University of Alberta

USING ARTIFICIAL INTELLIGENCE TECHNIQUES TO AUTOMATE SEWER SYSTEM PLANNING

by

David John O'Connell

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2007

Your file Votre référence
ISBN: 978-0-494-33318-1
Our file Notre référence
ISBN: 978-0-494-33318-1

Abstract

This thesis explores the use of computing science algorithms in sewer system automation. Related research can be separated into two subproblems; design and layout. Design determines pipe properties such as size, depth and slope. Sewer system layout specifies the topology of the pipe network. Many layout techniques consider only high-level connectivity between key neighborhood points. This thesis improves the automated layout process by finding detailed pipe and manhole positions. A set of primitive algorithms for placing a pipeline between two fixed points is developed. These primitive algorithms are used to develop two algorithms to minimize the entire neighborhood cost. The first uses a local greedy optimization heuristic to quickly generate high quality solutions. A second algorithm implements a branch-and-bound search to generate the best layout based on a set of fixed points. These algorithms are validated within a complete sewer planning prototype using a third party design module.

Table of Contents

List of Figures

Chapter 1

Introduction and Motivation

With the progression of artificial intelligence research, many tasks requiring skilled human expertise can now be automated. As well, many tasks can be formulated as optimization problems, often leading to solutions with costs lower than those produced by human experts. This thesis examines the application of such techniques to the sewer system planning process.

1.1 Problem Definition

A sewer is a subterranean pipe system. Each pipeline must have manholes either at a specified maximum distance, or at points where the pipeline changes direction. The sewer system must provide a path to convey waste or storm water from each manhole to a collection point, referred to as an outfall. The goal of a sewer system is to convey sewage for a set of clients.

This problem assumes that each client will be serviced as long as there is a pipeline along the corridor of each road. For sanitary and storm sewers this is a reasonable assumption. Residential and industrial sites are typically adjacent to a roadway. Therefore, each of these potential clients will be able to access the sewer system in the adjacent roadways. In addition, storm water will need to be drained from each of these roadways, necessitating a storm sewer system along the roads.

Sewer plans are required to have no loops. That is, for each manhole there is only one path to the outfall. In this thesis, loops will be removed by introducing small gaps, known as cuts, into the pipeline.

The inputs for the planning problem are:

1. Lines and arcs used for displaying the neighborhood. These delimit the curbs of the roads and define a corridor for the sewer system.
2. Hydrological information defining the input of sewage for the system.
3. A set of commercially available pipe sizes including costs per unit of length.
4. Maximum distance between manholes in a pipeline.

5. Construction costs, including factors such as manholes and ground excavation.

A complete sewer plan defines the position of each pipe and manhole. For each pipe, the plan also defines a diameter, and the upstream and downstream heights specifying how the pipe is buried. The position of pipes and manholes may be displayed within the two-dimensional neighborhood map. An example of the two-dimensional layout of pipes and manholes for a neighborhood is shown in Figure 1.1. In this figure, the dashed lines represent pipelines and the dots manholes. Each pipe in the figure contains an arrow indicating the direction of flow. Cuts can be seen as gaps at the end of two of the roads in this figure.

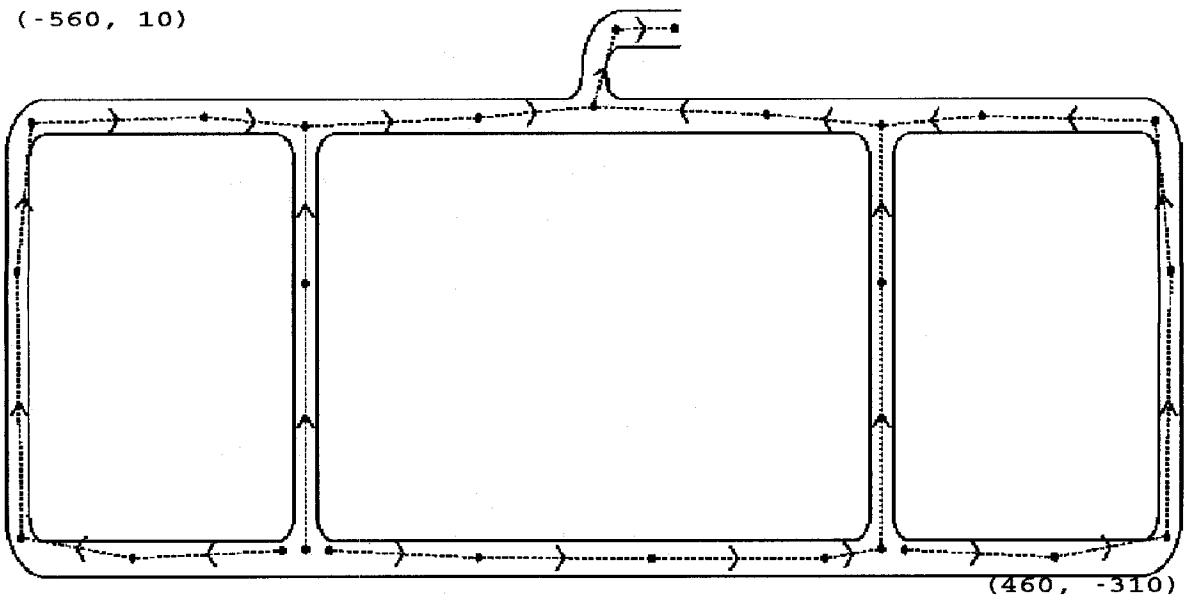

Figure 1.1: Example of a two-dimensional sewer system layout

The complete sewer system plan is typically represented as a table. A complete sewer plan corresponding to the layout shown in Figure 1.1 is shown in Figures 1.2 and 1.3. For each manhole in the sewer system, Figure 1.2 specifies the (x, y, z) coordinates and a label. To provide a frame of reference for the manhole positions in this example, the top left and bottom right coordinates are labeled in Figure 1.1. Figure 1.3 defines the pipes of the sewer system in terms of the manhole labels from Figure 1.2.

In this thesis, sewer plans are separated into two components. The layout will refer to the position of pipes and manholes in the two-dimensional neighborhood map. The neighborhood layout is shown in Figure 1.1. The design of a sewer will refer to the diameter and upstream and downstream heights of each pipe. In the solution for the example neighborhood, these quantities are presented in Figures 1.2 and 1.3.

In addition to automation, cost minimization is another goal of the planning process. The quality of a sewer plan will be based on its associated costs. The algorithms presented in this thesis will

Label	x	y	z
1	-549.74	-137.57	0.00
2	-546.62	-287.54	0.00
3	-537.54	-53.38	0.00
4	-449.97	-299.22	0.00
5	-387.57	-50.26	0.00
6	-320.03	-295.17	0.00
7	-300.04	-294.54	0.00
8	-300.00	-144.54	0.00
9	-300.00	-55.48	0.00
10	-280.05	-295.17	0.00
11	-150.11	-299.22	0.00
12	-150.07	-50.79	0.00
13	-50.04	-44.54	0.00
14	-30.66	-0.65	0.00
15	-0.11	-299.89	0.00
16	20.00	0.000	0.00
17	99.89	-49.22	0.00
18	149.89	-299.98	0.00
19	199.96	-294.54	0.00
20	199.00	-144.54	0.00
21	199.00	-55.48	0.00
22	219.95	-295.17	0.00
23	287.39	-50.25	0.00
24	349.89	-299.22	0.00
25	437.36	-53.27	0.00
26	446.45	-287.84	0.00
27	449.72	-137.88	0.00

Figure 1.2: Manholes labels and positions for sample sewer plan

Upstream manhole	Downstream manhole	Upstream height	Downstream height	Diameter
2	1	-2.23	-3.306	1.20
1	3	-3.38	-4.22	1.20
4	2	-1.38	-2.15	1.05
3	5	-4.30	-6.36	1.20
6	4	0.00	-1.30	0.83
5	9	-6.44	-7.91	1.20
7	8	0.00	-4.26	0.68
8	9	-4.34	-7.06	0.83
9	12	-7.99	-10.81	1.35
10	11	0.00	-1.31	0.83
11	15	-1.38	-2.57	1.05
12	13	-10.88	-12.96	1.35
13	14	-13.71	-16.93	1.50
17	13	-10.19	-13.64	1.50
14	16	-17.00	-20.61	1.50
15	18	-2.644	-3.62	1.20
21	17	-7.99	-10.12	1.50
18	19	-3.69	-4.15	1.20
19	20	-4.22	-6.18	1.20
20	21	-6.25	-7.69	1.20
23	21	-6.43	-7.92	1.20
22	24	0.00	-1.31	0.83
25	23	-4.299473	-6.363807	1.20
24	26	-1.381692	-2.151483	1.05
27	25	-3.380001	-4.224473	1.20
26	27	-2.226483	-3.305001	1.20

Figure 1.3: Information for each pipe in the sewer system

attempt to minimize costs of the generated sewer plans. For the sewer layout, this cost minimization is achieved by reducing the number of manholes. For the design, the cost of the pipeline and installation costs are minimized.

The aforementioned problem definition is specific for sewer systems. However, many other pipe systems, such as irrigation networks, share similar properties. The ideas specified in this thesis may be applicable to other pipe systems.

1.2 Benefits of Automation

An automated sewer system planning tool has many benefits. Planning a sewer system requires a large degree of effort. Many hours of expert labor go into producing a cost effective plan before construction begins. Automating this process liberates the engineer from this tedious exercise, and allows these talents to be better utilized in different aspects of the project. In addition, plan quality depends on the talents and experience of the engineer. Automating this process adds consistency that is independent of human skill. Oftentimes, the effort to manually produce a sewer system plan means that only one plan is produced. By automating the planning process, it becomes possible for an engineer to evaluate multiple plans. A software tool could produce several plans, possibly with different constraints, allowing the engineer to evaluate these solutions on the basis of additional criteria such as aesthetics.

1.3 Challenges of Automation

Each sewer system must adhere to a set of guidelines. Some of the basic constraints are as follows. Pipes must lie underground beneath the roadway corridors. Each pipe in the system has a maximum allowable length, depending on the diameter of the pipe. Whenever the pipe reaches the maximum distance or changes direction, a manhole must be installed. There are also constraints on how the pipe can be buried. For example, pipes must be buried at a minimum depth to prevent damage from surface pressures and frost.

The constraints presented above are relatively simplistic. The full set of constraints governing a real-world sewer plan are often more complicated. For example, pipes may not always be straight. In special cases, slightly curved pipes may be used. Also, it is rare that the neighborhood will require only a single sewer system. Commonly, a separate system is required to convey storm and sanitary sewage. Both of these systems are constrained to lie within the same road corridor. However, care must be taken to ensure these two systems do not intersect. Additionally, these systems are required to service a different set of clients, thus the path of these sewer systems may be quite different. In this case, the planning phase for one sewer system must consider the plan for the other.

A key complication is the lack of uniformity in the constraints. Each municipality has a set of constraints that a sewer system plan must adhere to. As well, each sewer system project may have

differing goals. These factors make it difficult to design a system capable of planning sewers to meet every constraint for a project.

Despite these difficulties, an automated sewer system planning tool is of great interest. A full tool would include techniques to solve problems common to each sewer system planning problem. Such a system may also allow the integration of problem-specific constraints into the solution process. Even if the results of the system do not perfectly meet the project's criteria, it provides an initial plan that can be tuned by hand. This minimizes the amount of work that the engineer has to do.

Designing a system to handle all possible constraints is a difficult task, making the full problem difficult to solve. Many aspects of the planning process are out of the scope of this thesis. Rather, this thesis examines several aspects of the sewer system planning process, and develops algorithms to solve these sub-problems. The goal is to produce algorithms that may be used as part of a system that solves the entire complex planning problem.

1.4 Research Contributions

The intent of this study is to explore how computing science techniques can be applied to a civil engineering problem. The algorithms presented in this thesis do not address all of the engineering considerations necessary to generate real world sewer plans. Rather, this work provides a set of computing science techniques useful for engineering research. The hope is that these techniques will be adopted and supplemented with the appropriate engineering expertise to generate realistic sewer system plans.

A preliminary step to this research is to represent the neighborhood data in a form conducive to computation. Techniques to transform the lines and arcs delimiting road curbs into data structures useful for computation are presented.

A basic sub-problem encountered when planning a sewer system is the placement of a pipeline between two fixed points. Several algorithms to perform this task are presented. These algorithms will be referred to as local placement algorithms. Local placement algorithms provide a key primitive operation that is used during the layout process.

Layout algorithms from the literature are discussed in Chapter 2. However, these algorithms mainly deal with the high-level connectivity for the pipe network. The details of how each pipe and manhole are placed are not considered by these algorithms. Techniques are presented to specify the location of individual pipes and manholes. This is broken into a two step process. First, a number of manholes are fixed in key locations of the neighborhood. These manhole positions are determined by global placement algorithms. Using this set of fixed manholes, the layout is completed using local placement algorithms to lay the pipeline between adjacent points.

Adding this detail to the sewer planning process has two main benefits. First, the degree of automation is increased. In handling this finer level of detail, the engineer is relieved of another planning task. Second, the computer can evaluate many more manhole and pipe placements, possi-

bly leading to a lower cost sewer plan.

Finally, this research explores how the layout techniques presented can be integrated into a system that automates both the layout and design processes. The global layout optimization techniques do not account for flow direction and avoidance of cycles. To address this problem, the integrated system incorporates the notion of cuts and flow directions. To implement these ideas, this thesis explores the use of spanning trees, each of which defines a unique network topology providing service from each client to the outfall. This provides a complete two-dimensional layout, which is fed into a design system to choose pipe diameters and upstream and downstream elevations so as to minimize the cost. [1] An initial investigation into using this system to minimize the cost of the overall sewer plan is presented.

This thesis explores algorithms to solve sub-problems of the sewer system planning problem. In particular, algorithms to specify the placement details of single pipes and manholes are explored. The intent is for these algorithms to be used as part of a system to automate the sewer system planning problem.

1.5 Outline of Thesis

An overview of the remainder of this thesis is as follows. Chapter 2 will present a brief overview of the engineering process related to sewer system planning. Following this, an overview of research related to several aspects of the sewer system planning process will be presented. In Chapter 3, the use of real-world engineering survey data is examined. This includes parsing the road layout and representing it in the graph data structure that is used by the algorithms presented in this thesis. Four separate local placement algorithms will be presented and evaluated in Chapter 4. These include a basic approach leading the placement toward the center of the road, two algorithms designed to cut corners, as well as a frontier approach that attempts to produce optimal solutions. Using the local placement algorithms presented in Chapter 4, Chapter 5 will present algorithms that place pipes and manholes for an entire neighborhood. Chapter 6 describes a prototype for a system used to solve the full planning problem defined in Section 1.1. An implementation using the local and global placement algorithms, and a sewer system design module are also discussed. Chapter 7 provides some concluding thoughts for this thesis.

[1] Note that the design algorithms used for these tests are considered out of the scope of this thesis research. Rather, the third party design solver, described in Chapter 6, is used to generate designs for the purpose of evaluating the system as a whole.

Chapter 2

Engineering Basics, Related Work and Research Context

The engineering process of planning and building a sewer system is a complex task. This chapter provides a brief overview of the steps involved in this process. The automation of several aspects of sewer planning has received a considerable amount of research attention. An overview of related research is presented. From this overview, an architecture is proposed to illustrate how existing automation algorithms and the planning tools introduced in this thesis can be combined to produce an automated sewer planning system.

2.1 Overview of Sewer System Planning

A sewer system is a subterranean system used to convey waste to one or more collection points. In most neighborhoods, there are two different types of sewer systems. The sanitary sewer conveys industrial and household waste, while the storm sewer prevents flooding by draining surface water during storms. In rare cases, these two systems are allowed to use the same system of pipes. This is known as a combined sewer system.

Building or modifying a sewer system can be considered as a four step process [39]. The first step is preliminary investigation. In this step, the specific goals for the project are defined. Broad technical and cost issues are contrasted, and a general form for the sewer system is decided. The next stage is the planning stage. During this stage, the technical details required to build the sewer system are specified. When multiple firms are bidding on a sewer system project, it is typically the detailed plan from this stage that is used to tender bids. After the planning stage is the construction phase, in which the sewer system is built from the detailed plan. Once the sewer system is built the operation and maintenance phase begins. This involves any tasks that need to be done for daily operation, as well as routine maintenance required to keep the sewer system in good operating form.

2.1.1 The Engineering Process

One of the first tasks of the planning process is to define the collection points for the system, commonly referred to as outfalls. Once the outfalls are known, the sewage flow to be conveyed needs to be determined. For a sanitary sewer system, the amount of waste water produced by domestic, commercial and industrial sources, plus the amount of liquid that will infiltrate the system is estimated. Usually, this can be estimated by knowing the population of the area and the types of business and industry in the area.

For storm sewers, the flow is estimated using hydrological analysis. Hydrological analysis is based on the hydrological cycle, which describes the distribution of water on the surface of the earth and underground. Evaporation, precipitation, snow meltage and other types of natural water movement are described by this cycle. When developing a detailed plan for storm sewers, this type of analysis is used to determine the amount of sewage that needs to be conveyed. The main considerations are rainfall events. First, the engineer usually sets a return period for analysis. For this return period, an engineer estimates the probability that a given rainfall event will occur. For example, a return period of twenty years includes all the rainfall events that are likely to occur at least once in twenty years, but would not include a rainfall event that is expected to occur once every one hundred years. Once the return period is set, the engineer collects rainfall data including rainfall volume per time and duration of the expected rainfall events. This data is used to determine the flow that the sewer system must handle. Although not as important for sanitary sewers, hydrological analysis is still used to estimate the amount of storm water that may infiltrate the sanitary sewer system. PondPack is a software tool that can be used for hydrological analysis [1].

Once this analysis is done, the placement of pipes and manholes can begin. When creating a sewer system plan, the engineer must consider the horizontal and vertical alignments. The horizontal alignment specifies where the pipes and manholes are placed within the neighborhood. It is the horizontal alignment that can be seen on the neighborhood map. Standard engineering practices usually define a conduit underneath the roadway where pipes are allowed to be placed [23]. To allow proper maintenance of the sewer system, manholes must be placed at specified intervals along the pipe. The maximum distance differs amongst the different diameters of pipe. Manholes are also typically placed wherever the pipe changes direction. For roads with a high degree of curvature, this can mean many manholes with close spacing. When allowed by the local jurisdiction, it is possible to use curved pipes to avoid closely spaced manholes. The horizontal alignment will generally be referred to as the layout in this thesis.

Vertical alignment is concerned with the depth of sewer pipes. There are several factors that must be considered when determining these depths. First, the pipes must be buried deep enough to prevent damage from the surface, including pressure from traffic and weather. Second, there is usually a minimum required separation between the sewer pipes and pipes and cables from other utilities. For example, underground power cables and clean water distribution systems may also lie

underneath the same road corridors as the sewer system.

The other issue that needs to be considered when burying pipes is the slope. Pipe slope affects both the flow velocity and the pressure within the pipes. The goal of design is to keep the flow velocity above the minimum self cleansing velocity. If the velocity of the flow is too low, deposits may build up within the pipe, obstructing the flow. However, if the velocity is high enough these deposits are prevented, and thus the system is self cleansing. Care must also be taken that the flow velocity is not detrimental to the integrity of the pipes. Most sewer systems will not always operate at full capacity. Therefore, the slope should be chosen to maintain an appropriate velocity during both low and peak flow times.

Sewer systems where the slope of the pipes alone convey the sewage are known as gravity sewer systems. In some cases, it is not possible to convey the waste using only gravity. In these cases, pumps can be added to the system to maintain the flow of sewage. However, it is desirable to avoid pumps whenever possible, as it adds considerable expense to the system. The process of determining the slope, depth and size of pipes will be referred to in this thesis as the design process.

Before construction, the candidate sewer system must be analyzed to make sure it has the structural integrity to handle the different expected loads. Hydraulic analysis is used to evaluate the force and movement of the sewage in a sewer system design. Typically, this process estimates the pressure at the entrance to each pipe, known as the head pressure. Losses in pressure from friction and components such as manholes are considered. If the minimum pressure cannot be kept throughout the sewer system, it may be necessary to add pumps to regulate the pressure.

In practice, a hydraulics evaluation software tool is used to evaluate the system under different levels of flow and with different failure conditions. Examples of software which can be used for hydraulic analysis are Storm CAD [2] and Hydra [3].

Much of this thesis deals with sewer system planning. However, many ideas are applicable to other types of pipe networks as well. One very similar case is that of water distribution networks. The main difference between these two types of networks is flow direction. In water distribution networks, the source is analogous to the outfall for a sewer system. Water flows from the source to each of the nodes in the network. Water distribution networks must be designed such that the demand at each node is met, and a minimum head pressure must be maintained. Many of the techniques for planning a sewer system are the same for a distribution system. Therefore, research related to distribution systems will be considered. Similarly, many ideas produced from the proposed research will be applicable to water distribution networks.

The process of preliminary investigation, planning and building of a sewer system network has many more factors than those mentioned above. The introduction in this section was purposely brief, intending to give an appreciation of the process and considerations of such a project. For more technical information consult civil engineering texts such as [23] and [39].

2.2 Related Work

A literature review has revealed that the optimization process for sewer system planning can be separated into the two interrelated problems of design (vertical alignment) and layout (horizontal alignment). This section will introduce both of these problems and discuss techniques used in the literature to solve both individually and co-operatively. Layout is further sub-classified as edge selection and manhole placement. The relation of the work in this thesis to the existing sewer system automation literature will be explored. An often overlooked optimization consideration, the placement of intermediate manholes, is discussed. A modified framework to solve the complete sewer system planning problem, including intermediate manhole placement, is presented. Intermediate manhole placement and its place in the planning process are also discussed.

2.2.1 Sewer System Optimization

In the literature, the main goal of sewer planning is to produce the plan with the lowest cost, while meeting the specified engineering constraints defined for the project. Cost considerations may differ between projects, but typically include factors such as infrastructure, excavation, operation and maintenance costs.

Traditionally, the research community has viewed this planning problem as the two interrelated problems of design and layout. The predominate goal of the design phase is to determine the depth, size and slope of each pipe in the sewer system. Techniques to calculate these quantities have received a large amount of research attention, see for example [10, 27, 30, 37]. In addition, some techniques also consider the placement of special facilities such as lift stations, pumps or treatment plants in the design phase[7, 12, 20].

The layout of a sewer system can be viewed as the portion of the plan represented on a two-dimensional road map. Positions of outfalls, manholes, pipes and other similar infrastructure components are expressed in the layout. This thesis explores the generation of layouts using two phases; edge selection and manhole placement.

The edge selection process defines a high-level network topology for the sewer system. This network topology specifies the core backbone for the sewer system. The backbone defines the primary pipelines for the sewer, used to convey both local and upstream flows. Consider a multigraph $G = (V, E)$, where the term multigraph is used to indicate the potential of multiple edges between two vertices. The vertices (or nodes) in V represent selected fixed points from the outfalls, manholes or treatment plants. Edges in the set E represent potential pipelines between nodes. Edge selection algorithms find a spanning tree in G, which describe the core backbone for the sewer system. Edges absent from the spanning tree are still part of the sewage network, but will contain a single pipeline connected to only one of the adjacent vertices, and thus are used to convey local sewage flow only.

Consider the example neighborhood shown in Figure 2.1, where the intersections are labeled 1 to 6. Figure 2.2 shows a multigraph G that corresponds to the neighborhood shown in Figure

2.1. This graph represents the case where there is one fixed vertex per intersection and the roads represent the edges of G. [1] Edge selection algorithms are used to find a spanning tree representing the core backbone of the neighborhood. This constitutes a directed network, where each edge in the tree flows toward the outfall. For example, Figure 2.3 is a potential edge selection result, where the outfall is described at node 6.

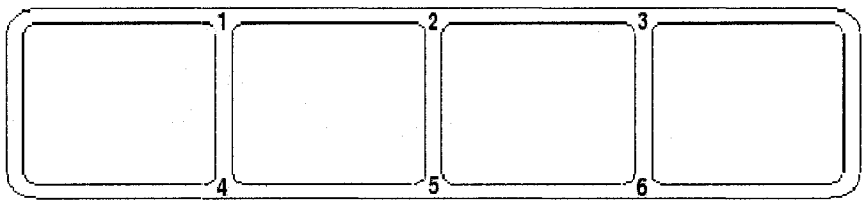

Figure 2.1: Sample neighborhood

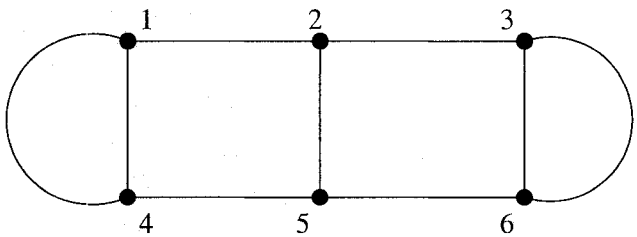

Figure 2.2: Multigraph corresponding to sample neighborhood in Figure 2.1

Part of the research presented in this thesis will examine the layout of the sewer system in more minute detail. Consider the edges in Figure 2.2. Each of these edges represents a pipeline between two nodes. However, this pipeline may be composed of multiple pipes and manholes. For example, Figure 2.4 represents the details of how a pipeline might be laid between fixed nodes 1 and 2 of Figure 2.2. How the pipeline is laid between two fixed nodes affects factors such as the number of manholes needed, and the depth of the pipes used. As a result, the cost of different pipelines may vary. This thesis examines techniques to minimize the cost of the pipelines between fixed points. The approach taken is to determine the position for the intermediate manholes that define the pipeline. This aspect of the layout problem will be referred to as the manhole placement problem.

