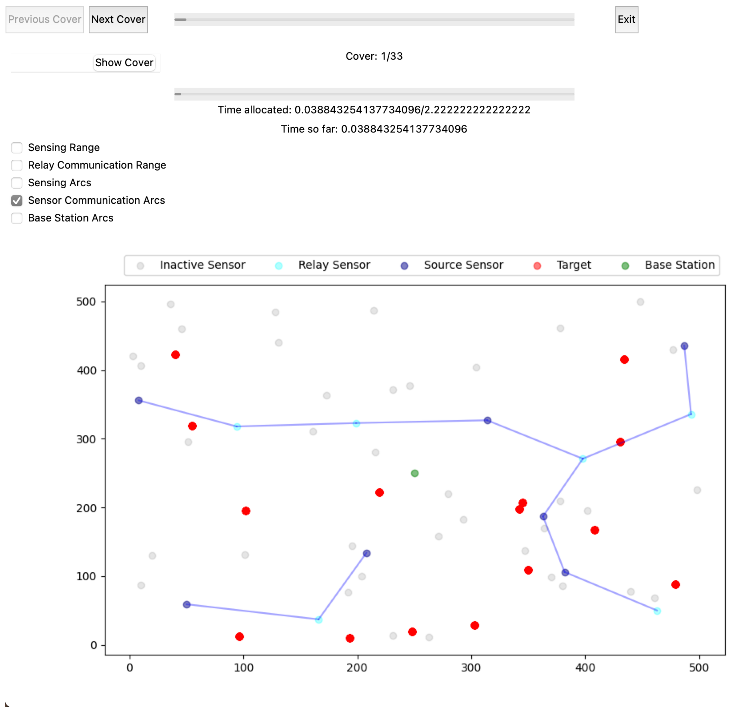
Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

- [ ] Sensing Range
- [ ] Relay Communication Range
- [✓] Sensing Arcs
- [ ] Sensor Communication Arcs
- [ ] Base Station Arcs

Figure A.7: The visualisation tool with the Sensing Arcs option checked.

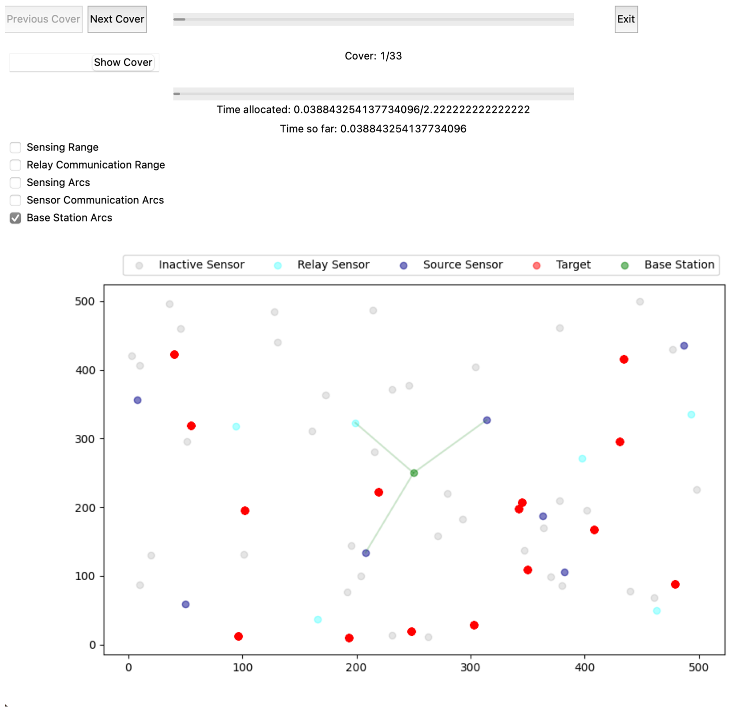


Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

Figure A.8: The visualisation tool with the Sensor Communication Arcs option checked.



Cover: 1/33

Time allocated: 0.038843254137734096/2.2222222222222222

Time so far: 0.038843254137734096

Figure A.9: The visualisation tool with the Base Station Arcs option checked.

Appendix B

Visualisation Tool - Maintenance Manual

B.1 Installation

This software can be installed by extracting the compressed submission folder.

B.2 Software requirements

In order to run the visualisation tool, some pre-requisites are required. Firstly, Python 3.0 or later should be installed as this is what the code is written in, as well as the command line command that will be needed to execute the code. Next, the packages for Python. Both TkInter and Matplotlib should be installed. This can be done using a tool such as PIP, or a package can be downloaded manually from the package's respective website. Note that TkInter may already be installed with your Python version.

B.3 Running the program

These instructions are specifically for users with Unix systems. If you are not using a Unix system, for example, Windows, please use the commands that are implemented for your machine.

This tool can be opened through the command line. First change to the appropriate directory, the 'Visualisation' folder. This can be done by using the **cd** command. Use the **ls** command to view the folders and files in your current directory if you happen to be unsure of where you must move to next. Once in the correct folder, use the following command: **python3 visualisation.py**. Note that your command for using Python may be slightly different to what is listed here.

B.4 Code listing

There are only two files for the visualisation tool and so a code listing is unnecessary. The first file is 'read_solution.py' and this deals with reading the solution file and the required cover data for the chosen solution, allowing the main file to use the correct information. The second file is 'visualisation.py' and this is the main file. This file defines each of the buttons, checkboxes, etc. It also sets up the grid where the solution information can be viewed. Please note also that a solution file (solutions from the experiment are provided in the submission, in the solutions folder) will be needed to use the program.

B.5 Known issues

Currently, the option for showing the communication range only applies to relay sensors, not source sensors. This is made clear by the name of the option but an important addition to the program could be to include source sensor communication ranges, as in the experiment, source sensors can also relay information.

Appendix C

Code Listing

C.1 Visualisation

File name	Description
read_solution.py	This file is used to read a solution text file and retrieve the information that will be needed to properly display the data in the visualisation tool.
visualisation.py	This file is the main file for the visualisation tool, that includes the code that creates the viewer, and implements the navigation and viewing features/options.

Table C.1: The files for the visualisation tool.

C.2 Experiment

File name	Description
generate_instances.py	This file is used to create instances that can be used in the optimisation program.
originalModel.py	This file is the optimisation program that uses the original models before changes were made.
modelAdditionalCost.py	This file is the optimisation program that uses the new proposed models
instances	The folder containing the instance text files.
solutions	The folder containing the solution files from the experiment.

Table C.2: The list of files that were used for the experiment.