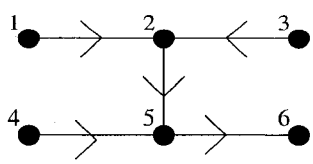

Figure 2.3: Example topology of a sewer system

[1] Chapter 3 provides more detail on the neighborhood multigraph used for this thesis, and how it is obtained.

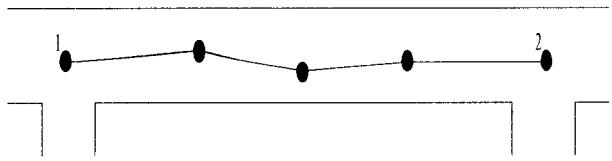

Figure 2.4: Pipe layout between two fixed points

Both design and layout are heavily interdependent. To produce an optimal plan, both design and layout must be considered simultaneously. Consider a simple change in the network topology. Let v_i denote the vertex i in the topology of a sewer system. In Figure 2.3 consider the pipe $\overline{v_2v_5}$. In the diagram shown, it accepts flow from vertices v_1 and v_3 as well as local flow. However, if the layout is modified by removing pipe $\overline{v_2v_3}$ and adding pipe $\overline{v_3v_6}$, then the pipe $\overline{v_2v_5}$ no longer receives flow from node v_3. Even though $\overline{v_2v_5}$ existed in the previous layout, the diameter of this pipe may have to change to handle the change in flow in a more cost effective manner.

Despite this interaction, much of the literature involves the optimization of the design based on a fixed layout. This is due to the perceived difficulty of the layout problem. As recently as 2004, Prasad and Park state that "*Most pipe optimization methods have not considered the layout optimization along with the cost due to the extreme complexity involved ...*" [25]. However, current research trends examine how both design and edge selection can be optimized in a cooperative fashion. An overview of solution techniques for both edge selection and design is presented in the following sections.

2.2.2 Design Optimization

Many design optimization techniques consider the layout as fixed. The main goal of design optimization is to determine pipe slopes and diameters that minimize the cost of the sewer plan. Other objectives, such as the placement of pumps and treatment plants, may also be addressed during this optimization.

Design optimization is commonly modeled as a cost minimization problem. The objective function for this problem represents the cost of the network, which may include infrastructure and maintenance costs. The control variables in this formulation are the diameter and slope of the pipes. The objective function must be optimized with respect to several constraints dictated by engineering rules. Although the specific constraints vary between problem formulations, the underlying goals are often the same. The following describes three categories of constraints commonly used in the design optimization process.

Diameter constraints govern valid pipe sizes. The most common constraint in this category ensures solutions are comprised of commercially available pipe sizes. Some formulations impose a size interval by defining a minimum and maximum pipe diameter [7, 30]. This allows the solver to consider diameter as a continuous variable, making the problem easier to solve by linear/non-

linear programming techniques. However, to be of use, these techniques must define a scheme to convert continuous pipe sizes to available discrete sizes. This rounding process may lead to a suboptimal solution. Design solvers generating commercially available pipe diameters have been developed using techniques such as dynamic programming [10, 20, 37] and genetic algorithms[28]. Sometimes, an additional diameter constraint is included to ensure that each pipe is at least as large as each of its corresponding upstream pipes. This constraint is based on the assumption that sewer system designs adhering to this constraint are more likely to handle the required flow at a reasonable cost.

Engineering principles impose constraints on how pipes can be buried. For design optimization, these are represented as depth constraints. Most formulations define a minimum depth to avoid pressure from surface factors such as traffic and snow. As well, in order to ensure the majority of flow is carried by gravity, a constraint is imposed on pipe elevations. Basically, this states that all pipes entering a node must do so at an elevation equal to or higher than the outgoing pipe.

Another category of constraints are the flow constraints, which are concerned with flow velocity and pressure. Most sewer system designs impose a range on the flow velocity. At the minimum end of this range is the self cleansing velocity. This velocity must be exceeded to prevent buildup in the pipes. In addition, a maximum velocity is enforced to avoid damaging the pipes. Another common constraint is minimum head pressure. The head pressure for a pipe is the flow pressure as sewage enters. Normally, the goal is to keep this pressure above a minimum value at each node to ensure adequate flow through the entire sewer system. The flow velocities and pressures are dependent on the design of the sewer system. Therefore, the optimization techniques often use a hydraulic solver to calculate these values for candidate solutions.

There are many algorithms that use this basic problem formulation. An overview of several design solving approaches is as follows. Dajani *et al.* developed a mathematical program to solve for the optimal pipe diameter and slope [10]. The objective function is of the form $C = a + bD^2 + cX^2$, where C is the cost of one pipe link, D is the diameter of the sewer pipe, X is the depth of the pipe and a, b and c are regression coefficients. Using the Manning Equation[2] for flow velocity, the pipe diameter is written in terms of the slope. Substituting this new expression for the diameter into the cost function allows the objective function to be written in terms of the slope only. Therefore, the resulting optimization involves only one variable. The authors have evaluated separable, dynamic and geometric programming as methods to solve this problem.

Separable programming is described by the authors as a *"technique for handling certain types of nonlinear functions within the framework of a general linear programming format"*. This technique may be applied as long as:

1. Each nonlinear function is a linear combination of single variable functions,

[2]The Manning Equation is an equation for gravity driven flow velocity in a pipe, where this velocity is a function of hydraulic radius, roughness and slope of the pipe[23].

2. Each function is expressible as a piecewise linear function, and

3. It is possible to reach a local optimal if the objective function is concave when minimizing, or convex when maximizing.

This approach is good for small problem instances meeting these criteria, but the authors note that this technique is not scalable [10]. The complexity increases rapidly as the number of pipes in the system increases. The authors also observe that dynamic programming provides solutions that use commercially available pipe diameters and is independent of the specific objective function. Geometric programming techniques can provide solutions for problems involving both nonlinear objective functions and constraints. These techniques find a solution by calculating an increasing sequence of lower bounds that converge to the true minimum value. To formulate the problem as a geometric program, the objective function must take the form of a general posynomial.[3] The authors claims it is not difficult to express most problems in this form.

Although no tests are provided, Dajani *et. al* [10] cite a claim from Merrit and Brogan [22] that their dynamic programming method reduces the cost of the sewer system by 10-20% over manually produced sewer plans, and from Velon [36] that as much as 35% reduction could be achieved from separable programming.

Walters introduces an algorithm that considers manhole position in addition to pipe depth and size[37]. To simplify the problem, a rough solution is given, specifying the initial number of manholes and the flow direction. An example of this input is represented in part a) of Figure 2.5. For each manhole, a grid of possible positions is defined. Manholes may coincide as long as the flow is not disrupted. Therefore, the optimal solution may have fewer manholes than the rough solution. An example of the elimination of a manhole is given in part b) of Figure 2.5. Dynamic programming is used to find a solution, with the manhole position and pipe depth as the variables. Pipe diameters are chosen as the smallest diameter that can handle the required flow. The manual storm sewer design for a housing estate is compared to the results generated by the algorithm. The cost of the networks generated using the algorithm were 4% lower for a low flow network and 7% lower for a higher flow network.

Genetic algorithms are a popular tool used to compute sewer networks [15]. Savic and Walters introduce a genetic algorithm to find pipe diameters that result in the minimum cost water distribution network [27]. The authors describe their solution as a *gray-coded genetic algorithm*, but neglect to describe exactly how solutions are coded. The objective function for a population member is the cost of the coded sewer system. The authors implement this method using pipe cost as the cost for candidate solutions. Each node of each candidate solution must meet two constraints. First, the flow continuity constraint ensures the portion of the flow consumed by the node does not exceed the water delivered to the node. The second constraint enforces the minimum flow pressure at the head of each

[3] A general posynomial is a function of the form $g = u_1 + u_2 + \cdots + u_n$, where $u_i = c_i t_1^{a_{i1}} t_2^{a_{i2}} \cdots t_m^{a_{im}}$. In this formulation c_i is a positive constant, the a_{ij} values are real constants and t_j are variables.

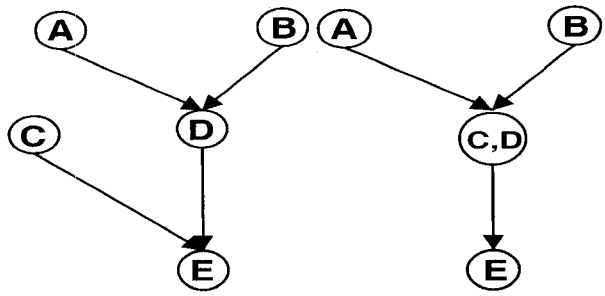

Figure 2.5: a) Example input flow b) Flow with manholes coinciding

pipe. As genetic algorithms are not designed to enforce constraints of this form, a hydraulic solver must be used to verify each candidate solution. In order to guide the search, infeasible solutions are allowed to remain in the population, but with a penalty. This genetic algorithm was applied to several sizing problems in the field, and the authors state that the solutions "*compared favorably in terms of cost and minimum head requirements to those obtained by several other techniques*". One of the drawbacks of this approach is the computation time required by the hydraulic solver. The verification process is computationally expensive and must be run once for each candidate solution. Vairavamoorthy and Ali follow a similar model, but offer several improvements [35]. First, candidate solutions are coded with real numbers, providing greater flexibility than techniques using binary strings to encode solutions. In particular, the goal is to eliminate situations where there are more binary encodings than actual solutions, and thus to avoid the overhead of filtering invalid solutions. Second, the hydraulic solver is improved using a linear transformation function. This technique produces solutions similar to the previous genetic algorithm approaches, but the verification stage requires much less computation.

Despite the promising results reported by many authors, these optimization approaches have not been adopted for every day use by the engineering community[7, 31, 35, 37]. Several reasons have been proposed for this. First, some of these solutions work well on small neighborhoods, but do not scale to large ones. Second, these techniques are not flexible enough to handle the wide range of constraints that can arise when designing real neighborhoods.

Taber *et al.* explore why design techniques have not been adopted in the design of water distribution networks[31]. They claim that these techniques are not designed to handle database information used by engineers in the manual design process, and there is a lack of user interface for data manipulation and display. The authors explore the integration of design approaches with a GIS system for the efficient gathering of information and improved graphical output of the design.

Chau *et al.* present a knowledge based system approach to encapsulate "*...rules of thumb, heuristics, judgment, code of practice and previous experience of the designer*"[8] . A semi-automated design tool was developed using a blackboard architecture. This architecture stores many different pieces of knowledge from many different sources. The blackboard acts as a global context, keep-

ing the current state of the solution, and applies the appropriate knowledge based on this state. This knowledge base is coupled with a user-friendly interface that provides support to the engineer during the design process. To conduct tests, a typical storm drainage network with 39 manholes and a number of secondary and tertiary sewage collection branches was designed using standard engineering practices and with the assistance of the tool. Using the tool, the number of man-hours was reduced from forty to one. Engineers were polled on the usefulness of the system, and results showed that the average engineer agreed that this tool was effective. Despite the perceived usefulness of this system, semi-automated tools have yet to be adopted by engineering practitioners.

2.2.3 Edge Selection Optimization

Many authors optimize the sewer system layout considering only the high-level connectivity [14, 29, 38]. This section presents an overview of several types of edge selection techniques proposed in the literature.

An intuitive approach to edge selection is to apply standard spanning tree algorithms to G. The difficulty with this approach is that the cost for each edge in the graph is not fixed. There are several factors that affect the cost of an edge. For an edge, the connectivity of the remainder of the network affects the cost, as modifying the connectivity might change the diameter of the pipe represented by the edge. In addition, design factors play a role in determining the cost. Different sized pipes have different costs, while the slope of the pipe affects the length of pipes needed as well as excavation costs. To be effective, an edge selection algorithm must address these issues.

Tekeli and Belkaya state that the optimal solution will be a shortest path spanning tree rooted at the outfall, since the goal is to find the shortest path from each node to the outfall [33]. They present a technique that tries to best utilize gravity to convey the sewage while finding a spanning tree with a short path. The first step to this algorithm is to define the flow directions each edge may assume in the final tree. Typically, this is done based on the topology of the neighborhood. The direction of each edge is set such that the flow direction matches the slope of the terrain. The only exception to this is nodes lying at local minima. To prevent these nodes from acting as sinks, the pipes adjacent to these nodes may assume either flow direction in the final tree. Once the directions have been set, a shortest path algorithm, such as Dijkstra's Algorithm, is used to find the shortest path between every pair of nodes. The authors observe that sometimes this algorithm does not result in a spanning tree. To remedy this, additional edges in the original graph are allowed to assume one of either flow directions in the resultant tree. This process is continued until the algorithm produces a spanning tree.

For this algorithm to be effective, a good edge cost metric is required. The authors stress the difficulty of this, since the cost of related design properties has yet to be determined. It is claimed that common engineering practice lays pipes parallel to the ground surface where possible, limiting the slopes to an acceptable minimum in heavily contoured areas. These standards motivate three cost

metrics. The first is horizontal length (HL). This measure chooses the pipe of the minimum length. Over flat surfaces, this means that each pipe will connect to the node at the minimum allowable elevation. Contoured surfaces are difficult to deal with, and thus not considered in this measure, making the depth of these pipes unpredictable. The second measure considers steepest descent for the inverse slope (IS) of the pipe. The measure can be expressed as $\frac{L}{\Delta Z}$, where L is the length of the pipe and ΔZ is the difference between the upstream and downstream elevation of the pipe. Finally, the last metric considers the excavation cost associated with an edge. Since the design phase has yet to be executed, the exact excavation costs of the pipes are not known. However, this value is estimated by assuming the upstream end of the pipe is buried at the minimum depth, and the slope is the minimum possible depth. Under this assumption, the excavation depth can be represented as $E_X = \frac{1}{2}(D_1 + D_2)L$, where D_1 is the upstream pipe depth, D_2 the downstream pipe depth and L is the length of the pipe.

Results for this technique are compared to manual layouts produced by senior civil engineering students. The edge selection algorithm described above was applied, and the design for the sewer system was found using the design algorithm described in [32]. Of the three metrics, only the E_X measure exhibits clear improvement showing a reduction of as much as 15.5% in excavation costs compared to the manual plan. The other metrics were typically on par with, and occasionally better than, the manually generated plan. At the time this paper was published, this algorithm could not be applied to larger neighborhoods due to memory constraints. To produce sewer system plans for these neighborhoods, the large neighborhoods were broken into several sub-neighborhoods. This approach was tested on a neighborhood with 312 nodes and 514 possible connections. From the author's test, employing the sub-neighborhood approach does not generate layouts better than those manually generated[33]. In some cases the sub-neighborhood approach produces better layouts and vice versa. This approach was only used to cope with the high memory demands of applying the method to an entire large neighborhood. Modern day computers are unlikely to be affected by these memory issues.

A number of authors explore optimization of edge selection. Many of these authors apply genetic algorithms. Walters and Lohbeck examine how to encode valid trees as binary and integer strings for use with GAs[38]. This encoding ensures that solutions are spanning trees by only allowing a single upstream connection for each node. For the binary string encoding, each node in the graph has one or more bits representing which of the potential upstream pipes is connected. Thus, if there are two possible pipes, then only one bit is necessary. However, for three or more possible interconnections, additional bits are necessary. This encoding has a major flaw. If the number of interconnections is not a power of two, the resultant bit strings will either have combinations with no meanings, or there will be multiple strings that define the same connection. The authors address this flaw by assigning multiple bit representations to some connections, thus avoiding meaningless strings. However, by using this approach, pipes that are represented by more than one string combination are more likely

to propagate to further generations. This problem is alleviated by using an integer representation, where there is a single integer for each node representing the upstream connection.

For this approach, the authors calculate the cost of each edge as

$$cost = length \times \sqrt{flow}$$

It follows that the cost of a tree, C_x, is the sum of each of selected edge. Before the reproductive phase, the fitness of each tree is assigned using the following fitness function

$$f(C_x) = 1 - \frac{C_x - C_{min}}{C_{max} - C_{min}}$$

where C_x is the cost of the current tree, C_{min} is population's lowest tree cost, and C_{max} is the population's highest tree cost.

Smith and Walters present a GA approach with a different crossover operator[29]. They remark that each child should inherit all edges shared by both parents, and fairly inherit from the unique edges of both. This is done by randomly selecting half of the edges from one parent, and half of the edges from the other parent. However, this approach is likely to produce many disconnected graphs. Instead, the authors present a technique that increases the probability of the progeny being trees. To do this, the unique edges of parent one are paired with the unique edges of parent two if each edge links a common set of nodes in the respective layout. That is, an edge e in parent one is paired to an edge $e^{'}$ in parent two if e connects subsets such that $S = S_1 \bigcup S_2$, and $e^{'}$ connects subsets such that $S = S_1^{'} \bigcup S_2^{'}$. Note that these pairings are not unique. An edge in parent one may be mapped to multiple edges in parent two and vice versa. These pairings are represented as a bipartite graph, where each side represents a parent, and there is a node for each unique edge. Each pair of edges in $P1$ and $P2$ linking the same subsets is represented by an edge in the bipartite graph. To produce a tree, a matching algorithm is used to pair edges for each node. This matching is used to produce children by randomly interchanging matched edges between parents, producing two networks with the proper number of edges.

Geem *et al.* apply an evolutionary search technique they refer to as Harmony Search[14]. This technique uses the analogy of harmonies in music, that are composed of several different building blocks. Each tree is considered as a harmony divided into n parts.

New harmonies are improvised by combining parts from previously appealing harmonies. The algorithm keeps a harmony memory which stores the m best harmonies (trees). During each iteration, a new harmony is improvised by randomly choosing each component from a harmony in the harmony memory. The major difference between this harmony search and GAs is that each new harmony may have more than two parents. If the resultant tree is better than the worst tree in harmony memory, then it is added to memory and the rest of the harmony memory is resorted. Otherwise, it is disposed of. This process continues until a stopping criteria is met.

While some authors choose to optimize edge selection alone, others attempt to integrate both design and connectivity considerations in the optimization process.

Hassalani and Dandy introduce a GA approach to simultaneously optimize connectivity and pipe sizing for a water distribution network[16]. The network is encoded using an integer string, divided into three components. The first component represents the tree, with an integer for each node representing the upstream pipe. Since only one upstream pipe is defined for each node, each string is guaranteed to encode a tree. The second component encodes pipe sizes, with the integers representing a commercially available pipe. Finally, the last integer in the string encodes the pump driving water through the network. For each member of the population, hydraulic analysis is applied to determine if minimum pressure constraints are violated. Members of the population violating these constraints have a penalty term added to their fitness cost, allowing them to contribute to further generations with a reduced probability. The authors evaluate this approach by applying the genetic algorithm to a network with 12 nodes and 30 possible node connections. In this best case, the cost was reduced from $190,000 to $106,534.

Li and Matthew present an iterative algorithm that optimizes edge selection and design in a co-operative fashion[20]. The entire sewer system planning problem is modeled as the following non-linear program

$$F(Q, H_1, H_2, \Phi)$$

used to calculated the values for each Q,H_1,H_2,Φ, where Q is a vector defining the flow, H_1 a vector of upstream pipe elevations and H_2 a vector of downstream pipe elevations. For node i in the graph, the Boolean Φ_i specifies whether or not a pumping station is present. This function is minimized subject to the following constraints.

1. Flow continuity equation $AQ + q = 0$ must not be violated, where A is the incidence matrix,[4] Q is the network flow rate vector, and q is the flow rate vector for local input,
2. $\Phi_i = 1$ if node i has a pumping station, else $\Phi_i = 0$,
3. Minimum and maximum flow velocities are enforced,
4. Minimum pipe depth,
5. Upstream elevation of downstream pipes equal to or lower than the upstream elevation of upstream pipes, except for nodes with pumping stations,
6. Pipe sizes are limited to commercial diameters,
7. Maximum proportional water depth, and

[4]This incidence matrix is a square matrix representing connectivity between nodes. If a connection exists between nodes i and j, then the entries in the i^{th} row and j^{th} column and the j^{th} row and i^{th} column will be one, otherwise these entries are zero.

8. Minimum elevation of outlets.

There are several reasons this formulation is very difficult to solve directly. One reason is that both Φ_i and the pipe diameters are discrete variables. Another reason is that the incidence matrix A, which depends on the specific tree, is not known *a priori*. To overcome this two sub-models are introduced. The first sub-model assumes a fixed layout and minimizes the design. It is represented in the following equation:

$$F = min_{H_1,H_2,\Phi} F(H_1, H_2, \Phi)$$

In this sub-model, the incidence matrix A, and therefore the flow Q are held constant. To find the pipe slopes and diameters, a technique known as Discrete Differential Dynamic Programming (DDDP) is used [17, 21].[5]

The second sub-model fixes H_1, H_2, Φ along with the pipe diameters D. The corresponding objective function is shown in the equation below:

$$F = min_Q F(Q)$$

To minimize this objective function, the authors introduce a technique called the searching direction method. For this technique, each pipe is either a tree branch or chord branch. Tree branches are the edges that are part of the spanning tree of the layout, while the remaining branches are designated as chord branches. Instead of removing chord branches entirely from the network, they are kept to convey flow from the local area. To ensure these pipes are not part of the main network, a cut is introduced at one extremity of the pipe. To derive the searching direction technique, the flow is separated into tree and chord branch components. Using these components, the objective function can be expressed in terms of tree branch flow alone and chord branch flow alone. Using standard calculus, the authors demonstrate the objective function can be decreased by swapping the status of a tree/chord branch pair. This derivation also provides a metric that determines the best pair to swap. Figure 2.6 shows an example of how a pair of tree branches can be swapped, where the solid lines represent tree branches, and the dashed lines represented chord branches. In this figure, the status of branches A and B are swapped.

Using both DDDP and the searching direction method, an algorithm to minimize a sewer system plan is proposed. To initialize the algorithm, an initial spanning tree is found using Dijkstra's algorithm, where edge weights are the product of pipe length and average ground elevation. From this initial layout, the iteration begins. Each iteration first swaps a tree/chord branch pair using the searching direction method, and minimizes the design of the resulting network using DDDP. The

[5]DDDP is a dynamic programming approach designed to decrease the memory and computational costs of standard dynamic programming. It is an iterative technique that uses a trial solution for each iteration. At the beginning of each iteration, an interval is defined around the trial solution. During the iteration, only state changes within this interval are considered. Conventional dynamic programming is applied to this restricted problem space to find a new solution, which is used as the trial solution during the next iteration. The process continues until a pre-defined ending criteria is met.

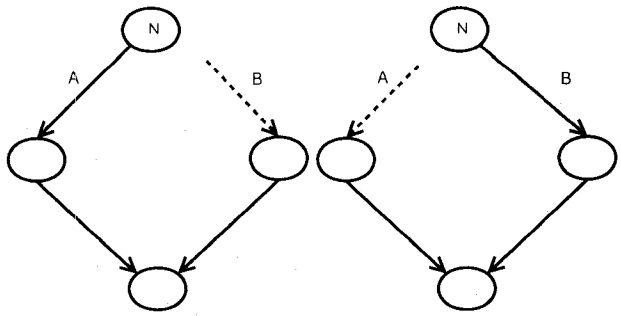

Figure 2.6: Swapping the status of branches A and B

swapping of branches continues until a stopping condition is reached, such as a lack of decrease in the objective function for n successive branch swaps.

The authors validate the performance of this method by comparing results to the manually produced plan for a real neighborhood of 10,000 people, with approximately 2.6 km^2 drainage area. The sewer plan produced shows a 14.79% savings for overall cost, and a 12.57% savings for construction costs alone.

Diogo *et al.* provide a framework for *three-dimensional* optimization of urban drainage systems[12]. This framework considers many facets of the sewer system planning stage, integrating modules that optimize the connectivity of nodes, pipe slope and diameter, manhole placement and position of special structures such as pumping stations and water treatment plants. It is claimed that this framework is applicable in planning for new sewer systems and for remodeling or expanding existing sewer systems. Three-dimensional optimization is defined by the authors as the "*simultaneous selection of the least-cost solution for the system (plan) layout and network (vertical) design subject to hydraulic, sanitary, and constructive constraints and to the restrictions of real applications.*"

Using this framework, the optimization process is divided into three main modules. The first module is concerned with minimizing the cost of the layout with respect to the connectivity of nodes. Similar to Tekeli, and Li and Matthews [20, 33], the authors seek a minimum cost spanning tree. However, the edge selection algorithms presented here are capable of generating several independent sewer networks, each with their own outfall. This is accomplished by generating multiple spanning trees. Given a fixed layout, the second module is used to determine which nodes should have fixed position manholes, pumping stations or outlets. Finally, the third module optimizes sewer sections, pipe levels and the number of intermediate manholes.

Two different approaches have been applied to optimize edge selection. The first is a simulated annealing approach [26]. To apply simulated annealing, it is necessary to define what constitutes neighboring trees, and a process for generating them. A technique similar to the tree/chord pair branch swapping of Li and Matthew [20] discussed previously is used. First, a node in the sewer system is selected at random. At this node, one of the downstream tree branches adjacent to the node is selected, and changed to a chord branch. To produce a valid tree, one of two things can be done.

The first is to choose a downstream adjacent chord branch, and reclassify it as a tree branch, thus reconnecting the tree. This is equivalent to the swapping procedure shown in Figure 2.6. Otherwise, if the node selected is classified as a valid outfall, then the sewage flow can be directed to this outfall instead of further downstream. Note that this second option means that an additional tree will be added to the forest. It is also possible to eliminate a tree if swapping the tree/chord branch pair connects the existing tree to another tree.

Genetic algorithms were also applied as an approach to optimize the layout. The authors develop a crossover operator that builds the children in an iterative fashion, where each iteration corresponds to a node in the network. Children inherit from each parent in alternating iterations. That is, child one inherits from parent one during odd iterations and from parent two during even iterations, and vice versa for child two. The process starts by generating a random permutation of the nodes. During the i^{th} iteration, the child inherits the downstream path from the i^{th} node in the permutation to either the outfall, or until the downstream path becomes connected to the outfall. Mutation is implemented by randomly applying the neighboring tree operator defined for the simulated annealing approach.

The design process involves two coordinated sub-models, one for the optimization of tree properties as a whole and another that optimizes properties of the individual edges. Tree optimization is analogous to the design optimization previously discussed. For this sub-model, pipes are buried at the minimum depth, and pipe diameter is optimized. In addition, the locations of pumping stations and treatment plants are specified. Edge optimization is primarily concerned with the placement of intermediate manholes. For this task, a grid of possible locations is defined for each new manhole. A dynamic programming technique, such as the Walters technique previously discussed [37], is used to calculate the position of these intermediate manholes.

Diogo and Graveto further examine edge selection algorithms for the three dimensional optimization architecture [11]. In this paper, a complete forest enumeration algorithm is developed and compared to the simulated annealing approach applied in [12]. This enumeration approach can be described as follows. Consider the graph defining the potential interconnections between nodes. The authors define four properties, shown in Figure 2.7, defining the connectivity and flow direction of each edge. In this figure, the left hand states represent a pipeline which is part of a forest, while the right hand states show pipelines that have cuts. As a pre-processing step, the legal states for each edge and the collection of outfall points are defined.

All of the forests for the graph are enumerated using a depth-first search approach. At each node in the search tree a state, chosen from Figure 2.7, is assigned to one of the edges. If this assignment produces an invalid forest, this search path produces a cutoff. This occurs when cycles are introduced into the graph or the resulting partial assignment does not admit a path from each node to an outfall. The simulated annealing approach discussed in this paper uses the neighborhood edge swap function described in the previously discussed Diogo paper [12].

The conclusion of this study is that full enumeration may be feasible for small to medium

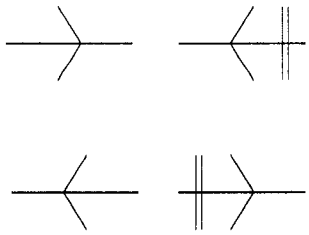

Figure 2.7: Four possible edge states

projects, but it is too computationally intensive for large ones. The authors do not present a size range for small, medium or large projects; however, results are presented for a real sewer system edge selection project.

The performance of these methods is evaluated using the plan for a real sanitary interceptor sewer in Coimbra City, Portugal. The initial base graph for the layout consists of 23 vertices, 35 arcs and 13 simple circuits. The total number of forests enumerated is 3,413,650. For each of these forests, a design was found using the design solver discussed in [12]. Producing a complete plan for each of these forests, including both layout and design, took about 97,277 seconds on a Pentium II system. The cheapest plan was about $3,620,016 Euros, while the most expensive cost was $10,839,242 Euros. The average cost over each forest was $7,794,650 Euros. Results for the simulated annealing were also promising. Over 20 series of 100 consecutive runs, the optimal layout was found approximately 69% times. Each of these runs required about two minutes computation time. These results imply that the simulated annealing approach will find high quality, or near optimal solutions while evaluating only a small portion of the search space.

2.2.4 Multi-Objective Sewer System Optimization

Sometimes, minimizing the infrastructure cost is not the only planning consideration. There are often other considerations such as water quality and network reliability. For example, several researchers have attempted to find low cost pipe networks that still provide a good degree of reliability. Todini presents a method for water distribution networks[34]. The concept of resiliency is defined in terms of power per node, or the pressure defined at the head of each pipe. The goal of a resilient network is to provide each node more power than necessary and to add redundancy in the pipe connections. The author presents several resilience indices based on this idea.

Using one of these resiliency measures, a set of *best solutions* is generated as follows. Since this optimization problem has multiple objectives, there is no single clear best solution. Therefore, as an input to this problem, an engineer providers an acceptable range of values for the resiliency index i. Consider a fixed resilience index $i = n$. For this value of i, the method returns the lowest cost solution with value $i = n$. In this manner, a set of best solutions is generated for each value of i in the specified range. The result is a *Pareto front* of solutions, where each solution represents the lowest cost solution with respect to its corresponding i value. A heuristic algorithm is used to

estimate the Pareto front. Using this front, an engineer can choose a solution deemed to be the best trade-off between network reliability and cost.

Prasad and Park present an approach to minimize the design cost, while maximizing network resiliency[25]. The network resiliency index is designed to incorporate both excess power delivered to each node and the degree of redundancy in the design. Multi-objective optimization is performed using non-dominated sorting genetic algorithms (NSGA). NSGAs are similar to the traditional GAs, with the exception that the selection operator is designed to prefer non-dominated candidate solutions. A candidate solution x_i dominates another candidate x_j if both:

1. The candidate x_i is no worse than x_j in all objectives, and
2. The candidate x_i is strictly better than x_j in at least one objective.

Each non-dominated solution is given the same large dummy fitness value. In order to maintain diversity in the population, each non-dominated solution is divided by a niche count, which is proportional to the number of solutions close to this candidate.

As in previous GA formulations, a hydraulic solver is used to verify flow constraints. For all other constraints, a failure index defines how much a candidate solution breaks the constraints. To allow the evolution process to produce better solutions, infeasible candidates may propagate from generation to generation. To allow the propagation of infeasible solutions, the definition for dominance is extended as follows. A candidate x_i dominates another candidate x_j if any of the following are true.

1. Both are feasible and x_i dominates x_j with respect to (1) and (2) above
2. Solution x_i is feasible and x_j is not,
3. Both are infeasible, but x_i has a smaller degree of violation, or
4. Both are infeasible, and x_i dominates x_j with respect to (1) and (2) above

Tests for this technique show promising results. The Pareto Front for a relatively small network was solved by using exhaustive search. The NSGA algorithm produced a solution set very close to the actual Pareto Front. It is worth noting that the quality of a solution is dependent on the network resiliency index. If this index does not accurately compare the reliability of different solutions, the resultant network design may produce poor reliability.

Afshar *et al.* present a two phase iterative algorithm to optimize both design and edge selection of water distribution networks, while maintaining a certain degree of reliability[6]. To measure reliability, the number of paths from the source to each node is used as a metric. A network with reliability of degree n would have at least n different paths from the source to any node.

The first phase of this algorithm deals with design optimization. The problem is formulated as a cost minimization problem where the design cost is minimized with respect to head pressure,

velocity and diameter constraints. Head and velocity constraints are enforced using cost penalties. Each network is evaluated using a hydraulic solver. If the network violates either head or velocity constraints, a penalty term proportional to the degree of violation is added to the cost. During this stage, a continuous range of pipe diameters is used. Based on this formulation, a design is found using the DOT optimization package[4].

The second phase is known as the floating phase. During this phase, pipes that may be removed from the network without violating reliability are identified. A measure, known as the floating index, is used to determine which pipe should be removed from the network. This measure is calculated as $F_i = \frac{C_i}{H_i}$, where C_i is the cost of the pipe, and H_i measures the hydraulic importance of the pipe to the network. Three different values for H_i are considered. These are pipe diameter, discharge of the pipe, and the contribution of the pipe to the head demand at its downstream node. During the floating phase, the pipe with the highest floating index is marked as a floating pipe. The minimum diameter for the pipe is set to zero, and the design phase of subsequent iterations may choose to remove this pipe from the network. This two phase process continues to iterate until it is not possible to label pipes as floating.

The solution produced by the preceding method produces continuous pipe sizes. To be useful in a real sewer system plan, the pipes specified must be comprised of commercially available pipe sizes. To obtain this, a simple search is applied. First, each pipe in the network is sized down to the nearest comparable pipe. This results in a network that violates the head and velocity constraints. Then, a search is conducted to determine which pipe in the network would provide the least increase in cost if the pipe size was rounded up. Heuristics to find this pipe are defined based on the type of constraint defined, i.e. velocity or head constraints. Searching continues until the resulting network does not violate any of the constraints.

The edge selection techniques previously discussed involved choosing a set of edges ensuring that each node is connected to the layout. In some cases, determining which nodes a network should service is also part of the problem. Lejano gives an example for water distribution networks[19]. In these networks, a certain amount of water will be returned to the potable water supply. Instead of supplying each node with fresh water from an outside source, reclaimed water may be redistributed throughout the system. However, delivering reclaimed water can be expensive. Given a set of clients, it may be desirable to choose a subset of these clients such that the benefits from distributing this reclaimed water justifies the cost.

In the previously cited work, Lejano adopts a mixed integer linear programming approach with an objective function that simultaneously maximizes the benefits of distributing reclaimed water and minimizes the cost of the system. In general terms, the problem solves for the set of clients supplied, the edges connecting these clients and the flow at each node. This formulation includes constraints to validate some simplifying assumptions, as well as preserving the integrity of the flow throughout each node in the network.

2.3 Integrating Manhole Placement into Sewer System Planning

As illustrated in the previous sections, a considerable amount of research effort has gone into the automation of planning sewer systems. Results clearly illustrate the potential for savings in both man-hours and infrastructure costs. Many authors report a reduction in pipe, excavation and/or maintenance costs.

Developing algorithms that simultaneously optimize all the considerations for a sewer plan is regarded as a difficult problem. Rather, it is a widely adopted simplification to break the sewer planning process into the two interrelated tasks of design and layout. Of these two tasks, the layout is regarded as more difficult.

Much of the existing research considers layout as only the connectivity of nodes, or edge selection. Each of these techniques assumes fixed nodes, and the possible interconnections to be defined beforehand. There are several problems with this approach. First, it provides only a semi-automation of the layout task. Manual effort is still required to produce this information. In addition, the quality of the solution depends on fixed nodes. If a little flexibility is allowed in the position of fixed position manholes, it might be possible to produce a better pipeline containing fewer manholes.

Most researchers disregard the placement of manholes. However, a clever manhole placement scheme may result in pipelines with fewer manholes, further reducing the cost of the solution. The reduction of even a few manholes in a sewer system plan may save tens or hundreds of thousands of dollars on infrastructure costs. Few authors have addressed this consideration. As discussed in Section 2.2.4, Walters [37] presents a method for doing this. However, this method requires an initial layout to be provided, and a reduction of manholes may only be achieved if two manholes can be merged into one. Diogo *et al.* [12] introduce a system architecture that considers this level of optimization, but provide little detail, stating a technique such as that of Walters [37] may be used for this. This thesis research develops manhole placement techniques. These techniques can be integrated with other layout and design algorithms to further automate the sewer planning process.

Many authors propose planning systems following the general architecture shown in Figure 2.8. These systems typically use algorithms similar to those previously discussed to optimize edge selection and design. Many such systems optimize edge selection and design in an iterative fashion. Tools to further optimize sewer plans by considering the placement of manholes are integrated into this process. An architecture for such a system is presented in Figure 2.9.

During the pre-processing phase, information is extracted from engineering survey data, and any data required for optimization algorithms is prepared. This includes the determination of fixed points and their connectivity. This thesis shows how this can be done for residential neighborhoods by identifying intersections and placing fixed points at these intersections. The main difference in this architecture is the manhole placement phase. The purpose of this phase is to define the position of intermediate manholes between fixed points. To further optimize the sewer network, the position of fixed points may be varied to facilitate the elimination of manholes. Note that this architecture

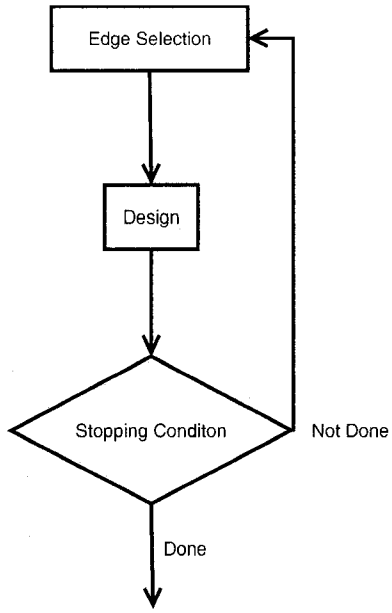

Figure 2.8: Typical sewer planner architecture

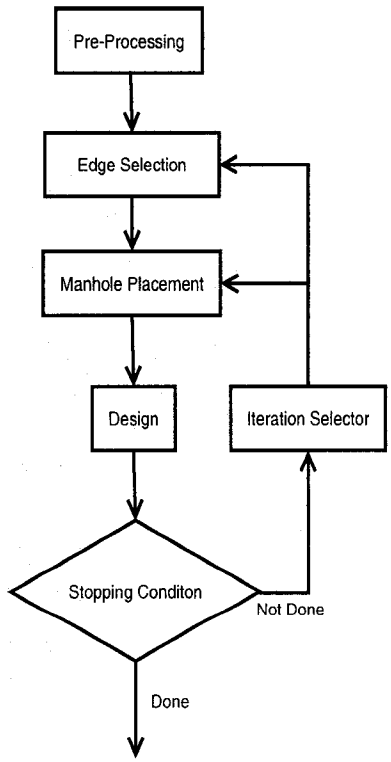

Figure 2.9: Proposed sewer planner architecture

allows iteration from the design phase to both edge selection and manhole placement. The iteration selector algorithm takes the current state of the solution and determines whether to iterate back to the edge selection or manhole placement phase. Each level of optimization may produce several alternatives to be considered for the final plan. An open research problem is to determine how these iterations will be performed.

2.4 Conclusion

This chapter provides an overview of the engineering process. For the purpose of automation, the sewer planning problem is separated into three sub-problems. Edge selection defines the high-level connectivity between fixed points in the system. Manhole placement defines the position of the pipes and manholes that compose the pipeline between fixed points. Finally, design specifies the diameter and upstream and downstream heights of each pipe.

An overview of the literature related to these three sub-problems is presented. This literature is rich in ideas, but most of these ideas are not validated on a wide range of real neighborhood data. Despite some promising results, there is no fully automated planning system widely used by industry. Many techniques presented in the literature do not attempt to minimize cost by considering the placement of each pipe and manhole. Considering this level of detail in the automation process can reduce both manual effort and infrastructure costs. An expanded sewer planner architecture has been presented, incorporating a more detailed manhole placement phase.

The following chapters will introduce tools that can be used within this architecture. Chapter 3 explores the feasibility of dealing directly with real engineering data. Chapter 4 develops algorithms to place a pipeline between two fixed points. Chapter 5 builds on this work developing several algorithms to place the pipes for an entire neighborhood, attempting to reduce the total number of manholes. Chapter 6 presents an initial investigation into how the algorithms developed in Chapters 4 and 5 can be integrated into a complete planning system.

Chapter 3

Representing Neighborhoods using Multigraphs

This research examines the automation of the sewer system planning process. Ideally, an automated planning system would use data in the same format already used by engineers. Otherwise, additional effort would be required to use such a system. In this chapter, strategies for imposing structure on neighborhood rendering data are examined. In particular, a road and intersection multigraph structure used by many algorithms in this thesis is introduced. Not only does this chapter introduce basic tools for this research system, it also provides an insight into data manipulation considerations for an industrial strength system.

3.1 Dealing with Raw Data

The proposed sewer system automation begins with the data compiled from a neighborhood survey. This data consists of lines and arcs representing the curbs that delimit roads in the neighborhood. For engineers, this data is sufficient for the two-dimensional road map to be visualized, and can be used in cooperation with CAD software to allow an engineer to manually place pipes and manholes. For this thesis, lines and arcs are taken to be the geometric primitives from which a more complex neighborhood multigraph structure is built. It is assumed that these geometric primitives are available as input to the automation process. Theses lines and arcs are represented in an input file, using the format shown in Figure 3.1. Lines are defined by the three-dimensional coordinates of their endpoints. Arcs represent a subset of a circle, and are defined by the center point, a radius and two sweep angles. The arc is the portion of the circle swept from the first angle to the second angle in the clockwise direction. These angles are measured in radians.

Note that each entry begins with a two letter prefix specifying what the geometry composes. Currently, the only prefix in use is CR, specifying that this geometry is a piece of the road curbs. This data field has been reserved to allow additional geometry, such as the pipelines in the sewer system layout, or pipes or wires from other utility services, to be represented in the same file.

CR Line x_1 y_1 z_1 x_2 y_2 z_2
CR Arc x y z radius $angle_1$ $angle_2$

Figure 3.1: Data format for line and arc geometric primitives respectively

The algorithms presented in this thesis require that additional structure be imposed on this data. Geometry is grouped into objects classified as either roads or intersections. The following section describes the process of identifying these components and organizing them into a multigraph composed of roads and intersections.

3.2 Inferring the Multigraph Structure

Extracting components from the initial data begins by identifying intersections. These are identified by matching portions of the geometry with intersection templates. The test system for this thesis defines templates for three-way and four-way intersections as well as cul-de-sacs. These templates are shown in Figure 3.2. Once the intersections have been isolated, the remaining geometry is separated into road segment components. To accomplish this, the geometry is organized into groups of connected chains. These chains are paired such that each pair represents the curbs for one road segment. The entrances of each road segment are identified as being either intersection adjacent or a dead-end. Each intersection mouth must be adjacent to a road segment. The structure defined above can be represented as a multigraph where the intersections are the vertices, and the roads the edges. This structure is a multigraph, and not a simple graph, as it is possible to have more than one edge between a pair of vertices. The terms neighborhood graph and neighborhood multigraph will be used interchangeably in this thesis.

Figure 3.2: Patterns for cul-de-sacs, three-way and four-way intersections

Consider the sample neighborhood containing five labeled intersections as shown in Figure 3.3. The neighborhood graph for this two-dimensional map is shown in Figure 3.4, where the intersections are represented as vertices and the edges as roads. Note, that a single pair of intersections may have multiple edges between them, as is the case for the intersection pairs (3,5) and (1,4). Also note the dead-end road adjacent to intersection 2. Edges such as this are often excluded from the graph since they only have once adjacency. In this case, the layout algorithms will operate on the graph without this edge. Layouts for these dead-end edges are determined separately and appended to the layout generated on the neighborhood graph.

To properly infer a neighborhood structure from the raw data, the geometry composing each

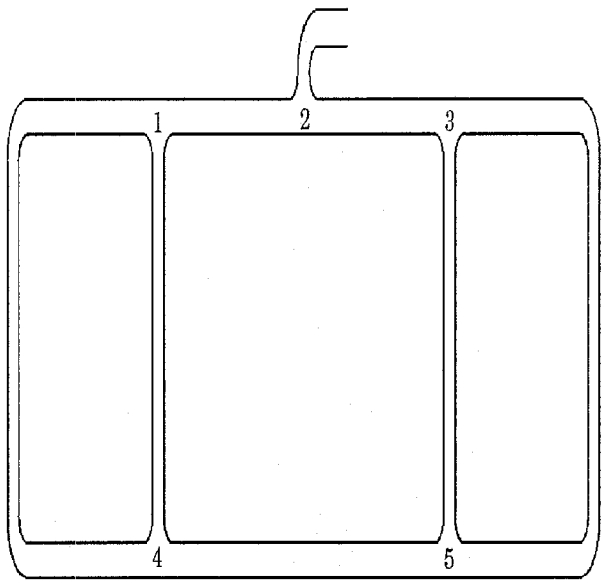

Figure 3.3: Sample neighborhood

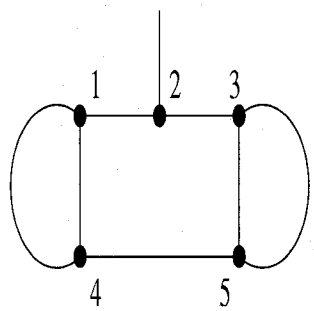

Figure 3.4: Neighborhood graph structure inferred from the neighborhood in Figure 3.3

intersection must match an intersection template. While the current research system can successfully impose this graph structure on many real data files, it is likely that the templates shown in Figure 3.2 are not representative of all intersections occurring in real data. For example, a neighborhood may contain five-way intersections.

As well, real data files may contain noisy data, complicating the pattern matching process. In fact, several obstacles had to be overcome to accurately parse the test neighborhoods used in this thesis. For example, there are often gaps between the lines and arcs defining a single curb. Theoretically, the endpoints for a line or arc that is part of a curb should be shared by adjacent geometry. In practice, the endpoint of one segment does not always correspond directly with its adjacent component. This has little affect when rendering the road, but such a gap can be problematic for computation. To generate the multigraph, these gaps must be identified and filled. In addition, there were also instances of duplicate geometry. These duplicates must be identified and removed to avoid errors in computation.

Such real world flaws are an implementation issue. Producing a robust system for converting real data into the neighborhood graph is beyond the scope of this thesis. This work has been used to implement a data conversion system adequate for this thesis, and to provide a proof of the concept that an automated sewer system planner can use existing rendering data. The use of this type of data is important, because it relieves the engineer of preparing new data sets for the automation process.

3.3 Cut Location and the Expanded Graph Structure

The previous section describes how the lines and arcs used to display the neighborhood can be organized into an intuitive multigraph structure. However, this multigraph does not specify all of the properties of the layout. Consider applying edge selection techniques to the neighborhood graph. This results in a spanning tree where the edges define the core backbone of the sewer network. Edges from the neighborhood graph not appearing in the spanning tree represent chord branches. Each chord branch requires a cut in its corresponding pipeline, or else there will be a cycle in the layout. This cut may lie anywhere along the pipeline. However, each potential cut location defines another two-dimensional layout, which in turn defines new flows in each graph. Consider a neighborhood where there are t trees, and n chord branches. If there are m candidate cut locations per chord branch, the number of layouts will be $t \times m^n$, as there will be m^n different combinations of cut locations amongst the chord branches.

Therefore, to reduce the size of the layout space, cuts are constrained to be located at one of the two extremes of the chord branch. Using this simplification, an expanded graph structure may be used to represent the exact location of each cut. Consider two adjacent vertices, v_1 and v_2, connected by the edge $e = \overline{v_1 v_2}$. If e is not a tree branch in the layout, it will be completely absent from the spanning tree generated during edge selection. However, this absence tells us nothing about the position of the cut within the pipeline. Based on the earlier assumption, there are two potential

locations for this cut. To represent this for an edge e, a dummy vertex $n_{1,2}$ is introduced. This edge e is then replaced by two new edges $e' = \overline{v_1 n_{1,2}}$ and $e'' = \overline{v_2 n_{1,2}}$. This expanded graph can be used to represent whether the edge e is a tree branch, and if not which end the cut will lie. In any spanning tree, there are three possibilities for the expanded edges: both e' and e'' are in the tree, only e' is in the tree or only e'' is in tree. If both edges are in the graph, then the original edge e will be a tree branch in the sewer network. The case where only e' is in the tree is used to represent the case where the cut is on the v_2 side. Similarly, e'' represents the case where the cut is on the v_1 side.

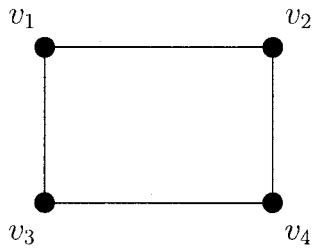

Figure 3.5: Simple neighborhood multigraph for four road loop

The above expansion is performed for each edge in the original neighborhood graph. Applying this expansion technique to Figure 3.5 produces the expanded graph shown in Figure 3.6. A complete edge selection of this graph is represented in Figure 3.7. This graph represents a layout with only one chord branch $\overline{v_2 v_4}$. The cut in this chord branch is located at vertex v_2, which is specified by the absence of $\overline{v_2 n_{2,4}}$ in the spanning tree from the expanded graph shown in Figure 3.7.

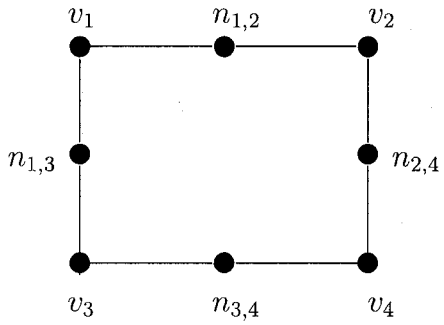

Figure 3.6: Expanded neighborhood graph for four road loop

The expanded graph structure introduced here provides a powerful representation for edge selection algorithms. This representation has several convenient consequences. The expanded graph structure facilitates a cleaner description of algorithms. With the simple neighborhood graph, many algorithms need to be defined as a two step process. In the first step, the core backbone for the sewer network is calculated by finding a spanning tree in the graph. However, the resulting graph does not specify the position of cuts, only the edges where they exist. Thus, a second step to define the location of cuts in these cut edges is required. In the expanded graph, each valid layout maps to a spanning tree and vice versa. Edge selection can be represented as a single step algorithm generating

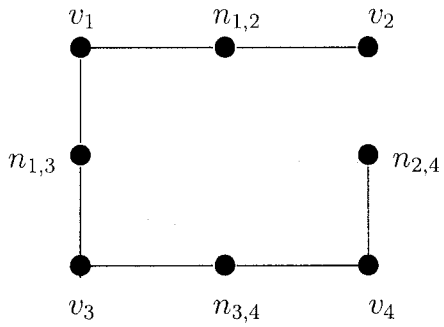

Figure 3.7: Spanning tree in expanded graph representing a complete edge selection

spanning trees in the expanded graph. Thus, the expanded graph provides a more complete layout representation and a more convenient way to express edge selection algorithms. It is worth noting that the expanded graph will be larger than the original neighborhood graph. In the expanded graph, the number of vertices increases by the number of edges in original graph. As well, the number of edges in the expanded graph is twice the number of edges in the original graph.

This expanded graph will utilized by the sewer planning prototype presented in Chapter 6.

3.4 Conclusion

This chapter discussed how data used to display a neighborhood can be converted into a form used to compute the layout. Several basic templates that can be used to identify intersections are introduced. Once intersections have been identified and extracted, the remaining geometry is separated into the roads connecting these intersections.

This chapter also discusses the position of cuts within chord branches. The location of cuts is constrained to the extremities of each road to limit the number of sewer system layouts. Based on the assumption that the cuts will lie at the extremities of each road, an expanded graph structure is introduced. This graph structure efficiently represents the tree and chord branches in a sewer network as well as the position of the cuts within each chord branch. The expanded graph structure contributes a simple structure that represents direction and cut information for primary and secondary pipelines of a sewer system. This structure can be adopted for existing and new edge selection algorithms.

Chapter 4

Local Placement

The automation and optimization of real-world sewer systems involves many variables, such as manhole and pipe positions, and the depths, slopes and diameters of pipes. The approach adopted in this thesis separates the planning process into easier to solve sub-problems. Each of these sub-problems are solved separately, and the results combined to form a complete sewer plan. One important sub-problem is the positioning of pipes and manholes within a single pipeline. This chapter explores techniques to generate these pipelines by placing pipes and manholes between two fixed points.

4.1 Problem Definition

Consider the problem of connecting two fixed points with a pipeline. In the sewer planning problem, pipelines are constrained to lie beneath roads. For this problem, roads are represented as a simple polygon. The term simple polygon is used to denote a polygon with a well-defined interior and exterior. To ensure the pipeline is placed underneath the road, the two initial points, and each point in the pipeline, are required to lie totally within this polygon. In addition, manholes are required whenever a pipe either reaches a maximum distance, or changes directions. Therefore, even if the two points are visible via a straight line, it may not be possible to connect them using a single pipe.

There are two aspects of this sub-problem that affect the cost. These are pipe length and the number of manholes. The variance in pipe length between two layouts is relatively small. This fact, coupled with the high cost of installing manholes, makes the number of manholes the cost-dominating factor. Therefore, the performance of placement algorithms is evaluated by the number of manholes in the pipeline.

Several placement approaches are examined in this chapter. First, a suite of techniques using control points to guide the placement is presented. The first technique uses control points in the center of the road to guide the placement. To improve upon this center-based approach, two solutions using a heuristic that guides the pipeline toward curbs are explored. The final approach iteratively expands frontiers of reachable points.

4.2 Control Point Placement

The first placement techniques explored guide placement using a set of control points. Any point lying inside of the road polygon may be used as a control point. Pipes are placed one at a time, in the direction of the furthest reachable control point. This placement scheme is greedy in that the longest pipe length possible is always used. The basic algorithm can be defined as the following two step process:

1. If the goal manhole can be reached using an allowable pipe length, connect the current manhole to the goal, otherwise
2. Identify the furthest control point that is reachable via a straight line lying totally within the road polygon and, using this line as a direction vector, place a pipe of maximum allowable length.

PLACEPIPE($currentmanhole, p_2, maxlength, layout$)

/*
INPUT VARIABLES:
p_2: Control point guiding search along vector $\overline{currentmanhole, p_2}$
maxlength: The maximum allowable pipe length

INPUT/OUTPUT VARIABLES:
currentmanhole: Last placed manhole, used as starting point for the pipe
layout: List of manholes defining a pipeline
*/

if DISTANCE($currentmanhole, p_2$) $> maxlength$
then
$norm \leftarrow$ NORMALIZEDDIRECTIONVECTOR($currentmanhole, p_2$)
$endmanhole \leftarrow currentmanhole + maxlength \times norm$

else
$endmanhole \leftarrow p2$

ADDMANHOLE($endmanhole, layout$)
ADDPIPE($currentmanhole, endmanhole, layout$)
return ($endmanhole$)

Figure 4.1: Pseudo-code for function placing the largest possible pipe starting at the point *currentmanhole*

```
CONTROLPOINTPLACEMENT(road, p1, p2, maxlength)

/*
INPUT VARIABLES:
road: Polygon delimiting the area of the road
p1, p2: Beginning and ending points for the pipeline
maxlength: The maximum allowable pipe length

RETURNS:
layout: List of manholes that defines the pipeline between p1 and p2
*/

layout ← ∅
controlpoints ← FINDCONTROLPOINTS(road)
currentmanhole ← p1
while currentmanhole ≠ p2
  do
  { if REACHABLE(currentmanhole, p2)
      then
    currentmanhole ← PLACEPIPE(currentmanhole, p2)

      else
    { fp ← FURTHESTREACHABLEPOINT(currentmanhole, controlpoints)
      currentmanhole ← PLACEPIPE(currentmanhole, fp)
return (layout)
```

Figure 4.2: Pseudo-code for function to place a pipeline between two points where the placement is guided by control points

Consider the placement of a single pipe. For a moment, assume the control point guiding this placement has already been identified. PLACEPIPE, with pseudo-code shown in Figure 4.1, describes the details of how the pipe is placed. Placement begins from the last manhole placed, represented by *currentmanhole* in the pseudo-code. The direction of this pipe will be defined by the vector from *currentmanhole* to p_2. The resulting pipe will meet one of two criteria. Either the pipe will be of the maximum allowable length, or it will halt on the boundary of the road so as to stay within the polygon. The fact that each control point lies within the road polygon ensures pipes are not placed outside of the road. PLACEPIPE ensures this is the case by not allowing the placed pipe to be longer than the vector $\overline{currentmanhole, p_2}$.

Using the function PLACEPIPE, the pseudo-code for placing the pipeline is defined in CONTROLPOINTPLACEMENT, shown in Figure 4.2. During each iteration, the algorithm checks whether the goal point, p_2, is reachable from *currentmanhole*. If so, the algorithm proceeds to place a straight pipeline until the goal point is reached. If not, then the furthest reachable control point is identified and pipe placement is conducted using the function PLACEPIPE. Note that this function assumes the existence of a reachable control point. If no reachable point exists, a pipe will not be placed. Each method used to generate control points has been developed to ensure that at least one control point is reachable from anywhere in the road.

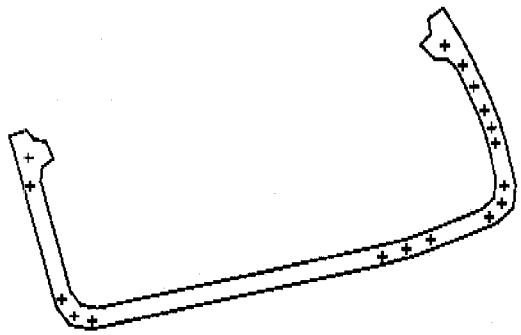

Figure 4.3: Control points for centerline placement

The function FINDCONTROLPOINTS is key in determining the form of the pipeline produced by CONTROLPOINTPLACEMENT. It is this function that determines the control points guiding the search. Two different approaches are used to identify control points. The first is referred to as centerline, and is intended to guide the search toward the center of the road. Control point selection begins by arbitrarily choosing one of the two curbs. Control points are defined at the center of the road at each place where the chosen curb changes direction. Note that this process could have been done for both curbs. However, for most roads the two curbs are very similar in structure, and thus defining control points using two curbs generate pairs of very close points. Figure 4.3 shows an example road component, with the control points represented by crosses. The second approach is

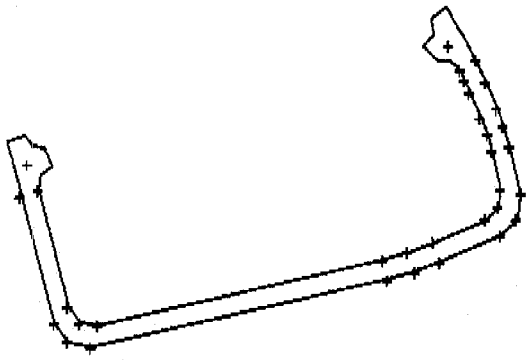

Figure 4.4: Control points for curbpoint placement

referred to as the curbpoint approach, and is used to guide the pipeline in such a way that is likely to cut corners. For each curb, the points where the curb changes direction define the control points. An example of these control points are shown in Figure 4.4.

When using curb points to guide placement, the aforementioned algorithm only selects a single point from two curbs. However, there is no guarantee that the chosen curb point will generate the best solution. To improve the chances of generating a better solution, this algorithm is modified to consider the furthest control point from each of the two curbs. Conceptually, this algorithm builds a binary tree, as each step in the placement expands the solution using two different manholes. The leaves of the tree correspond to complete layouts, and a node at depth n corresponds to the placement of the n^{th} manhole in the pipeline.

The pseudo-code for this approach is presented in Figures 4.5 and 4.6. To differentiate between the convention and doubling approaches, the two techniques will be referred to as curbpoint single and curbpoint doubling placement respectively.

4.3 Frontier Placement

The frontier placement algorithm returns a close approximation to the minimum number of manholes required to place the pipeline for a road segment. This approach is based on the idea of propagating a frontier of reachable points. A frontier can be defined in terms of an initial set of points and a maximum pipe length. The frontier for a set of points is a curve defining the furthest reachable points in all directions from the set of points. The idea of this algorithm is to propagate reachability frontiers from the initial point until the goal point can be reached. In continuous space, the number of frontiers required to reach the goal point defines the minimum number of manholes required to place a pipeline from the initial point to the end point. A proof sketch for this claim is presented in Appendix A.

Consider the initial manhole as the first frontier, the i^{th} frontier represents the furthest reachable points using $i - 1$ pipes. Figure 4.7 shows a screen shot of the frontiers generated during an actual

```
GREEDYDOUBLING(road, p1, p2, maxlength, layout)

/*
INPUT VARIABLES:
road: Polygon delimiting the area of the road
p1, p2: Beginning and ending points for the pipeline
maxlength: The maximum allowable pipe length

RETURNS:
Manhole layout from GreedyRecur function call
*/

// Takes the polygon representing the road, and returns geometry for each
// curb separately
GETCURBS(curb1, curb2, road)
controlpoints1 ← FINDCONTROLPOINTS(curb1)
controlpoints2 ← FINDCONTROLPOINTS(curb2)
currentmanhole ← p1
layout ← ∅
return (GREEDYRECUR(p1, p2, controlpoints1, controlpoints2, maxlength))
```

Figure 4.5: Pseudo-code for main greedy doubling function

placement. The initial manhole is the rightmost cross, with the leftmost cross being the target manhole. Each cluster of crosses is the discrete approximation of the frontier found by applying frontier placement. Note that the target manhole is considered as the fifth and final frontier. To find a specific placement, a backward trace is conducted from the destination manhole, through each frontier back toward the source manhole. In choosing the manholes on adjacent frontiers, care must be taken to ensure no pipes longer than the allowable length are placed. It is a straightforward process to trace a single valid layout from the frontiers. It is worth noting that pipe length can be minimized by searching through the space of different manhole pairings between frontiers. However, the computational effort required for this search is often expensive and provides little in the way of cost savings to the overall layout.

4.4 Local Placement Tests

The performance of these local placement algorithms has been tested using several synthetic and real roadway corridors. These roads have been chosen to exhibit different characteristics. Test roads range from straight curbs, to gentle bends to very windy roads. The synthetic roads were generated using a tiling approach, where each tile is either a straight road piece or a bend.

Layouts were produced for each test road using the centerline, curbpoint single, curbpoint doubling and frontier approaches. In addition, the maximum pipe length was varied, providing additional data for analysis. The performance of each algorithm is evaluated by measuring the number

```
GREEDYRECUR(p1, p2, controlpoints1, controlpoints2, maxlength)

/*
INPUT VARIABLES:
road: Polygon delimiting the area of the road
p1, p2: Beginning and ending points for the pipeline
controlpoints1, controlpoints2: Set of control points from each curb respectively
maxlength: The maximum allowable pipe length

RETURNS:
layout: Least cost layout from lay1 and lay2
*/

GETCURBS(curb1, curb2, road)
if p1 = p2
  then return (layout)
if REACHABLE(currentmanhole, p2)
  then PLACEPIPE(currentmanhole, p2)

  else
  { f1 ← FURTHESTREACHABLEPOINT(currpoint, curb1)
    f2 ← FURTHESTREACHABLEPOINT(currpoint, curb2)

    lay1 ← lay2 ← layout
    f1 ← PLACEPIPE(currentmanhole, f1, lay1)
    f2 ← PLACEPIPE(currentmanhole, f2, lay2)

    lay1 ← GREEDYRECUR(f1, p2, road, controlpoints1, controlpoints2, lay1)
    lay2 ← GREEDYRECUR(f2, p2, road, controlpoints1, controlpoints2, lay2)
  }
return (MINIMUMLAYOUT(lay1, lay2))
```

Figure 4.6: Pseudo-code for recursive greedy doubling function

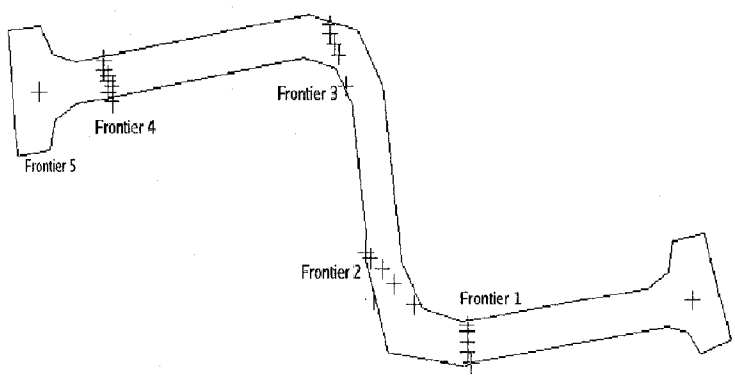

Figure 4.7: Example road segment with five frontiers

of manholes in each layout. The length of the pipeline is not considered, as the number of manholes is the cost dominating consideration for these pipelines.

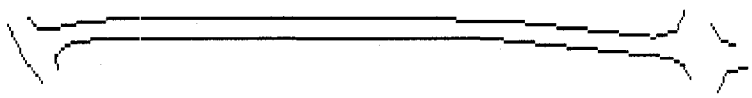

Figure 4.8: Real neighborhood straight stretch

Tests were conducted using a test suite of five different roads. The geometry for these roads is displayed in Appendix B, while the results of the tests are shown in Appendix C. For the test instances discussed in this section, the geometry from Appendix C has been reproduced within the chapter.

From the results of this test suite, the following observations can be made. Even for relatively straight road segments, a simple algorithm may not generate a placement with the fewest number of manholes. An example of a relatively straight road from a real neighborhood can be seen in Figure 4.8. For each of the pipe lengths used to generate a layout for this road, the two curbpoint approaches and the frontier approach generate the same number of manholes. However, for several instances the centerline generated a layout with one more manhole than layouts generated by the other three algorithms. These results indicate that for roads with only gentle bends, the choice of local placement algorithms can affect solution quality.

Pipe Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
40	8	8	8	8
50	7	7	7	7
60	6	6	6	6
70	6	5	5	5
80	5	5	5	5
90	5	4	4	4
100	5	4	4	4
110	5	4	4	4
120	4	4	4	4
130	4	4	4	4
140	4	4	4	4
150	4	4	4	4

Figure 4.9: Number of manholes in placements for Real Straight Road

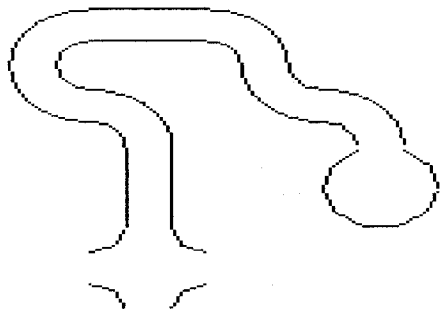

Figure 4.10: Test Road B

Pipe Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
40	12	12	11	11
50	10	10	10	10
60	10	10	9	9
70	8	10	9	7
80	7	8	7	7
90	7	6	6	6
100	7	7	6	6
110	7	7	6	6
120	7	7	6	6
130	7	7	6	6
140	7	7	6	6
150	7	7	6	6

Figure 4.11: Number of manholes in placements for Test Road B

For roads with curves, the curbpoint and frontier algorithms are better at discovering layouts with fewer manholes than the centerline approach. The layouts generated for Test Neighborhood B, shown in Figure 4.10, present a good illustration of the differing capabilities of each algorithm. Consider the number of manholes generated for each layout, as shown in Figure 4.11. For most of the pipe lengths in this test, the curbpoint doubling and frontier algorithms generate solutions with fewer manholes than centerline and curbpoint single. Cases where this happens are shown for the six problem instances in the range from 100 to 150.

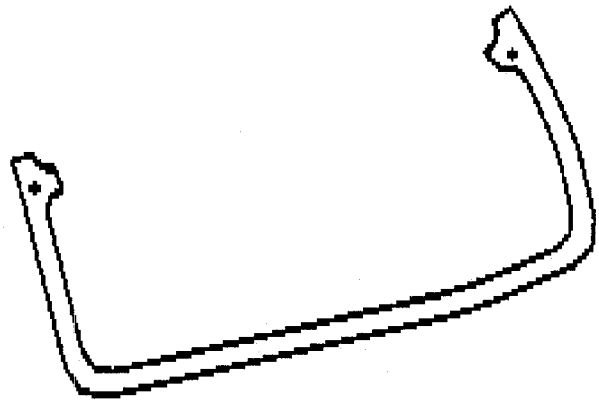

Figure 4.12: Test Road D

Another consideration for these algorithms is computation time. To provide an analysis of the time required for each algorithm, tests have been conducted on Road D, shown in Figure 4.12. Road D is a real road taken from a real neighborhood. This road was chosen for two reasons. First, the road is of realistic length. Second, this road contains two curves, making the placement more complex than a straight stretch of road. Maximum pipe lengths were varied from very small to about twice the expected pipe maximum to evaluate time for layouts with varying numbers of manholes. These results are presented in Figure 4.13. The computation time required for both the centerline and curbpoint single approaches is almost negligible. For the frontier approach, computation time is consistently in the 0.5-0.7 second range regardless of the maximum pipe length. This result is important, as it shows that the frontier approach scales well. The shorter the maximum pipe length, the larger the number of frontiers that will need to be calculated. Therefore, this result indicates that the additional frontiers required are calculated with a very low cost. The curbpoint doubling approach did not fare as well. For the shortest pipe length, this algorithm required 2,738 seconds to compute a layout containing 24 manholes.

Maximum Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
15	0.002	0.043	2,738.0	0.645
20	0.001	0.034	70.64	0.586
25	0.000	0.027	11.56	0.506
30	0.001	0.024	3.17	0.564
35	0.001	0.019	0.910	0.510
40	0.000	0.018	0.714	0.482
45	0.001	0.017	0.383	0.555
50	0.000	0.015	0.284	0.567
55	0.000	0.015	0.225	0.625
60	0.001	0.015	0.162	0.607

Figure 4.13: Layout computation times in seconds for Test Road D

The curbpoint approaches employ a heuristic to cut corners that often outperform the centerline

approach. However, the curbpoint approaches will not always outperform centerline. An example of this occurs at pipe length 70 for Neighborhood B, shown in Figure 4.11. Notice here that the centerline approach produces a layout with fewer manholes than both the curbpoint single and curbpoint doubling approach. However, in this case the frontier approach is able to produce a solution with fewer manholes than each of the other algorithms.

Remember that the frontier approach finds an approximation of the optimal layout. The quality of the solution depends on the sampling of the continuous space. In each of the tests shown here, the frontier technique generates layouts that are at least as good as the solutions generated using other algorithms. In some cases, the quality of the solution found by the frontier algorithm is better than the other solutions. While this does not indicate that the frontier approach is finding optimal solutions, it gives evidence that it is effective at finding good quality solutions.

In this section, performance for control point and frontier algorithms was measured against the performance of simpler algorithms. However, a more interesting metric is to compare the number of manholes with placements from real neighborhoods. Unfortunately, there are several reasons that this is not a fair comparison. One reason is that local placement assumes a single fixed maximum pipe distance. In the pipeline from the real neighborhood, this restriction does not exist. Furthermore, an engineer may sometimes use curved pipes to avoid placing manholes.

For local placement, the only data needed is the curb data. However, engineers consider additional factors, such as the elevation at different points in the road. Part of the reason these algorithms ignore elevation was the absence of this information in the testing data.

These different problem constraints make it difficult to compare solution quality with real layouts. However, this does not indicate that the algorithms presented here are not useful. Rather, these algorithms provide a set of basic tools to be used in the automation process. For an industrial strength planning system, they will have to be enhanced to meet the additional real world constraints.

4.5 Overview of Computational Complexity

In each of the tests presented in this chapter, road layouts were always generated in a relatively small amount of time. However, pipeline placement is a primitive tool for the entire sewer system planning problem. To predict how each of these algorithms will scale as the size of the roads grow, computational complexity will be examined.

Consider the process of placing pipe(s) from a fixed starting manhole. During each step, the algorithms presented here evaluate a set of candidate manholes, expanding a single or multiple layouts respectively. Determining the candidate manhole set and adding the pipe to the layout are not computationally intensive. The main computational consideration is determining whether the resulting pipe lies within the curbs of the road. This analysis will include a description of the computation entailed by this test, and estimate a bound on the number of these tests required for each of these algorithms.

Testing whether a pipeline lies within the curbs of the road is done primarily using line segment intersections. If the line segment does not intersect the road polygon, it can be said that the segment is either totally inside or outside of the polygon. To determine whether the line segment is totally inside the polygon, the midpoint of the line segment is tested to see if it is inside or outside the polygon. A single line segment intersection test is not expensive. However, the number of these tests depends on the line segments in the road. For a highly detailed road with many line segments, this test must be performed many times, adding to the overall computation required.

Consider the centerline approach, where the number of control points is p and the number of manholes in the layout produced is m. During each iteration, there is a maximum of p line segment tests, one for each potential control point. Therefore the number of line segment tests for this algorithm is on the order of $O(mp)$. A similar argument can be made for the curbpoint single approach. However, in this case the number of control points, p, will be roughly twice that of centerline since the control points are generated from both curbs instead of one. Now, consider the curbpoint doubling approach. For each placement, it spawns two layouts. If the layouts each have m manholes, then this approach will generate $2^m - 1$ layouts to be considered. Therefore, the number of line segment tests required for this approach is on the order of $O(mp2^m)$

The implementation of the frontier approach casts r rays to grow the frontier, where r is the number of rays used to sample the propagation of the complete frontier. That is, from each point in the previous frontier, r rays are cast for the next frontier. From each of these rays, s points are sampled along the ray. It seems that this approach will produce a large number of points very quickly, r after the first frontier, r^2 along the second frontier, and in general r^n on the n^{th} frontier. Using this logic, the number of line segment tests required will be on the order of $O(sr^m)$. However, not every point will belong to the sampled frontier. In this implementation, the true frontier is sampled by q regions. Each of these regions represents a distance range from one of the curbs. For each region, the frontier point with the most forward progress toward the goal is chosen as a true frontier point. Consider the road segment show in Figure 4.14, where the pipeline is being placed from left to right. The frontier for this road is sampled using three regions, where the regions are separated using dashed lines. Regions 1 and 2 contain only one point each, p_1 and p_2 respectively. Each of these points will be the chosen frontier sample point for their respective regions. Region 3 contains two points, p_3 and p_4. Since the point p_4 represents further forward progress along the road, it is chosen as the frontier point for region 3.[1]

Therefore, there will be a maximum q frontier points produced during each iteration. The number of line segment tests required for this algorithm will be on the order of $O(qrsm)$. While this may seem like an expensive operation, sample sizes can be chosen relatively small without affecting the quality of the algorithm. Therefore, frontier placement is generally not computationally expensive

[1]For any point within the road, its progress toward the goal is measured as follows. One of the two curbs is arbitrarily chosen as a basis for the measurement. To measure the forward progress represented by a point, it is first projected to the closest point on the chosen curb. Progress is measured as the distance between the start of the curb and this projected point. So, the point with the largest progress distance is considered the point with the greatest forward progress

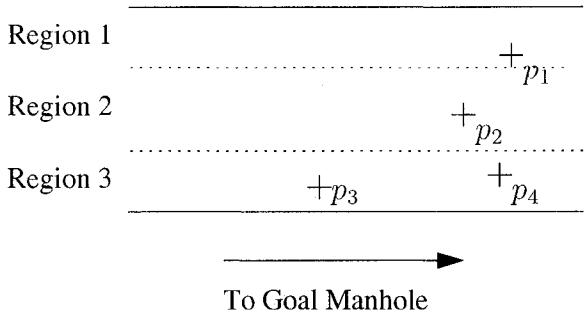

Figure 4.14: Frontier sampling example

and should be preferred to each of the other placement algorithms.

4.6 Conclusion

When given the freedom to place pipes anywhere underneath the road, the curbpoint and frontier approaches are often capable of generating layouts that save manholes. However, in some cases engineers are restricted to lay the pipeline in the center of the road. If this is the case, the centerline placement scheme can be used to automatically and quickly produce such a layout. In other cases, the placement of pipes and manholes is limited to a smaller corridor referred to as a conduit. In this case, the curbs of the road may be replaced with the boundaries of the conduit and any of the aforementioned algorithms may be used. The centerline and curbpoint single approaches should be used when local placement is required as a fast primitive operation. When more time can be allotted, frontier placement should be used.

The algorithms presented in this chapter are effective for placing a pipeline between two fixed points. This is only a small sub-problem of the sewer system planning problem. However, these algorithms can be used as primitive components for algorithms used for planning an entire sewer system. The next chapter introduces several higher level algorithms that use local placement to generate a pipeline for each of the roads in an entire neighborhood.

Chapter 5

Global Manhole Placement

In the previous chapter, techniques for placing a pipeline between two fixed points were introduced. Using these techniques, this chapter examines algorithms to reduce the number of manholes and pipes in a placement for an entire neighborhood. The approach taken is to fix a manhole at each intersection, dividing the problem into several separate local placement problems. By defining a set of candidate manholes for each intersection, the global placement problem is transformed to a discrete domain. Two algorithms to choose candidate manholes for each intersection are presented. The first sequentially fixes intersection manholes based on a greedy heuristic measuring the best layout for directly adjacent roads. The second approach employs a branch-and-bound search to find the best set of fixed manholes with respect to the manhole discretizaton. The performance of the branch-and-bound search is improved with the introduction of two enhancements. The algorithms presented in this chapter are evaluated using real neighborhoods.

5.1 Modeling the Problem

Global placement is based on the assumption that each intersection contains a manhole. This assumption is justifiable. Intersections are components where multiple road segments merge. Such merging constitutes a change in pipe direction for at least one pipeline, which requires a manhole by definition. Therefore it is reasonable to say that the globally optimal layout should contain a manhole in each intersection.

The previous argument motivates the necessity of a manhole for each intersection. However, the fixed manhole may lie anywhere within the intersection. The algorithms presented here are based on the hypothesis that the placement of the fixed manholes affects the number of manholes in a pipeline between two intersections. Using this assumption, a general two step solution strategy is proposed.

1. Fix one manhole for each intersection;

2. Using these fixed manholes, apply local placement to each road component.[1]

[1] In some cases, one end of a road will not be adjacent to an intersection, rather it ends at a dead-end. In this case, a fixed point can be chosen anywhere in the region of the dead-end.

The algorithms in this chapter address the first step. The local placement problems in step two are solved using algorithms from Chapter 4.

Consider fixing a manhole for a single intersection. The domain of potential manhole positions is continuous; any point that lies inside the geometry delimiting the intersection is a valid manhole position. The approach taken is to transform this continuous space into a discrete one. Each intersection will be sampled, resulting in a discrete set of fixed candidates. This discretization allows the application of discrete search techniques to evaluate different sets of fixed candidate manholes.

Transforming the candidate manhole set into a discrete space facilitates the use of discrete search-based algorithms. However, this transformation can potentially result in a loss of optimality. If the transformation from the continuous domain to the discrete domain does not include a set of optimal candidate manhole positions, it will not be possible to find the optimal solution.

By fixing a candidate manhole for each intersection, the problem is transformed into separate local placement problems; one for each road in the neighborhood graph. For each road, the algorithms in this chapter fix a single point in both adjacent intersections. Using a local placement algorithm from Chapter 4, a pipeline can be generated between the two fixed points, where the geometry of the road is the simple polygon.

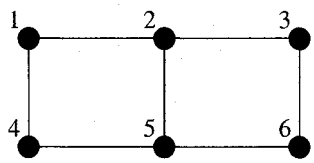

Figure 5.1: Fixed nodes and potential connections

Determining the optimal placement of pipes and manholes for an entire neighborhood is a complex problem. The placement of a single intersection manhole is highly correlated with the placement of other intersection manholes. Consider the process of choosing a candidate manhole set for the example graph structure shown in Figure 5.1, where each of the vertices represent intersections and the edges represent roads. Let the process start by choosing the best manhole for intersection 2. The position of this manhole affects the quality of the solutions for the roads $road_{12}$, $road_{23}$ and $road_{25}$. To choose the best manhole position for intersection 2 one may fix the manholes for intersections 1, 3, and 5 such that the pipelines for the roads adjacent to intersection 2 contain the fewest manholes. However, this local optimization approach may not result in a globally optimal solutions. Consider intersection 1. In addition to $road_{12}$, it is also connect to intersection 4 via $road_{14}$. Therefore, intersection 2 is indirectly related to intersection 4, as the position of the fixed manholes for intersections 2 and 4 must be considered when determining the best manhole position for intersection 1. An algorithm designed to find the globally optimal solution must consider each of these correlations.

5.2 Definitions and Basic Functions

In this section, some terminology and functions used in the algorithm pseudo-code from this chapter are introduced. Define N to be a neighborhood, and I and R to be the set of intersection and road components of the neighborhood respectively. Let the number of intersections be $|I| = n$ and the number of roads be $|R| = m$. Further, let $i_j \in I$ be the j^{th} intersection. The function $layout(I)$ defines a mapping from each $i \in I$ to its corresponding fixed manhole. If no manhole has been fixed for i, then $layout(i) = \emptyset$. A solution to the global manhole placement problem is a mapping, $layout$, such that $layout(i_j) \neq \emptyset, \forall i_j \in I$. In other words, a manhole has been fixed for each intersection of N. For the purpose of the algorithms presented here, these solutions will be considered complete, as the actual layout can be generated from this mapping using local placement algorithms.

The cost of the placement for a road is defined to be the number of manholes in the placement between the two fixed endpoints. The function $PLACE(c_1, c_2)$ will be used to denote a generic local placement between manholes c_1 and c_2 and the number of manholes in the placement is denoted $|PLACE(c_1, c_2)|$. To define the manhole placement for each road, a manhole must be fixed for both adjacent intersections. Each pairing of fixed manholes in both intersections adjacent to a road will be referred to as a candidate manhole pair.

The pseudo-code presented in this chapter utilizes several simpler functions. A description of these functions is presented below.

ROAD(N): This function returns the set of roads R contained in neighborhood N.

ADJACENTINTERSECTIONS(r, i_1, i_2): Given a road r, returns the two adjacent intersection i_1 and i_2.

GETOTHERADJACENTINTERSECTION(r, i): Given a road r and an intersection i, this function returns the other intersection adjacent to this road

BUILDINTERSECTIONLIST(N): This is a generic function which given a neighborhood N, builds an ordered list of intersections. This function may be implemented using any intersection ordering heuristic.

LOWESTCOSTROAD(r): For a road r, returns the number of manholes in the minimum cost placement for the road. This function considers each pair of fixed manholes for a road.

GETGRID(i): For an intersection i, returns the set of candidate manholes.

EMPTY(l): Given a list l, this function returns whether this list is empty or not.

POPFRONT(l): Given a list l, this function returns the front item and removes it from the list.

Partial solutions are an integral concept to the algorithms presented in this chapter. A partial solution is a mapping where manholes are fixed for some intersections, and not fixed for others. If a

manhole is not fixed for the intersection i_k, the mapping function returns $layout(i_k) = \emptyset$. The size of the map corresponds to the number of fixed manholes, and $layout$ defines a complete solution when $|layout| = |I|$.

To evaluate the quality of a partial solution, a cost estimation function is necessary. Pseudo-code for the cost estimation function is presented in Figure 5.2. This estimation provides a lower-bound based on the best achievable layout for each individual road. This estimate is calculated as the sum of cost estimates for each road in N. The estimate for an individual road depends on one of three situations:

1. Both adjacent manholes are fixed. In this case, the exact number of manholes in the pipeline for the road is used. In the pseudo-code shown in Figure 5.2, this cost is calculated by Case I.

2. Neither adjacent manhole is fixed. In this case, the estimated cost corresponds to the lowest number of manholes in a pipeline between any of the candidate manhole pairs. This quantity is calculated by Case II in Figure 5.2.

3. One adjacent manhole is fixed. The estimated cost is the lowest number of manholes from a pipeline using the one fixed candidate manhole. Cases III and IV from the pseudo-code in Figure 5.2 represent this situation.

Note that this cost estimation function requires the calculation of the local placement cost for each corresponding candidate manhole pair. The cost of a single candidate pair may be calculated multiple times. Therefore, for each road the cost of placements for each candidate manhole pair is pre-calculated and cached in a cost map. Note that if the size of each candidate manhole set is p, that means that there are p^2 candidate pairs per road. Therefore, this pre-processing step will calculate exactly mp^2 local placements. This number will typically be small. Thus, this pre-processing step is not a key computational consideration for most global placement algorithms.

```
ESTIMATECOST(R, layout)

/*
Input Variables:
R: A set of roads for which to estimate a layout cost.

layout: A partial layout defining the current set of fixed manholes

Returns:The sum of the estimated costs for each r ∈ R
*/
cost ← 0
for each r ∈ R
do {
    /* Get intersections adjacent to road r */
    ADJACENTINTERSECTIONS(r, i1, i2)
    /* Case I:Both adjacent manholes fixed, use precise cost */
    if layout[i1] ≠ ∅ and layout[i2] ≠ ∅
      then cost ← cost + |PLACE(layout[i1], layout[i2])|

    /* CASE II: If neither adjacent manhole is fixed, the least number of manholes
     in any pipeline generated from a candidate pair is added to the cost */
      else if layout[i1] = ∅ and layout[i2] = ∅
      then {cost ← cost + LOWESTROADCOST(r)

    /* Case III: If layout[i2] ≠ ∅, then the lowest possible road cost
    with i2 fixed is added to the total cost */
      else if layout[i1] = ∅
      then {grid = GETGRID(i1)
            mincost = ∞
            for each m ∈ grid
              do {if |PLACE(m, layout[i2])| < mincost
                  then mincost ← |PLACE(m, layout[i2])|
            cost ← cost + mincost

    /* Case IV: If layout[i1] ≠ ∅, then the lowest possible road cost
    with i1 fixed is added to the total cost */
      else if layout[i2] = ∅
      then {grid ← GETGRID(i2)
            mincost ← ∞
            for each m ∈ grid
              do {if |PLACE(layout[i1], m)| < mincost
                  then mincost ← |PLACE(layout[i1], m)|
            cost ← cost + mincost
}
return (cost)
```

Figure 5.2: Pseudo-code for function to estimate an upper bound on the cost for a road

5.3 Traversing the Candidate Manhole Space

Consider the different ways to fix candidate manholes as a discrete solution space S. To minimize the cost of the layout, the best set of fixed manholes from S must be found. However, as the number of candidate manholes grows, the size of this space can become quite large. Consider a neighborhood with n intersections and p candidate manholes for each intersection. The number of ways to fix a single manhole for each intersection is then p^n. As the values of p or n grow, enumerating all of the sets of fixed manholes is not computationally feasible. More sophisticated algorithms are required to traverse the solution space. In this section, two algorithms to traverse the candidate manhole space are proposed. The first algorithm defines an iterative process where manholes are fixed based on a localized greedy heuristic. The second algorithm employs a branch-and-bound search strategy to find the optimal placement subject to the constraints in the problem formalization[24]. [2]

5.3.1 Greedy Sequential Manhole Placement

The greedy sequential manhole placement approach fixes manholes in an iterative fashion, where each iteration corresponds to a single intersection. For each intersection, a localized cost heuristic is used to determine the best candidate manhole. The chosen manhole is the one that minimizes the sum of the cost estimates of the adjacent roads.

One way to potentially improve the performance of this algorithm is to fix the order in which intersections are processed. One heuristic explored orders intersections based on the number of adjacent roads, referred to as the degree of the intersection. Using this heuristic, intersections are sorted by descending degree. The rationale behind this is that fixing a manhole for a higher degree intersection will have more effect on the optimization since this manhole will be involved in the placement of more roads.

The pseudo-code for this algorithm is presented in Figure 5.3. This algorithm starts by ordering the intersections. The function BUILDINTERSECTIONLIST represents a generic function that builds the list of intersections. This function may implement any intersection ordering scheme. Each iteration of the *while* loop corresponds to the process of fixing a candidate manhole for a single intersection. Consider the iteration for fixing a manhole for intersection i. The outermost *for* loop, **for each** $m \in grid_1$, iterates through all of the candidate manholes for intersection i to identify the one with the lowest cost estimate. The next two nested *for* loops are used to find the lowest cost estimate for each road adjacent to i. At the end of each *while* loop iteration, the candidate manhole that produces this lowest cost estimate is fixed for the current intersection. Note that $grid_2$ represents the candidate manhole grid for the intersection at the opposite end of an adjacent road. It is possible that a candidate manhole has been fixed for i_2 during a previous iteration. In this case, the fixed manhole is used instead of the candidate manhole grid.

[2] This placement is optimal with respect to the discrete grid of possible manhole positions and the chosen local placement algorithms. A better solution may exist in the continuous problem domain, but is not be represented in the discrete domain.

```
GREEDYSEQUENTIALLAYOUT(N)

/*
Input Variables:
N: The neighborhood graph, containing the roads and intersections

Returns:
layout: Specifies the fixed manhole for each intersection in N
*/

/* Build an ordered list of intersections from N */
IntList ← BUILDINTERSECTIONLIST(N)

/* Fix the manhole for each intersection in the list sequentially */
while not EMPTY(IntList)
do { i ← POPFRONT(IntList)
     /*Initialize best manhole to null, and the estimated cost for that manhole to ∞ */
     bestManhole ← ∅
     minAdjacentRoadCost ← ∞

     grid1 ← GETGRID(i)
     /*This loop estimates the cost associated with each candidate manhole for i.*/
     for each m1 ∈ grid1
     do { adjacentRoadCost ← 0
          for each road r adjacent to i
          do { i_opposite ← GETOTHERADJACENTINTERSECTION(r, i)

               /* If a manhole is fixed for i_opposite, add this as the only point in
               grid2, otherwise grid2 gets the entire set of candidates*/
               if layout[i_opposite] = ∅
                 then grid2 ← GETGRID(i_opposite)
                 else grid2 ← layout[i_opposite]

               minRoadCost ← ∞
               for each m2 ∈ grid2
               do { if |PLACE(m1, m2)| < minRoadCost
                    then minRoadCost ← |PLACE(m1, m2)| }
               adjacentRoadCost ← adjacentRoadCost + minRoadCost }

          /* If the sum of estimated costs for adjacent road is less than the previous
           best then set the best manhole to m1 */
          if adjacentRoadCost < minAdjacentRoadCost
            then { minAdjacentRoadCost ← adjacentRoadCost
                   bestManhole ← m1 }

     layout[i] ← bestManhole
return (layout)
```

Figure 5.3: Pseudo-code for the greedy sequential global placement algorithm

5.3.2 Branch-and-Bound Placement

As previously discussed, pure enumeration is often not a computationally feasible method for finding the best set of candidate manholes. To find this best set without generating each combination, a branch-and-bound approach is used.

Instead of sequentially fixing only one manhole per intersection, a backtracking approach is used to evaluate multiple solutions. Partial solutions only need to be expanded if the lower bound on the cost is less than the cost of the current best solution. This lower bound for a partial solution is found by applying the cost estimate function defined in Figure 5.2. Pseudo-code for the branch-and-bound algorithm is defined in Figures 5.4 and 5.5.

```
BRANCH-AND-BOUND(N)

/*
Input Variables:
N: The neighborhood graph, containing the roads and intersections

Returns:
bestlayout: Specifies the fixed manhole for each intersection in N
*/
lowestcost ← ∞
intersections ← BUILDINTERSECTIONLIST(N)
/* Initialize variables for the current and best layouts used by BBRecur */
emptylayout ← ∅
bestlayout ← ∅

/* Recursively search each set of fixed candidate manholes, returning the best one */
BBRECUR(N, intersections, emptylayout, bestlayout, lowestcost)
return (bestlayout)
```

Figure 5.4: Pseudo-code for branch-and-bound global placement algorithm

```
BBRECUR(N, intersections, layout, bestlayout, lowestcost)

/*
Input Variables:
N: The neighborhood graph, containing the roads and intersections

intersections: The ordered intersection list

layout: The partial layout being expanded by this function

Input/Output Variables:

bestlayout: The best layout built so far. When this function exits, the
 best layout found during the recursive descent is returned through this variable.

lowestcost: The number of manholes in the best layout built so far. When this
function exits, the number of manholes in the best layout is returned through this
variable.
Returns:
 Both the best layout and the number of manholes it contains through bestlayout
 and lowestcost variables respectively.*/

/* Get the set of roads from the neighborhood */
R ← ROADS(N)

/* If there are no intersections left in intersections, this constitutes a complete layout.
 Check if this complete layout is better than the previous best complete layout, if so then
 it is set as the current best layout. */
if EMPTY(intersections)
  then { if ESTIMATECOST(R, layout) < lowestcost
         then { lowestcost ← ESTIMATECOST(R, layout)
                bestlayout ← layout

  else
{ i ← intersections.POPFRONT()
  /* Returns the set of non-dominated candidate manholes. This function implements
   the domination measure described in Section 5.3.2. */
  grid ← NONDOMINATEDCANDIDATES(i)

  /* Recursively expand a layout for each non-dominated candidate manhole */
  for each m ∈ grid
      do { newlayout ← layout
           newlayout[i] ← m
           cost ← ESTIMATECOST(R, newlayout)
           if cost < lowestcost
             then BBRECUR(N, intersections, newlayout, bestlayout, lowestcost)
```

Figure 5.5: Pseudo-code for the recursive function implementing the branch-and-bound global placement algorithm

Two enhancements have been added to the branch-and-bound algorithm to further prune the search space. The first enhancement prunes candidate manholes which are provably inferior to another candidate manhole. This approach is referred to as *domination* and can be applied at every node in the search. In the pseudo-code, the grid of non-dominated manholes is returned by the function NONDOMINATEDCANDIDATES.

This function prunes the candidate manhole set in the following manner. Consider the candidate manholes for an n-way intersection. For each candidate, two vectors of length n will be calculated. These vectors contain the cost of the best and worst placement for each of the adjacent roads respectively. To calculate the values for these vectors, the cost of the layout between each candidate manhole pair must be calculated. A candidate manhole, c_i, dominates another candidate manhole, c_j, if every component for the worst solution in c_i is equal to or better than the corresponding component for the best solution of c_j.

Consider a four-way intersection I_0, where R_1, R_2, R_3, R_4 and I_1, I_2, I_3 I_4 represent the adjacent roads and their intersections respectively, as shown in Figure 5.6. The intersection I_0 is connected to the roads R_1, R_2, R_3 and R_4. This means that there will be four components in the domination vectors. One value represents the cost to place the pipeline for R_1, one the cost for R_2, the third the cost for R_3 and the fourth the cost for R_4.

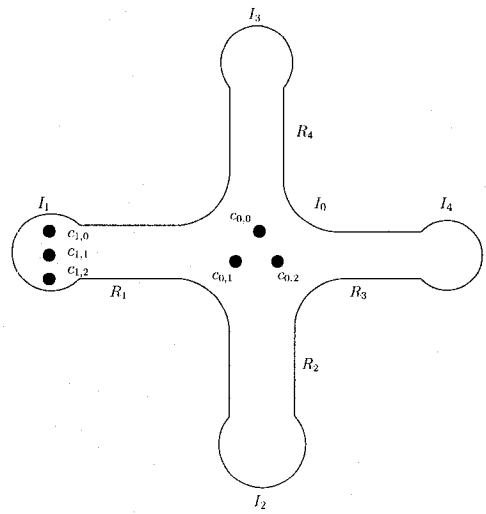

Figure 5.6: Example domination for intersection 0

Now, let I_0 have three candidate manholes: $c_{0,0}$, $c_{0,1}$, $c_{0,2}$. Each of these manhole candidates will have one vector representing the best placements, and one the worst. As well consider intersection I_1, which has three candidate manholes $c_{1,0}$, $c_{1,1}$, $c_{1,2}$. This example configuration is shown in Figure 5.6.

The R_1 component for each domination vector is calculated as shown at the top of the following page. The other three vector components are calculated in a similar manner.

$$c_{0,0}^{worst} = max(|PLACE(c_{0,0}, c_{1,0})|, |PLACE(c_{0,0}, c_{1,1})|, |PLACE(c_{0,0}, c_{1,2})|)$$

$$c_{0,1}^{worst} = max(|PLACE(c_{0,1}, c_{1,0})|, |PLACE(c_{0,1}, c_{1,1})|, |PLACE(c_{0,1}, c_{1,2})|)$$

$$c_{0,2}^{worst} = max(|PLACE(c_{0,2}, c_{1,0})|, |PLACE(c_{0,2}, c_{1,1})|, |PLACE(c_{0,2}, c_{1,2})|)$$

$$c_{0,0}^{best} = min(|PLACE(c_{0,0}, c_{1,0})|, |PLACE(c_{0,0}, c_{1,1})|, |PLACE(c_{0,0}, c_{1,2})|)$$

$$c_{0,1}^{best} = min(|PLACE(c_{0,1}, c_{1,0})|, |PLACE(c_{0,1}, c_{1,1})|, |PLACE(c_{0,1}, c_{1,2})|)$$

$$c_{0,2}^{best} = min(|PLACE(c_{0,2}, c_{1,0})|, |PLACE(c_{0,2}, c_{1,1})|, |PLACE(c_{0,2}, c_{1,2})|)$$

Figure 5.7 shows an example of how domination relations are determined once the vectors are calculated. A three-way intersection with three candidate manholes is considered. In this example, c_2 dominates c_1, since for every adjacent road, the worst placement for c_2 has the same number or fewer manholes than the best solution for c_1. As well, the worst solution using c_2 is equivalent to the best solution using c_3. So c_2 also dominates c_3. This means that the worst possible placement using c_2 is at least as good as the best possible placements using c_1 and c_3. Therefore, c_1 and c_3 can be pruned from the search without affecting solution quality.

Candidate	Best	Worst
c_1	$\{2,4,5\}$	$\{3,4,5\}$
c_2	$\{2,3,4\}$	$\{2,4,4\}$
c_3	$\{2,4,4\}$	$\{3,5,5\}$

Figure 5.7: Domination example where c_2 dominates c_1 and c_3

Using domination, pruning also occurs when the cost for each adjacent road is the same regardless of the chosen candidate manhole. When this occurs, the worst and best vectors for each candidate manhole will be identical. An example of this is shown in Figure 5.8. When this occurs, the branch-and-bound search only needs to expand solutions using a single one of these candidate manholes.

Candidate	Best	Worst
c_1	$\{2,4,5\}$	$\{2,4,5\}$
c_2	$\{2,4,5\}$	$\{2,4,5\}$
c_3	$\{2,4,5\}$	$\{2,4,5\}$

Figure 5.8: Domination example where the choice of candidate manhole is inconsequential

The computation required for domination can be reduced by filling the best placement vectors first. Once these are filled, the worst manhole placement vectors can be assembled one at a time. After the worst manhole placement vector is calculated for a manhole, it can be compared to the best placement vector for each other manhole. If this comparison yields manhole domination, the

remaining worst manhole placement vectors need not be calculated. From the example above, c_2 gives the best solution. Generating each of the best manhole placement vectors, and the worst placement vector for only c_2 is sufficient to show that c_2 is a dominant manhole position.

Another enhancement is to separate the neighborhood graph into smaller, independent sub-graphs. This will be referred to as graph reduction. Graph reduction is based on the following observation. When calculating the placement for a road segment, the choice of candidate manholes for each adjacent intersection may not affect the cost of the road. In some cases, a placement algorithm may generate a layout for the road containing the same number of manholes regardless of the candidate manhole pairing. Such a road will not affect the global placement, as no matter which manholes are fixed, the placement of the road has the same cost. Conceptually, these roads can be removed from the graph. For the adjacent intersections, any choice of candidate manholes is later used to generate the pipeline. By disregarding these roads, the search space is reduced, and sometimes separated into several independent sub-graphs. An example is presented in Figure 5.9. In this graph the vertices represent intersections, and the edges roads. Consider road R_5 in the graph on the left. If the placement algorithm for each candidate manhole pairing for R_5 generates a layout of the same cost, then R_5 can conceptually be removed. In this case, the two sub-graphs on the right are created. The manholes may be fixed for both of these sub-graphs independently without affecting the global layout cost.

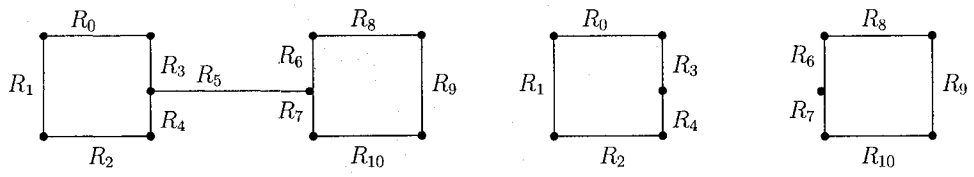

Figure 5.9: Sub-graph example: Removing R_5 on the left creates two sub-graphs to be solved, shown on the right

Recall that the description for the greedy sequential algorithm explored the use of an ordering heuristic. The use of such a heuristic may improve the performance of the branch-and-bound search as well. The branch-and-bound algorithm presented in this chapter prunes partial solutions if the minimum bound of their cost is no better than the best observed solution. The goal of such a heuristic is to fix intersection manholes so that at each step the minimum cost estimate is as close to the minimum solution cost as possible. In doing this, non-optimal partial solutions may be recognized and pruned at a higher level of tree. The effect of intersection ordering is explored in the next section using several random ordering techniques.

Another issue with the branch-and-bound search is the cost of the initial solution. If no extra information is available, the initial minimum cost bound is set to infinity. As better solutions are found, the value for this cost decreases. However, the performance of the search is highly dependent on the quality of solutions discovered early during the search. If the quality of the first solution

discovered is poor, then many other poor solutions may be fully expanded before a better solution is found. One way to overcome this issue is to initialize branch-and-bound with solution from another algorithm. This can be done using the solution from the greedy sequential algorithm. As will be seen in Section 5.4, the greedy sequential algorithm typically requires much less computation than the branch-and-bound and often provides good quality solutions to this problem.

5.4 Results

Performance of the global placement algorithms is evaluated using data from three real neighborhoods provided by David Schaeffer Engineering. Figure 5.10 provides a breakdown of the graph components for each neighborhood, while the neighborhoods themselves are shown in Figures 5.11, 5.12 and 5.13. To test the computational limits of branch-and-bound and its enhancements, a sub-neighborhood from Neighborhood C is used. This sub-neighborhood is shown in Figure 5.14. To provide multiple data points for each test neighborhood, the maximum pipe length is varied.

This section presents tests to evaluate both the greedy sequential and branch-and-bound algorithms in terms of both solution quality and computation time. First, the number of manholes in placements generated by both algorithms is compared to several basic placement schemes. Second, solution times for each algorithm are explored. Third, the effect of the domination and graph reduction branch-and-bound enhancements are evaluated by measuring the number of nodes and solution times for several problem instances. Finally, the performance of the branch-and-bound algorithm is further explored by solving instances using different intersection orderings.

Neighborhood	Cul-de-Sacs	Three-Ways	Four-Ways	Roads
A	5	4	3	16
B	0	14	2	26
C	0	30	4	55
C_{23}	0	20	3	40

Figure 5.10: Component breakdown for test data

With the exception of cul-de-sacs, each intersection has five candidate manholes. For cul-de-sacs, only the center point will be considered as this manhole placement only affects the single adjacent road. For each other intersection type, a grid containing five points has been defined. The position of the candidate manholes have been pre-defined to include the center of the intersection as well as a point close to the opening to each road. For three-way intersections, a fifth manhole candidate is defined as the average between the center point and another arbitrarily chosen fixed point. These grids are displayed in the figures for the test neighborhoods.

5.4.1 Solution Quality

As with the local placement algorithms, the solution quality for global placement is measured in terms of the number of manholes. Results have been generated using frontier placement as the local

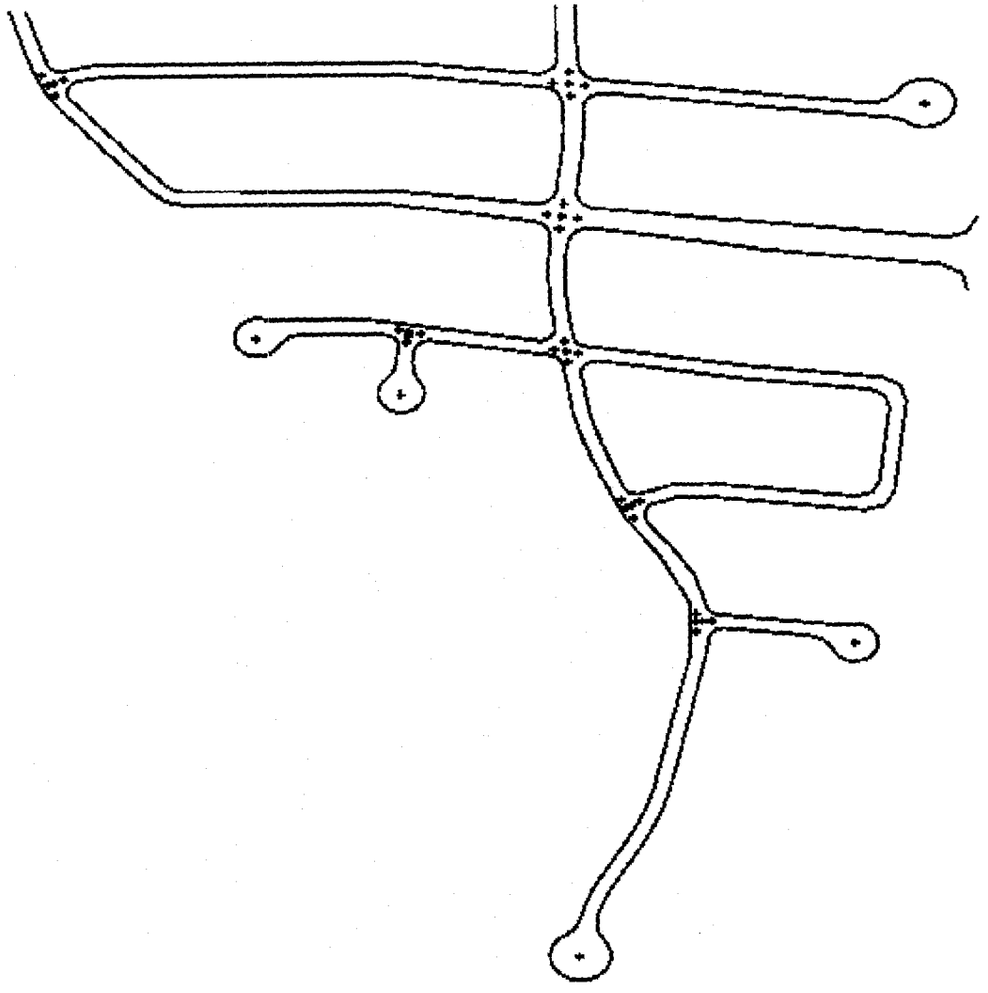

Figure 5.11: Neighborhood A

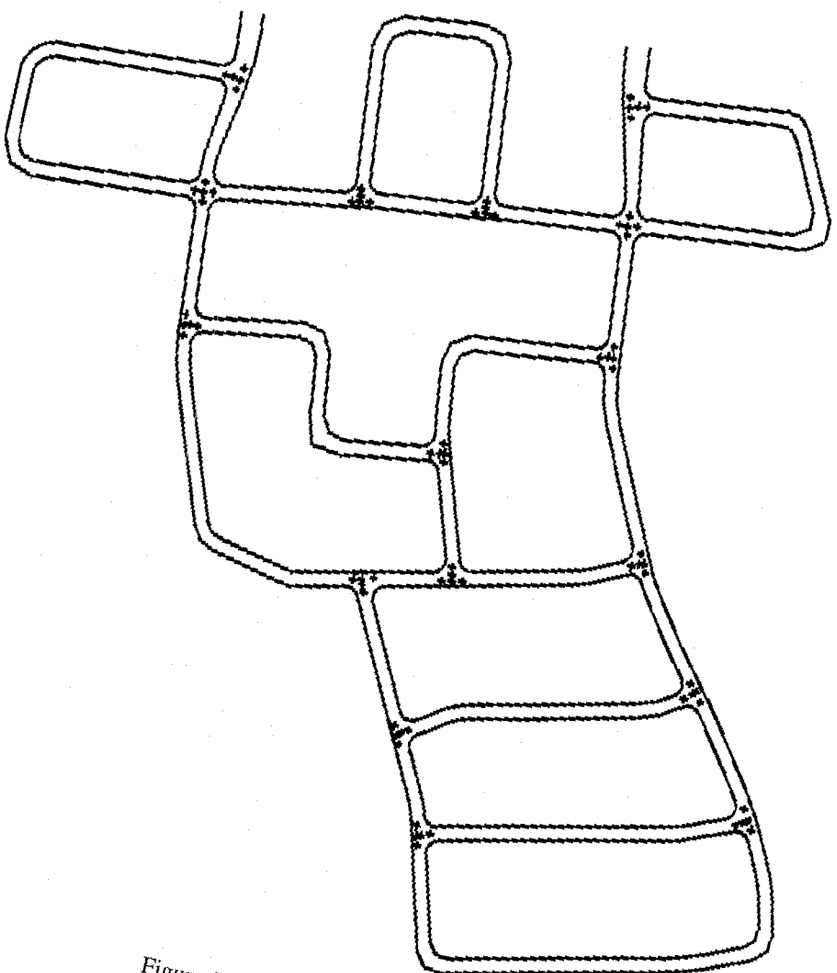

Figure 5.12: Neighborhood B

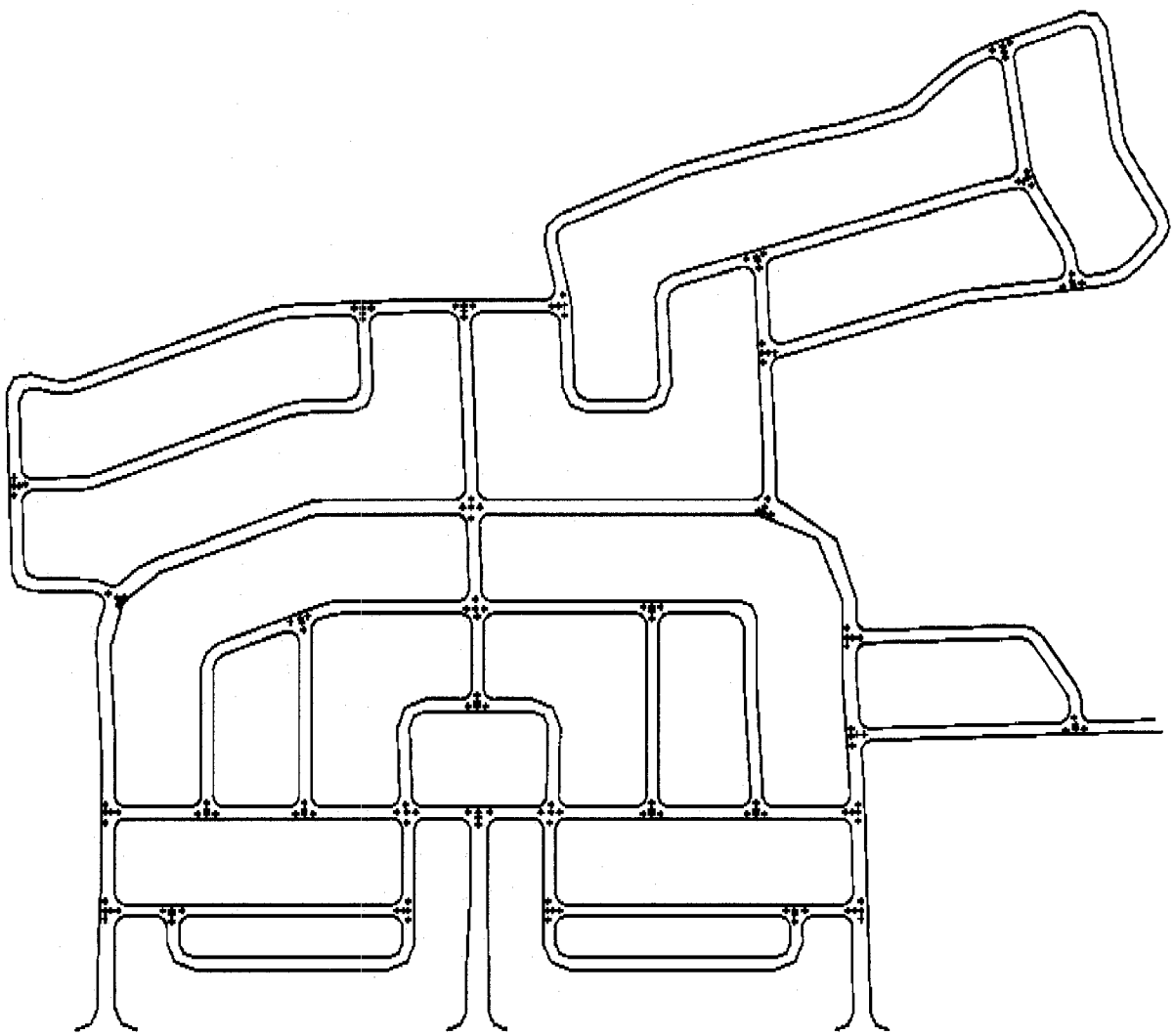

Figure 5.13: Neighborhood C

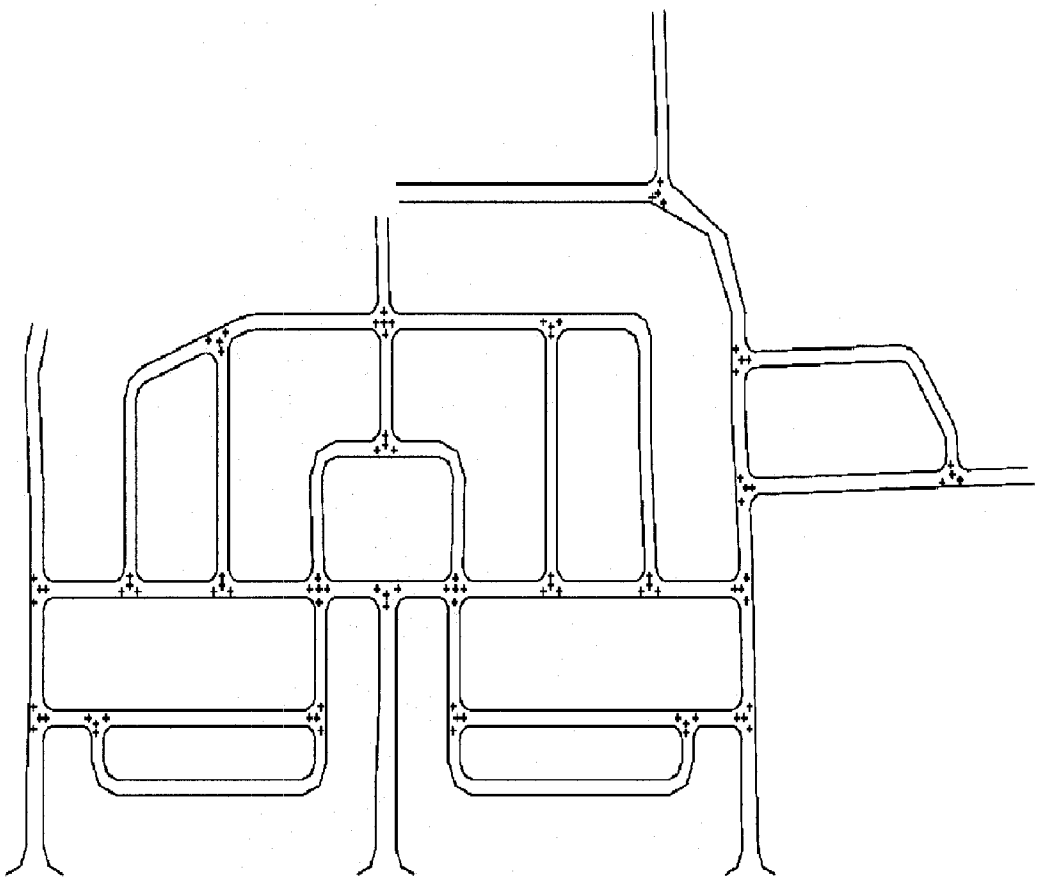

Figure 5.14: Neighborhood C_{23}. A subset of neighborhood C with 23 intersections

placement algorithm. Recall that the branch-and-bound algorithm will find the best possible solution with respect to both local placement algorithm and selection of candidate manholes. Therefore, the best solution for each test instance will correspond to the results of the branch-and-bound approach.

Results for five global placement algorithms are presented in Figures 5.15, 5.16 and 5.17. The first two algorithms are simplistic approaches used as a baseline for the greedy sequential and branch-and-bound approaches introduced in this chapter. The first simple algorithm randomly chooses a candidate manhole from the discrete candidate manhole grid for each intersection. This technique has been implemented to validate the assertion that the position of the intersection manhole is important. Results for this approach are presented in the *Random Fixed* column. These results are an average over twenty-five runs using this algorithm.

Following this, another simplistic algorithm that places a candidate manhole at the center of each manhole is evaluated. This approach is used to mimic a simplistic rule an engineer may employ. Intuitively, the number of manholes in a global placement generated using this approach should be reasonable. The quality of solutions generated using this simple rule are presented in the column entitled *Center* in the following results tables.

Two variations of the greedy sequential algorithm were implemented. The first variation does not implement a fixed intersection ordering scheme, rather it measures the performance of the algorithm using numerous random intersection orderings. For testing, solutions were generated for twenty-five random orderings. The column entitled *GS-Random* in Figures 5.15, 5.16, 5.17 presents only the cost associated with the best solution from twenty-five random intersection orderings. The other implementation of the greedy sequential algorithm investigates the effectiveness of the descending degree ordering heuristic, with results presented in the *GS-Descending Degree* column. Finally, the number of manholes generated using the branch-and-bound solution are presented in the column labeled *B-and-B*.

	Algorithm				
Max Length	Random Fixed	Center	GS-Random	GS-Descending Degree	B-and-B
50	52.6	50	48	48	48
60	46.0	44	44	44	44
70	42.6	41	41	41	41
80	40.4	38	36	36	36
90	37.0	33	31	31	31
100	36.3	32	29	30	29

Figure 5.15: Number of manholes in layout for Neighborhood A where Random Fixed is averaged over twenty-five runs and GS-Random is the lowest cost over twenty-five runs

From the results, it is clear that randomly fixing the candidate manholes produces solutions of considerably poorer quality than the other algorithms discussed. In every test instance, random candidate manhole selection performs significantly worse than the more sophisticated algorithms introduced in this chapter. These results support the hypothesis that the position of the fixed intersection manholes has an effect on the quality of the global neighborhood layout.

	Algorithm				
Max Length	Random Fixed	Center	GS-Random	GS-Descending Degree	B-and-B
50	75.9	75	72	73	72
60	67.0	63	62	63	62
70	60.2	56	53	54	53
80	57.2	50	48	49	48
90	55.4	45	45	49	45
100	51.0	42	40	41	40

Figure 5.16: Number of manholes in layout for Neighborhood B where Random Fixed is averaged over twenty-five runs and GS-Random is the lowest cost over twenty-five runs.

	Algorithm				
Max Length	Random Fixed	Center	GS-Random	GS-Descending Degree	B-and-B
50	101.1	99	94	97	94
60	91.7	88	82	82	80
70	81.8	69	67	68	67
80	75.0	60	59	59	59
90	72.0	58	57	57	57
100	70.1	57	55	56	55

Figure 5.17: Number of manholes in layout for Neighborhood C_{23} where Random Fixed is averaged over twenty-five runs and GS-Random is the lowest cost over twenty-five runs.

Consider the global placements generated by fixing the candidate manhole in the center of each intersection. These results indicate that the center points are a good choice for the intersection manhole. In each test case, the layout produced using this rule greatly outperforms random selection. However, the greedy sequential and branch-and-bound approaches generate fewer manholes in all but three test instances; pipe lengths 60 and 70 of Figure 5.15 and pipe length 90 in Figure 5.16. In the majority of test cases, the number of manholes in the center-based solution is within one to three manholes of the best solution. However, the discrepancy between the center-based and best solutions are not always as close. For example, consider maximum pipe length 60 for Neighborhood C_{23} in Figure 5.17. In this case, the center-based solution has eight more manholes than the best observed solution; making it 10% more expensive. From these results, it can be concluded that the practice of placing manholes in the center of each intersection is better than purely random placement, but is not guaranteed to produce good quality solutions. Since this approach requires a single local placement to be calculated for each road, it is a computationally inexpensive way to generate solutions. However, both the greedy sequential and branch-and-bound algorithms generate higher quality solutions.

The greedy sequential algorithm implemented with the descending degree ordering heuristic produces good quality solutions, but these solutions often contain more manholes than the branch-and-bound solution. In 11 of the 18 test instances shown in Figures 5.15, 5.16 and 5.17, the layouts contain more manholes than the branch-and-bound approach. A particularly poor solution can be observed for Neighborhood B with maximum pipe length 90, shown in Figure 5.16. In this case, the descending degree heuristic generates a solution with 4 more manholes, or 8.8% more expensive,

than the solution generated using the branch-and-bound approach.

Results show that the greedy sequential algorithm can often equal branch-and-bound solutions by considering a relatively small number of intersection orderings. In all but one test instance, at least one of the twenty-five random orderings generates a solution with the same number of manholes as the branch-and-bound approach. For the exception instance, which occurs in Neighborhood C_{23} with a maximum pipe length of 60, the solution quality still outperforms both the random fixed and center placement approaches by 9.7 and 6 manholes respectively. In addition, the best observed greedy sequential solution is within 2 manholes, or 2.5%, of the solution generated using the branch-and-bound algorithm.

With only twenty-five randomized intersection orderings, the greedy sequential approach is often able to generate solutions with the minimum number of manholes. However, the greedy sequential algorithm with descending degree ordering does not generate the best solution as frequently. Therefore, the recommended use of the greedy sequential technique is to solve the instance multiple times with different random orderings.

As mentioned previously, the branch-and-bound approach generates the best solution corresponding to the chosen local placement algorithm and candidate manhole grids. In many cases, the greedy sequential approach generates solutions that are equal to or slightly more expensive than the branch-and-bound solution. Ideally, the branch-and-bound approach should always be chosen over the other algorithms presented in this chapter. However, this analysis ignores the computation time required for each approach. The following section examines computation time, exploring the usefulness of these algorithms for real sized neighborhoods.

5.4.2 Computation Time

This section explores the computation time required for the greedy sequential and branch-and-bound algorithms. For each of the tests presented here, the branch-and-bound algorithm has been enhanced with both domination and graph reduction. Each algorithm orders intersections by descending degree.

	Algorithm	
Max Length	Greedy Sequential	B-and-B
50	59.4 + 0.003	59.6 + 0.000
60	64.2 + 0.003	63.8 + 0.000
70	65.6 + 0.004	65.7 + 0.060
80	74.2 + 0.005	77.0 + 0.000
90	81.3 + 0.004	81.2 + 2.000
100	85.9 + 0.005	83.6 + 0.025

Figure 5.18: Pre-processing time + Search time results in seconds for Neighborhood B

	Algorithm	
Max Pipe Length	Greedy Sequential	B-and-B
50	32.1 + 0.005	32.8 +0.001
60	33.4 + 0.006	33.3 + 444.0
70	34.2 + 0.005	*
80	36.9 + 0.005	36.7 + 0.013
90	41.0 + 0.005	39.7 + 0.002
100	43.5 + 0.005	43.3 + 0.002

Figure 5.19: Pre-processing time + Search time results in seconds for Neighborhood C_{23}

	Algorithm	
Max Pipe Length	Greedy Sequential	B-and-B
50	95.6 + 0.016	95.4 + 0.001
60	101.88 + 0.018	*
70	103.0 + 0.016	*
80	109.7 + 0.015	*
90	115.8 + 0.017	*
100	127.5 + 0.015	*

Figure 5.20: Pre-processing time + Search time results in seconds for Neighborhood C

The computation times for Neighborhoods B, C_{23} and C are presented in Figures 5.18, 5.19 and 5.20 respectively. Times were measured using a machine with dual 2 GHz AMD Athlon processors and 1 GB of memory. A computation limit of four hours was imposed for each test instance. For instances where computation is incomplete after this time limit, an '*' is recorded in the table. Reported times are separated into two categories. The first category represents pre-processing costs. For both algorithms, this pre-processing is the amount of time required to cache the number of manholes in the road placement for each candidate manhole pairing. The second cost represents the time required by the global placement algorithm to generate the set of fixed manholes. Using this approach, a result reported as 60.0 + 200.0 indicates 60.0 seconds for pre-processing and 200.0 seconds to compute the positions of the fixed candidate manholes.

The first test case involves Neighborhood B, a real neighborhood of medium size. As shown in Figure 5.18, the time required by both greedy sequential and branch-and-bound to calculate the global placement is insignificant. The only case in which the computation is above a fraction of a second is for the branch-and-bound algorithm with a maximum pipe length of 90, where the computation time is 2.0 seconds. For this neighborhood, the bulk of the computation time occurs in the pre-processing phase.

The results for neighborhood sub-graph C_{23}, and the corresponding full neighborhood, Neighborhood C, are presented in Figures 5.19 and 5.20 respectively. Several things can be said from these results. First, these results give evidence that the greedy sequential approach scales well to larger neighborhoods. Similar to Neighborhood B, the greedy sequential computation cost is insignificant, and the dominant cost is in the pre-processing phase.

However, results for the branch-and-bound approach are mixed. For some test instances, a solution cannot be calculated in the allotted four hour time period. Examples of this occur for maximum pipe length of 70 for each of these two neighborhoods. However, in many other cases the algorithm time is actually insignificant. This can be seen in the majority of test instances for neighborhood C_{23} in Figure 5.19 as well as in pipe length 50 for Neighborhood C in Figure 5.20.

These results show a considerable difference in the computation time between problem instances. Even more interesting is that varying the maximum pipe length within a neighborhood can change the solution time drastically. The following two sections will explore properties of the branch-and-bound algorithm. First, the effectiveness of the enhancements is explored. Second, the effect of intersection ordering on the performance of the algorithm is examined.

5.4.3 Branch-and-Bound Algorithm Effectiveness

The branch-and-bound search algorithm finds the best set of intersection manholes. Each of the branch-and-bound variations presented here is designed to search the space of intersection manholes, eliminating inferior solutions as they are identified. This section explores how effectively these algorithms identify these inferior solutions, and thus prune the search. Tests were conducted using

three variations of the branch-and-bound algorithm:

1. Non-enhanced branch-and-bound
2. Branch-and-bound with domination
3. Branch-and-bound with domination and graph reduction

In some instances, the number of nodes searched using the graph reduction enhancement is reported as a sum. This occurs when this enhancement successfully separates the problem into independent sub-graphs. Therefore, a result reported as 10,000 + 500 + 1,000 means that graph reduction separated the instance into three independent sub-graphs, solved by searching 10,000, 500 and 1,000 nodes respectively.

Tests were performed on several neighborhood instances. Both the computation time and the number of nodes searched to reach a solution were recorded. A maximum time allotment of four hours was set for each problem instance. Whenever an algorithm exceeds this allotment, an '*' is recorded in the corresponding table entry.

To appreciate the size of the solution space for each problem, the effort for the branch-and-bound search to enumerate all possible solutions is presented in Figure 5.21.[3]

Neighborhood	Tree Size	Estimated Enumeration Time
B	1.90×10^{11}	236 days
C_{23}	1.5×10^{16}	1.9×10^{7} days
C	8.3×10^{18}	1.0×10^{10} days

Figure 5.21: Full enumeration time and tree size for each test neighborhood.

Figure 5.21 shows that it is hopeless to fully enumerate each potential solution. Even for the medium-sized Neighborhood B, the time estimated to enumerate every single solution is 236 days. For sub-neighborhood C_{23} and its corresponding full neighborhood C, the cost to fully enumerate every solution is well beyond a single lifetime. To generate the best candidate manhole sets, each of the algorithms presented here must effectively prune large portions of the search tree.

[3]The size of the search tree is calculated as $(\sum_{j=1}^{m} p^j) + 1$, where p is the number of candidate manholes per intersection and m is the number of intersections. Time estimates are calculated based on 9,281 nodes being searched per second. This total was calculated from the branch-and-bound search for Neighborhood B with a maximum pipe length of 60. Here 1,937,010 nodes are searched in 208.7 seconds. This is approximately $\frac{1,937,010}{208.7} = 9281$ nodes per second.

	Algorithm		
Max Length	B-and-B	B-and-B + Dominance	B-and-B + Dominance + Graph Reduction
50	61	36	11
60	1,937,010	1,772	83 + 4
70	6,647	689	645 + 4
80	64	48	20 + 13
90	907,974	78,429	20,932
100	1,859	557	245

Figure 5.22: Number of nodes searched for branch-and-bound and enhanced branch-and-bound for Neighborhood B

	Algorithm		
Max Length	B-and-B	B-and-B + Dominance	B-and-B + Dominance + Graph Reduction
50	59.5 + 0.005	59.3 + 0.002	59.6 + 0.000
60	63.9 + 206.4	63.9 + 0.114	63.8 + 0.000
70	66.3 + 0.680	66.2 + 0.080	65.7 + 0.060
80	74.7 + 0.004	74.6 + 0.004	77.0 + 0.000
90	81.4 + 101.6	81.5 + 8.950	81.2 + 2.000
100	83.0 + 0.186	83.8 + 0.689	83.6 + 0.025

Figure 5.23: Pre-processing time + Search time results in seconds for Neighborhood B

Consider the node counts and time results for Neighborhood B, shown in Figures 5.22 and 5.23 respectively. Using the branch-and-bound approach, the best candidate manhole set is identified very quickly. In each instance, the number of nodes searched is an insignificant fraction of the full tree. Solution times measure under one second for each instance, except maximum pipe lengths 60 and 90. Even in these cases, the search time is on the order of hundreds of seconds. Using the domination enhancement, the search times are further reduced for these two cases. For length 60, the time is reduced from 206.4 seconds to 0.114 seconds. Similarly, the time for length 90 is reduced from 101.6 seconds to 8.95 seconds. Adding the sub-graph enhancement along with domination for length 90 further reduces the time from 8.95 to 2.00 seconds.

	Algorithm		
Max Length	B-and-B	B-and-B + Dominance	B-and-B + Dominance + Graph Reduction
50	*	3,343	5+4+2+35
60	*	4,965,539	4,331,685
70	*	*	*
80	121	112	110
90	30	25	22
100	38	25	22

Figure 5.24: Number of nodes searched for branch-and-bound and enhanced branch-and-bound for Neighborhood C_{23}

	Algorithm		
Max Length	B-and-B	B-and-B + Dominance	B-and-B + Dominance + Graph Reduction
50	*	31.7 + 0.390	32.8 +0.001
60	*	33.1 + 577.4	33.3 + 444.0
70	*	*	*
80	36.9 + 0.010	36.6 + 0.020	36.7 + 0.013
90	40.1 + 0.003	39.7 +0.003	39.7 + 0.002
100	43.5 + 0.005	43.0 + 0.003	43.3 + 0.002

Figure 5.25: Pre-Processing time + Search time results in seconds for Neighborhood C_{23}

The branch-and-bound algorithm was also tested on Neighborhood C_{23}, a subset of a real neighborhood containing 23 intersections. The results for these tests are shown in Figures 5.24 and 5.25. As shown in Figure 5.21, the enumeration of every solution for this Neighborhood C_{23} is intractable. However, using the branch-and-bound approach it is possible to calculate solutions in many instances. Furthermore, using the enhanced branch-and-bound algorithms solves more instances in the four hour time window than the non-enhanced branch-and-bound. The non-enhanced branch-and-bound algorithm only finds a solution for three of the six instances, while both enhanced versions of the algorithms solve five instances. One of the instances unsolved by the non-enhanced branch-and-bound algorithm is solved by the enhanced version in less than a second. The other instance solved only by the enhanced branch-and-bound algorithm took less than ten minutes.

These results indicate that the enhanced branch-and-bound algorithms can potentially be used to generate global placements for some real neighborhoods. However, it also appears that some placement instances may prove to be too difficult for the branch-and-bound approach.

The problem instance with a maximum length of 70 in Neighborhood C_{23} provides an interesting example of a difficult problem instance. To identify what causes this problem instance to be difficult, the following test was performed using the fully enhanced branch-and-bound algorithm. To obtain the actual branch-and-bound cost, the search was run to completion. This gave a cost of 67 manholes, calculated in just under seven hours. Each time the branch-and-bound algorithm improved on its previous best solution, the time and cost of this new solution was recorded. It was observed that the search found a solution with 67 manholes almost immediately. This means that the remainder of the search was used to prove that there is no better solution. One reason it took so long to identify this initial solution as a lowest-cost layout is the quality of the lower-bound calculated using the pseudo-code in Figure 5.2. In instances where the solution is found quickly, not much of the tree needs to be searched to verify the minimum cost bound. In this particular instance, the search must delve further into the tree to prove that the initial solution is indeed the minimum cost solution.

Up to this point, the branch-and-bound results presented use a fixed intersection ordering. While instances such as the instance with maximum pipe length 70 of Neighborhood C_{23} may be legitimately difficult, it may also be possible that an unlucky intersection ordering makes this problem more difficult to solve. The following subsection examines the effect intersection ordering has on the search.

5.4.4 Branch-and-Bound and Intersection Ordering

In the previous subsection, experiments showed a high variance in computational time for the branch-and-bound algorithm. Even within the same neighborhood, it was observed that some instances were trivial to solve, while others were still unsolved after four hours. However, each of the solutions in the previous section is based on a single intersection ordering. This subsection explores the effect of intersection ordering on the computation time.

Neighborhood	Length	Solved	Min Time	Avg Time
B	50	242	0.001	3.7
C_{23}	70	33	1.420	289.4

Figure 5.26: Instances solved using branch-and-bound for random intersection ordering

Neighborhood	Length	Solved	Min Time	Avg Time
B	50	250	0.001	0.711
C_{23}	70	141	0.88	90.38

Figure 5.27: Instances solved using domination enhanced branch-and-bound for random intersection ordering

Several problem instances were chosen from the previous section. Each instance was solved for 250 randomly chosen intersection orderings using both the non-enhanced branch-and-bound and the domination enhanced branch-and-bound algorithms. Each of these algorithms is tested using an identical set of random intersection orderings. In addition, the minimum cost bound is initialized using a solution found by the greedy sequential algorithm. From the results in the previous section, it was observed that the algorithm either solved the problem very quickly, or timed out after the four hours of allotted search time. For this reason, each branch-and-bound search was allotted five minutes of computation time. For each test, the number of instances solved in this time frame is recorded. For those searches which did complete, the solution time and number of nodes is presented. Results are presented in Figures 5.26 and 5.27 for the branch-and-bound and domination enhanced branch-and-bound algorithms respectively.

As shown in Figure 5.23, the problem instance for Neighborhood B with maximum pipe length 50 was easily solved using each variation of the branch-and-bound algorithm. In Figure 5.26, it can be seen that this problem instance is not difficult for most intersection orderings. The five minute search limit is able to generate a solution in all but 8 of the 250 orderings. However, the enhanced branch-and-bound algorithm is able to solve the instance for each of the 250 orderings. The fact that these algorithms were able to solve this instance for most orderings suggests that this is an easy instance. It is of interest to determine if similar instances from other medium-sized neighborhoods are also easy. However, more testing is required to generalize the performance of these algorithms for medium-sized neighborhoods.

Consider the difficult instance from Neighborhood C_{23} with maximum pipe length 70. Using the descending degree intersection ordering, this problem instance remained unsolved after four hours of computation. However, it was observed that this instance was solved in under 5 minutes for some random orderings. Using the non-enhanced branch-and-bound this occurred in 33 out 250 cases, while for the enhanced search 141 of the 250 instances were solved. Also, this instance was solved very quickly for some orderings. The minimum time for the non-enhanced branch-and-bound search was 1.42 seconds, while this instance was solved as quickly as 0.88 seconds for the enhanced search. These results indicate that the intersection ordering does affect search performance. These results are promising. They suggest the use of different random intersection orderings as an approach to solve seemingly difficult problem instances.

Further research is required to develop a specific intersection ordering to improve branch-and-bound performance. However, these results can already be used. To avoid bad intersection orderings, a time limit can be set for each branch-and-bound search. If after that time limit, a solution is not found, then the intersection can be randomly permuted and a new search started. In this manner, it is possible to exploit a good intersection ordering without actually knowing how to find such an ordering.

5.5 Overview of Computational Complexity

This section provides a brief complexity analysis for the pre-processing phase and each of the algorithms presented in this chapter. For this analysis let m and n be the number of roads and intersections in the neighborhood respectively, and p the number of candidate manholes per intersection.

First, consider the number of local placements that must be calculated to generate the cached cost map for each candidate manhole pair. If each intersection has p candidates, the number of local placements to be calculated per road is p^2. Thus, there will be exactly mp^2 local placements calculated per neighborhood. The exact complexity for the pre-processing phase depends on the local placement algorithm used. A complexity analysis of the different road placement algorithms is presented in Section 4.5.

In practice, the pre-processing phase scales well to the largest neighborhood tested, Neighborhood C, containing 34 intersections. Each of the tests conducted in this chapter utilized the frontier placement algorithm, one of the more computationally complex local placement algorithms. The highest observed pre-processing time, observed for Neighborhood C and shown in Figure 5.20, is 127.5 seconds. Based on this complexity analysis and the observed times, the pre-processing phase should remain computationally tractable for larger real-world neighborhoods.

The greedy sequential algorithm is computationally simple. To fix the intersection manholes, a single linear sweep through each intersection is required. Recall that for each intersection, a manhole is fixed so as to minimize the cost estimate for each adjacent road. The candidate manhole pair costs are calculated during the pre-processing phase, therefore the greedy sequential algorithm is only required to perform table look ups. Therefore, the cost of the greedy sequential algorithm depends on the number of intersections alone, and is $O(n)$.

In the worst case, the branch-and-bound search will enumerate every possible set of candidate manholes for the graph. If there are n intersections, this means that the size of the search tree will be $(\sum_{j=1}^{n} p^j) + 1$. Therefore, the worst case computational complexity of this algorithm is $O(p^n)$. However, the tests from the previous section indicate that the branch-and-bound algorithm performs much better than the worst case scenario.

Consider the domination enhancement. For each intersection, the domination measure is used to eliminate inferior candidate manholes. Therefore, instead of a fixed value p, each intersection k will expand $z_k \leq p$ candidate manholes. The number of solutions expanded with dominance is $\prod_{k=1}^{n} z_k$. If each intersection contains no dominant candidate manholes, the performance of the dominance enhanced algorithm will be the same as the non-enhanced branch-and-bound algorithm.

The graph reduction enhancement separates the neighborhood into independent sub-graphs. Each of these sub-graphs is solved independently, with complexity $O(p^q), q \leq n$. Therefore, the complexity of the branch-and-bound algorithm using the graph reduction enhancement will be the complexity of the largest sub-graph solved. If no sub-graphs can be found, then this method is equivalent to the non-enhanced branch-and-bound.

5.5.1 Future Work

In this thesis, tests were conducted on a limited number of real neighborhoods. To fully understand the difficulty of the global placement problem, further test results from real neighborhoods are necessary. The following experiments are proposed to further demonstrate the capabilities of the algorithms presented in this chapter.

First, the quality of solutions for each algorithm was measured by comparing its performance to that of other algorithms. As an attempt to compare these results to real world pipe layouts, the performance of each algorithm was compared to a center of the intersection based layout. To further assess the quality of these algorithms, their layouts should be compared with layouts generated by skilled engineers.

The results presented in this section showed a high-level of variation in the computation required for the branch-and-bound algorithm. To better gauge the difficulty of this problem, tests on more real neighborhoods are required. The branch-and-bound algorithm should be applied to a collection of small, medium and large real neighborhoods. Recording solution time statistics about solved instances from each of these categories should give a better idea on the difficulty in generating layouts for real-world neighborhood instances.

Another interesting study involves the identification of difficult problem instances. Even within the same neighborhood, the difficulty of problem instances varied from very easy to difficult. The ability to identify hard instances and what makes them difficult is an interesting area of study. One way to test this is to vary the maximum pipe length for a fixed neighborhood to identify hard instances. Then, properties of the problem which makes these instances hard can be explored. Two such properties that can be explored are the number of dominant candidate manholes, and the similarity of manhole costs between each pair of candidate manholes.

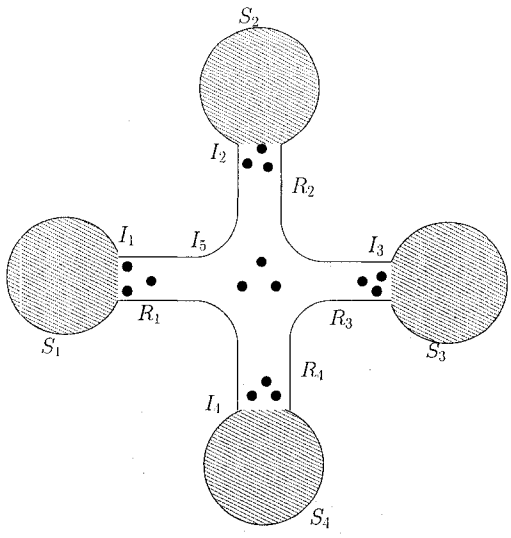

Figure 5.28: Example domination for intersection 0

It was suggested by Martin Müller that if the sewer network is tree-structured, then global placement may be solved using a dynamic programming approach. This suggestion was motivated from the following example, shown in Figure 5.28. In this figure, the filled circles S_1, S_2, S_3 and S_4 represent subsets of the neighborhood. Each of these subsets is connected to the main graph by an intersection, and thus contain a candidate manhole grid. The optimal cost for the neighborhood is calculated in a bottom up fashion. Assume that for S_2, S_3 and S_4 the optimal costs with respect to each candidate manhole in I_2, I_3 and I_4 respectively are known. The optimal cost for each candidate manhole in I_5 is calculated as the costs of roads R_1, R_2, R_3 and R_4, and subsets S_1, S_2, S_3 and S_4. Note that the cost of S_1 is not known at this point in the calculation. Rather, it is computed during the next recursive iteration of the algorithm.

To model global placement as a dynamic program, this cost is represented as a recurrence relation. To calculate the optimal solution, this recurrence relation will need to be expanded at most once for each intersection. Disregarding the effort to compute individual road layouts, the worst case complexity of this algorithm is $O(n)$, where n is the number of intersections in the neighborhood.

5.6 Conclusion

The algorithms presented here automate and reduce the cost for the placement of pipes and manholes for a neighborhood. To simplify this problem, it is assumed that there must be at least one manhole present in each intersection. For each intersection, a discrete collection of candidate manholes is defined. Under this problem definition, a solution is a collection of fixed manholes, where there is exactly one fixed manhole per intersection. The domain of this problem is defined as the combinations of ways to choose a single manhole for each intersection. Given this problem definition, a solution scheme is to transform the continuous problem to a discrete search problem where each

node in the search corresponds to fixing a candidate manhole for an intersection, and the leaf nodes represent a solution where a manhole is fixed for each intersection.

The candidate manhole set for each intersection was chosen in an ad-hoc fashion. The size of the candidate manhole set was chosen based on the largest order intersections, four way. Choosing five candidate manholes for a four-way intersection allows a candidate to be placed close to the opening of each adjacent road, with the final one being in the middle of the intersection. The size of the candidate set was kept the same for three way intersections for consistency. Finally, only a single candidate manhole in the center of the intersection was chosen for cul-de-sacs. This was done since the intersection manhole in a cul-de-sac only affects the placement for one road, thus making it unlikely that further optimization can be achieved by varying the position of this manhole. Therefore, fixing the cul-de-sac manhole at the center of the road decreases the complexity of the problem. The intuition behind the choice of candidate manhole positions is that each provides a representative set of different manhole positions. In other words, these candidate manhole choices are different enough that each is more likely to exploit different manhole placement savings. Defining a more representative candidate manhole set may be possible, and is left as future work.

Two algorithms for the global placement of manholes were presented. Each of these algorithms performed better than random placement, indicating that the position of intersection manholes is important. The branch-and-bound algorithm provides the optimal solution based on the constraints of the candidate manhole grids and the local placement algorithms. However, for large neighborhoods this can quickly become computationally intractable. The greedy sequential approach provides good quality solutions, and can be used to quickly compute global layouts for much larger neighborhoods.

The branch-and-bound algorithms discussed in this chapter showed mixed results. Tests were primarily conducted on a medium-sized real neighborhood with 16 intersections, and a sub-set of a larger real neighborhood with 23 intersections. In some instances, the branch-and-bound approach was able to quickly generate solutions for these neighborhoods. However, for some problem instances with the larger neighborhood, a solution could not be found in the allotted four hour time window. Two enhancements for the branch-and-bound algorithm were introduced, manhole domination and graph reduction. The addition of these enhancements allowed more problem instances to be solved, but there are still instances which remained too difficult to solve. An initial examination into intersection ordering was explored by randomly permuting the order in which the search fixes candidate manholes. It was noted that the ordering of candidate manholes does have an effect on the performance of the algorithm. Further research to determine a good ordering scheme is necessary.

The algorithms presented in this chapter are useful for generating global manhole placements for neighborhoods. In particular, the greedy sequential approach can be used to quickly generate placements. To generate better quality solutions using the branch-and-bound algorithm, the following approach is proposed. First, set a time limit specifying how much computational effort to expend for the solution. Then, calculate a global layout using the greedy sequential approach and use its

cost to initialize a branch-and-bound search. Following this employ the fully enhanced branch-and-bound algorithm to attempt to improve upon this solution. To avoid unlucky intersection orderings, the search can be periodically restarted using a random ordering. If the branch-and-bound search exhausts the desired computation time, simply halt the algorithm and use the best solution found so far.

The sewer system layout problem defined in Chapter 1 specifies that there is a single path from each manhole to the outfall. However, the algorithms presented in this chapter produce a fully connected network. A single path sewer network can be acquired by introducing cuts into the global layout produced by the algorithms presented here. The following chapter introduces some basic algorithms for this and discusses more advanced considerations for this problem.

Chapter 6

Sewer Planner Prototype

In the previous two chapters, algorithms for placing manholes and pipes were developed. However, manhole placement is only one component of a complete sewer system planner. This chapter describes a sewer planning prototype based on an architecture introduced in Chapter 2. This prototype uses algorithms from Chapters 4 and 5, along with a simple edge selection technique and a third party design solver.

This sewer planning prototype is tested using two real-world neighborhoods and one synthetic neighborhood. From these results, the potential of this complete solver architecture is evaluated. Following this is a discussion of the weaknesses of the current system, and considerations for an industrial strength system.

6.1 System Architecture

In Chapter 2, the sewer system planning problem is separated into a group of related sub-tasks organized into the architecture shown in Figure 6.1. Using this architectural model, each of these sub-problems is solved by an independent component. This chapter explores the process of combining these components to produce a complete sewer planning system.

Pre-processing tasks were explored in Chapter 3. The pre-processing component imposes the neighborhood graph structure on the raw line and arc data used to display the neighborhood. Manhole placement specifies the position of manholes and pipes within the two-dimensional neighborhood road map. A manhole placement solver can be implemented using local and global placement algorithms introduced in Chapters 4 and 5. The edge selection and the design components will be discussed in the following two sections. Finally, there is the notion of iteration selection. This component is used to define how a sewer plan can be iteratively modified to improve plan quality. This component is ignored for the prototype presented here; however, implementation considerations are discussed later in the chapter.

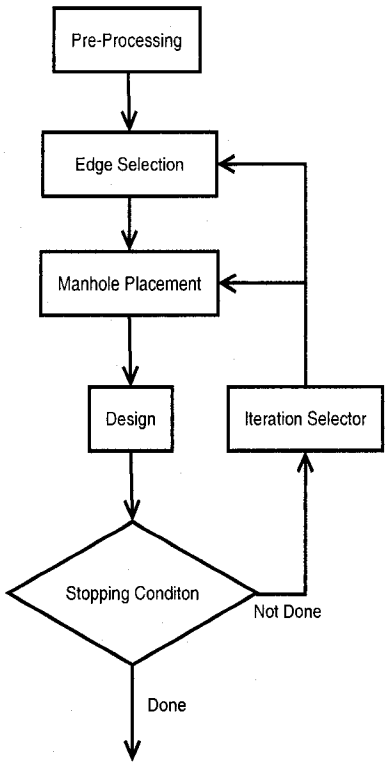

Figure 6.1: Proposed sewer planner architecture

6.2 Edge Selection

Consider the edge selection problem introduced in Chapter 2. Edge selection is defined in terms of the graph $G = (V, E)$, where the vertices represent fixed points and the edges represent potential connections between them. In terms of the neighborhood graph, intersections correspond to vertices and roads correspond to edges. Consider the Li and Matthew edge selection model defined in Chapter 2 [20]. This model separates E into two categories, tree branches and chord branches. The tree branches are the edges belonging to a spanning tree in G, as chosen by an edge selection algorithm. This spanning tree constitutes the main conveyance backbone of the sewer network. The remaining edges are chord edges. These edges are not part of the main backbone, rather they are used to convey local flow. To eliminate loops in the sewer network, a cut is introduced at one extremity of each chord branch.

Several edge selection techniques were presented in Chapter 2. Many of these techniques could be used to implement the edge selection component of this system. For example Tekeli and Belkaya introduce several cost measures and use standard spanning tree algorithms[33]. Walters and Lohbeck [38] and Hassalani and Dandy [16] introduce genetic algorithm approaches to edge selection.

The edge selection algorithm used in this prototype is based on the expanded graph format introduced in Chapter 3. Recall that each spanning tree in the expanded graph specifies both the tree and chord edges, as well as the location of the cut within a chord edge. The flow direction of

the network may also be inferred from this tree. Based on this graph representation, edge selection can be viewed as the problem of finding the spanning tree that represents the best network topology. To achieve this, Dijkstra's Algorithm is used to find the shortest path between the outfall and each other vertex in the graph[9]. Thus, the resulting topology will be the shortest path spanning tree. To use Dijkstra's Algorithm weights must be assigned to each of the edges. For the edge selection implemented for this prototype, each edge in the expanded graph is assigned an edge weight equal to curb length of the corresponding road.

In solving the sewer planning problem, cuts are only permissible at road extremities. Because of this constraint, the prototype calculates the layout as follows. Using the expanded neighborhood graph, edge selection separates the roads into tree and chord branches. Independent of edge selection, manhole placement generates full pipelines for each road. Then, for each road classified as a chord edge, cuts are then introduced into the specified extremity.

If cuts are not constrained to occur at the extremity of a road, then the manhole placement solver will need to consider the cut position when generating pipelines. This topic is discussed further in Section 6.6.

6.3 Design Solver

The design module used in this prototype was developed at the University of Alberta by Andrew Neitsch and Neil Burch. This component models the design as a cost minimization problem, and implements a solver using the CPLEX optimization engine [5]. The design module outputs a triplet *<diameter, upstream height, downstream height>* for each pipe in the sewer network.

To simplify the presentation of the design optimization problem, some terminology and utility functions are first defined. Denote the set of pipes from the layout as P. Let D be the set of commercially available pipe diameters. Let v_{min} and v_{max} be the minimum and maximum flow velocities respectively.

The loss of energy in flow within a pipe depends on the material properties of the pipe. This is represented by the roughness coefficient and will be denoted n in the following problem definition. It is assumed that the roughness coefficient is known; roughness coefficients for different materials are presented in many civil engineering texts such as [23].

Recall that each pair of adjacent pipes meet at a manhole. The downstream end of the source pipe may not buried at the same depth as the upstream end of the destination pipe. Rather, the sewage may enter the manhole at a higher height than it leaves. The difference in height from which the sewage enters and leaves the manhole is referred to as the drop height, or simply the drop.

When referring to pipes, the term invert elevation, or simply invert, is used to describe the deepest portion at a particular point along the pipe. Consider a cross-section of the pipe at the upstream end. The invert elevation of this cross-section refers to the deepest buried point. Conversely, the obvert elevation refers to the shallowest point of the pipe cross-section.

The following list of functions are used to specify simple properties of each pipe. Each of these quantities may be determined directly from the *<diameter, upstream height, downstream height>* values for each pipe.

- $AverageDepth(p)$: Returns the average depth for which this pipe is buried. Since this thesis deals with only straight pipes, this is simply the depth of the middle of the pipe.
- $PerUnitCost(p)$: Returns the per-unit cost of p, where p has a diameter from a set of discretely available pipes. The set of discrete pipe diameters and associated costs are shown in Figure D.1 of Appendix D.
- $Length(p)$: Returns the length of the pipe p.
- $Diameter(p)$: Returns the diameter of p.
- $Slope(p)$: Returns the slope of pipe p.
- $Flow(p)$: Returns the flow value for pipe p.
- $DownstreamInvert(p)$: Returns the invert point of the downstream end of a pipe.
- $DownstreamObvert(p)$: Returns the obvert point the downstream end of a pipe.
- $UpstreamInvert(p)$: Returns the invert point of the upstream end of a pipe.
- $UpstreamObvert(p)$: Returns the obvert point of the upstream end of a pipe.

Considered as a black box, this design solver requires the following input:

1. Available pipe sizes with per unit costs;
2. Flow contributed from each manhole;
3. Layout defining positions of pipes and manholes;
4. Heights of manholes;
5. Minimum drop height for each pipe; and
6. Maximum expected flow within each pipe.

The cost for a design is measured using a quadratic function. This function was empirically fitted to closely approximate costs from a table of real designs.[1] For a single pipe, the cost can be separated into two parts; the actual cost of the pipe and the cost to bury it. The cost of the pipe is calculated as $Length(p) \times PerUnitCost(p)$. The cost to bury the pipe is expressed as:

[1] The real neighborhood costs used to fit this function may not be reproduced in this thesis by request of the firm which provided this data.

$$DepthCost(p) = (2.1 \times AverageDepth(p) + 3.78) \times AverageDepth(p) - 5.88$$

Note that $DepthCost(p)$ is quadratic. However, the goal of this problem formulation is to represent the design problem as a linear program. To be able to use the $DepthCost(p)$ function in such a formulation, it must itself be a linear function. Therefore, a piece-wise linear approximation of this fitted function is used. This approximation will be denoted $DepthCost'(p)$. For this thesis, a linear piece-wise approximation using ten line segments has been used.

Using the depth cost function, the design cost for a pipe p is expressed as:

$$DesignCost(p) = (DepthCost(p) + PerUnitCost(p)) \times Length(p)$$

If the linear approximation $DepthCost$ is used, denote the above function:

$$DesignCost(p)' = (DepthCost(p)' + PerUnitCost(p)) \times Length(p)$$

The terminology and functions just introduced are used to define a mathematical program for the sewer design problem. The mathematical program is defined below, followed by a description of the constraints and variables:

$$\textbf{Min} \sum_{p \in P} c_p$$

Where, for each pipe p the following set of control variables is defined

c_p: The cost of p

p_{up}: The upstream height of the pipe p.

p_{down}: The downstream height of the pipe p.

$X = \{x_{p,d}\}$: A set of binary variables, one for each available pipe diameter.

The following set of constraints is used to ensure the solver generates legal solutions. For each pipe in the sewer network, there will be one instance of each of the following twelve constraints:

1. $c_p \geq DesignCost'(p)$
2. $UpstreamObvert(p) \leq$ ground
3. $DownstreamObvert(p) \leq$ ground
4. upstream manhole depth $\leq UpstreamInvert(p)$
5. downstream manhole depth $\leq DownstreamInvert(p)$

6. $Slope(p) \geq \frac{v_{min} \times n}{(r^{\frac{2}{3}} \times 1.486)^2}$

7. $Slope(p) \leq \frac{v_{max} \times n}{(r^{\frac{2}{3}} \times 1.486)^2}$

8. $Slope(p) \geq (\frac{Flow(p) \times n}{r^{\frac{2}{3}} \times 1.486 \times \Pi \times r^2})^2$

9. $Diameter(p_{up}) \leq Diameter(p_{down})$

10. $DownstreamObvert(p_{up}) \leq UpstreamObvert(p_{down})$

11. $DownstreamInvert(p_{up}) + drop \leq DownStreamInvert(p_{down})$

12. $\sum_{d \in D} x_{p,d} = 1$

Note that the objective function for this minimization problem is a sum of individual cost variables for each pipe; $\sum_{p \in P} c_p$. Minimizing the objective function does not directly minimize the $DesignCost(p)'$ function. Instead, this function is indirectly minimized using the first constraint. This constraint ensures the solution for the mathematical program has pipe cost variables $c_p, p \in P$ such that each c_p is the minimum value that is greater than or equal to the piece-wise linear approximation of $DesignCost(p)'$ and the corresponding design obeys to each of the other constraints.

Constraints 2 and 3 ensure that each pipe p is buried completely underneath ground-level. Constraints 4 and 5 ensure that neither the upstream nor the downstream ends of a pipe are buried deeper than the depths of their adjacent manholes. Using constraints 6 and 7, the flow velocity is kept between self cleansing and pipe damaging velocities. Constraint 8 ensures a steep enough slope to provide the required flow. Constraints 6 through 8 are derived directly from the Manning Equation for flow velocity. Each of these constraints involve a term r, which represents the hydraulic radius of the pipe. The Manning Equation and the hydraulic radius are two quantities that are discussed in most introductory civil engineering texts, such as [23].

Sewage is prevented from flowing from larger to smaller pipes using constraint 9. Uphill flow between pipes is prevented by constraints 10 and 11. Constraint 12 ensures that exactly one of the binary variables in X is set for each pipe, assigning a single diameter per pipe.

The final constraint involves a set of binary variables. Thus, the mathematical program just described is a mixed integer program; not a linear program. Solving a mixed integer program is an NP-Complete problem[13]. Thus, solving this problem may be computationally intractable. Therefore, a variation of this problem that uses continuous relaxation to transform these binary variables into continuous variables is introduced. This new problem is a linear program that may be solved using a polynomial-time algorithm, such as an interior path method [18].

Continuous relaxation allows the binary variable from the initial formulation to assume values in the continuous range [0,1], where the sum for each set of these variables is still constrained to be 1. In this case, the variables in X are considered as weights for their corresponding commercially

available diameter values. Using this technique, the pipe diameter is the weighted sum of each of the commercially available pipe diameters.

To solve the design problem described here, the solver requires an estimate of the flow volumes for the system. However, the actual flow is a function of the design of the system. To overcome this, the solver implements an iterative process that first estimates these flow values, then generates a design based on this estimate. This iterative process is defined as follows:

1. Initialize the flow values with a conservative overestimate.
2. If current set of flow values are within a threshold of another set of flow values for which a solution has already been generated, then return the lowest cost solution.
3. Solve for pipe depths and sizes.
4. Recalculate new maximum expected flows.
5. If current network does not support the flow, update the flow using the average of the current and last valid flows. Return to step 2.
6. If the flow is supported but the cost is higher than the previous design, then return the previous design.
7. If the solution is valid and of lower cost than the previous best, then store the flow and cost. Return to step 2.

By using an overestimate in the first phase, it is guaranteed that this process generates a sewer plan that can handle the expected flows within the system. Based on each new design, the estimated flow values are revised and the solver attempts to generate a better solution. If during this iterative process an illegal solution is generated, then a new solution is generated using the average of the current flows and that of the last solution. The iterative process may terminate in one of two ways. First, it stops if during step 2, the averaged flows are within a threshold of the flows from a previous solution. Otherwise, if a newly generated valid solution is higher in cost than the previous valid solution, the process terminates with the lowest cost valid solution.

It is worth noting the difference in solutions produced by the two solver modes. The original solver will generate an optimal design using legal pipe diameters. However, the relaxed mode generates solutions in a continuous range. It is likely that these pipe sizes will not match the commercially available sizes. Therefore, the design from the continuous relaxation solver will need to be modified to use real commercial pipe sizes. For instance, pipe diameters will need to be increased to the nearest larger commercial pipe size. However, increasing pipe size alone is not enough to generate a valid solution. In some cases, it may be necessary to adjust the pipe depth so that this new solution does not break constraints from the mathematical program. Consider a pipe, p, from the continuous relaxation, and its corresponding pipe, p', adjusted to be of a commercially available

size. The depth of p' may need to be adjusted for several reasons. For example, the larger p' may no longer be buried completely below ground. In this case, the depth of the pipe, and all corresponding downstream pipes, will need to be increased. In other instances, the invert point of p' may be buried deeper than the manhole depth requiring the pipe to be raised.

An algorithm to adjust these continuous solutions to valid real-world designs is needed. This transformation process is left as future work. To validate the sewer planning architecture, the continuous relaxation solver has been used. To ensure that real pipe sizes are used, the pipe size is increased to the closest commercially available size. It is acknowledged that this approach does not guarantee legal designs. However, this approach provides a design approximation process that can be used to evaluate the architecture as a whole.

Using the architecture defined in Figure 6.1, the pipe layout algorithms from Chapters 4 and 5 are combined with the design solver to generate a complete sewer system planner. Results for this planner are discussed in the next section.

6.4 Algorithm Testing

This planner has been tested using real and synthetic data. However, it has been extremely difficult obtaining real-world data representing the input volume collected at each manhole. Thus, the sewer systems produced are not generated for the expected flow of the real-world neighborhood. For the purpose of these tests, a constant flow volume is added to the network at each manhole. It is acknowledged that these synthetic flow values may not be truly representative of real flows. Despite this, it is anticipated these techniques would work well with real flow data. The tests presented here can still be used to evaluate the architecture of the prototype. The machine used to perform these tests has dual 2 GHz AMD Athlon processors and 1 GB of memory. Tests were conducted on three neighborhoods. Neighborhoods B and C, shown in Figures 5.12 and 5.13 of Chapter 5 respectively as well as synthetic neighborhood D, shown in Figure 6.2.

6.4.1 Solver Computation Times

This chapter discusses the two implementations of a design solver; discrete and continuous relaxation. The computation time required for each implementation to generate sewer designs is explored in this section. Tests are conducted on each of the three test neighborhoods. For each test neighborhood, edge selections are computed from thirty-five random spanning trees. Both the discrete and continuous relaxation solvers are used to generate a solution for each spanning tree, and the time to generate each design is recorded. A half-hour computational time limit is placed on each design. For test instances where every design computation failed to complete after thirty minutes, an '*' is recorded in the appropriate table column. The minimum, maximum and average time statistics are calculated using only the instances which completed in the allotted time period. Timing results for the discrete and continuous relaxation solvers are presented in Figures 6.3 and 6.4 respectively.

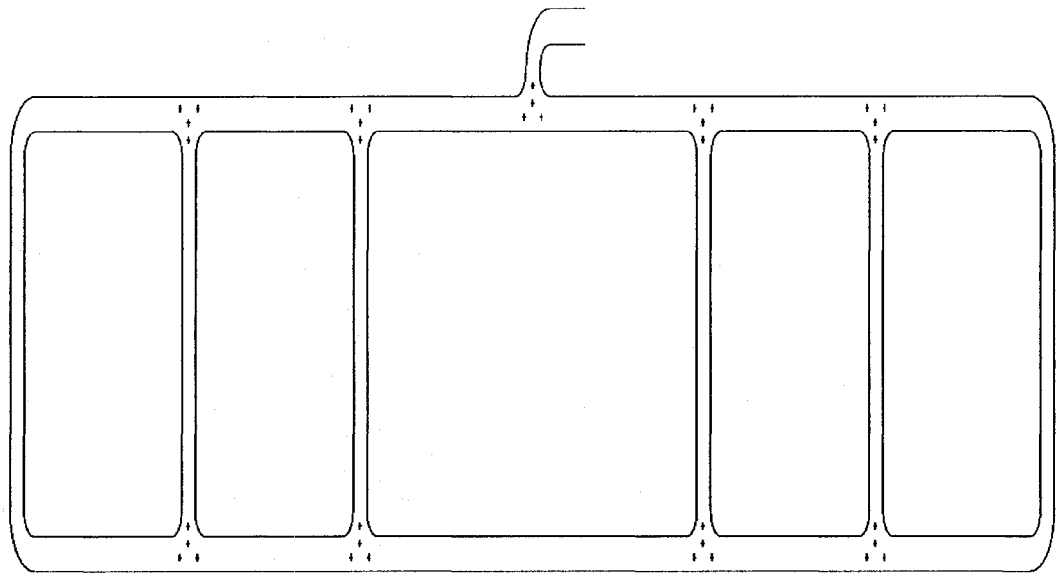

Figure 6.2: Test Neighborhood D

Neighborhood	Minimum Time	Maximum Time	Average Time	Unsolved
B	75.5	1,775.4	734.4	5
C	*	*	*	35
D	8.2	1,604.3	551.9	2

Figure 6.3: Discrete solver solution times in seconds

Neighborhood	Minimum Time	Maximum Time	Average Time	Unsolved
B	2.9	6.2	4.6	0
C	5.2	15.7	12.1	0
D	5.3	10.0	8.3	0

Figure 6.4: Relaxed solver solution times in seconds

As expected, these results exhibit that the computational costs for the discrete solver are more expensive than the continuous relaxation version. Consider the results for the discrete solver, shown in Figure 6.3. For neighborhoods B and D, the average solution times are 734.4 and 551.9 seconds respectively. However, the maximum solution times for both neighborhoods are considerably higher at 1,775.4 and 1,604.3 seconds respectively. Additionally, neighborhood B had five problem instances unsolved after the thirty minute time limit, while neighborhood D had two unsolved. For large-sized real world neighborhood C, the discrete solver was not successful at solving any of the thirty-five problem instances. The continuous relaxation solver generates designs much quicker. For each of the three neighborhoods, all thirty-five instances were solved, with a maximum solution time of 15.7 seconds.

These results indicate that the discrete design solver is only computationally feasible for designing small to medium neighborhoods. Even in this case, it appears that there are problem instances for such neighborhoods that may not be computationally feasible. However, further testing is required to confirm this claim. These results also suggest that the discrete solver is not feasible within the design architecture presented in Figure 6.1. When multiple layouts need to be considered in the planning process, the discrete design solver is too computationally expensive. Thus, for the remainder of the tests presented in this chapter, the continuous relaxation version of the design solver has been used.

6.4.2 Solution Quality

This subsection examines the quality of sewer plans generated using the shortest path spanning tree for edge selection. For synthetic neighborhood D, the shortest path spanning tree cost is compared to each other potential solution. For neighborhoods B and C_{23} the cost of the shortest path spanning tree sewer plan is compared to the sewer plan cost from 20,000 randomly generated edge selections.

Neighborhood costs are measured in dollars using the following function:

$$PlanCost = \sum_{p \in P} DesignCost(p) + 10,000 \times m$$

where $DesignCost(p)$ is the cost for pipes as presented in Section 6.3, and m represents the number of manholes in the layout. Note that the second term assumes a value of \$10,000 for each manhole in the layout. While this is a reasonable manhole installation cost, the actual cost depends on factors such as the diameter and depth of the manhole.

The sewer plans presented here were generated using the continuous relaxation version of the design solver. For each design, pipe sizes were increased to the next largest commercially available pipe diameter. It is noted that this practice may cause the solution to be in violation of the constraints from the mathematical program. Therefore, the sewer plan costs presented here should be taken as an approximation. These approximations are used to evaluate the cost of sewer plans with respect to the underlying pipeline layout.

Tests were conducted on Neighborhood D, a synthetic neighborhood shown in Figure 6.2. The outfall in this neighborhood is located at the dead-end of the top curved road section. This neighborhood was chosen because it represents a shallow, but wide, network. Consider the intersections represented at the bottom of this neighborhood. Intuitively, the most cost effective network should route sewage from the bottom part of the neighborhood to the top part as directly as possible. That is, the sewage should pass through at most one intersection at the bottom on its route to the top. Conversely, the worst solution should be one that conveys sewage around the whole network before finally depositing it at the outfall point.

To test Neighborhood D, a sewer system plan was generated for each of the 6,944 different edge selection outcomes. Each sewer plan is based on a layout generated by the enhanced branch-and-bound search, using frontier placement to generate layouts for each road.

	Cost Type			
Max Pipe Length	Least Cost	Highest Cost	Average Cost	Shortest Path Spanning Tree
50	$5,119,450	$13,594,700	$6,546,730	$5,124,840
60	$4,598,080	$11,896,700	$5,788,650	$4,625,020
70	$4,242,640	$11,139,100	$5,451,140	$4,310,810
80	$3,839,560	$8,438,010	$4,865,800	$3,852,350
90	$3,527,620	$8,133,060	$4,604,820	$3,660,190
100	$3,523,140	$8,089,650	$4,557,560	$3,537,650

Figure 6.5: Sewer plan costs for neighborhood D

The cost results for these plans are summarized in Figure 6.5. From each of these spanning trees, the minimum, maximum and average plan costs are presented. In addition, the cost of the plan corresponding to the shortest path spanning tree is shown. These results indicate that the high-level network topology affects the cost of the system. With a maximum pipe length of 50, the most expensive sewer plan is approximately 2.6 times more expensive than the cheapest plan. In addition, the plan corresponding to the shortest path spanning tree is a mere 0.1% more expensive than the optimal solution.

The total distance sewage is conveyed has a big effect on the overall cost of the plan. Figures 6.6 and 6.7 show the cheapest and most expensive plans for Neighborhood D. Examining the cheapest neighborhood, sewage is conveyed to the outfall as directly as possible. In contrast, for the most expensive plan the sewage is carried along an indirect path through the network. This suggests that the sewer plan corresponding to the shortest path spanning tree should be of good quality. Evidence to support this claim is presented in column entitled *Shortest Path Spanning Tree* in Figure 6.5.

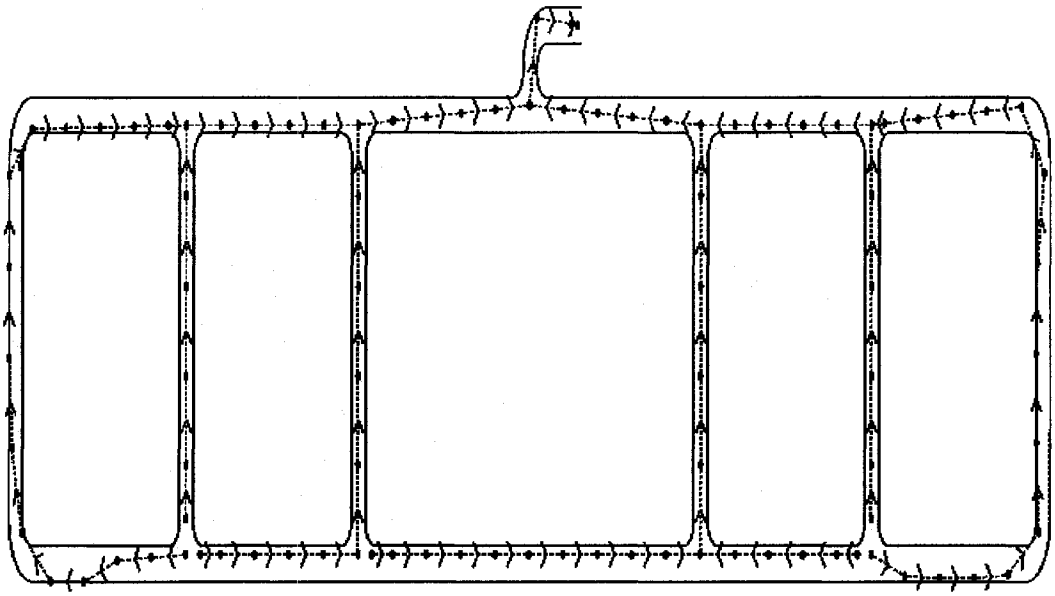

Figure 6.6: Minimum cost network topology for Neighborhood D

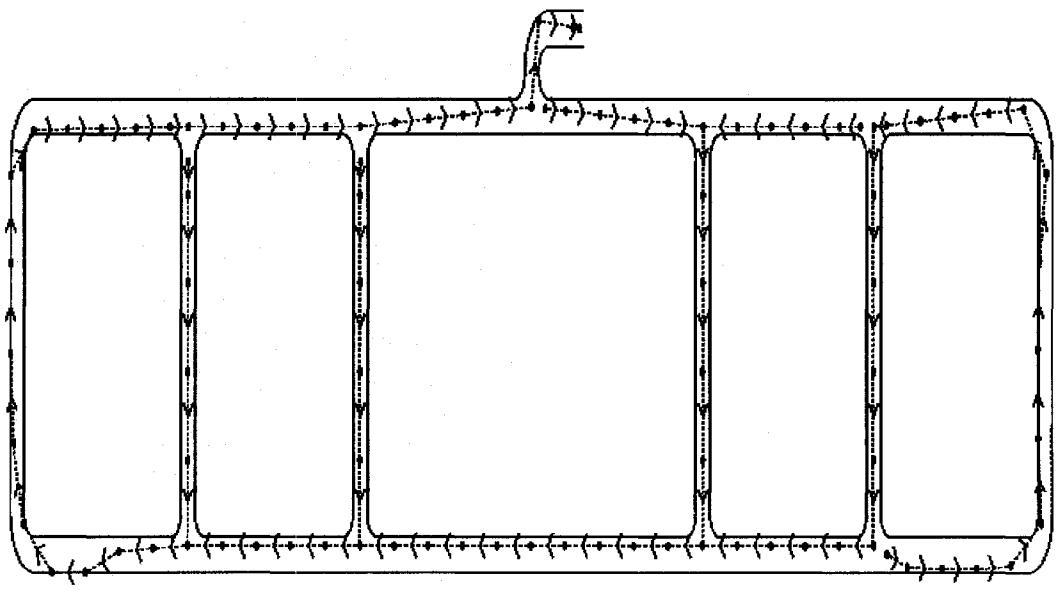

Figure 6.7: Maximum cost network topology for Neighborhood D

	Cost Type			
Max Pipe Length	Least Cost	Highest Cost	Average Cost	Shortest Path Spanning Tree
70	$2,249,170	$3,760,680	$2,798,350	$2,299,780

Figure 6.8: Sewer plan costs for Neighborhood B

	Cost Type			
Max Pipe Length	Least Cost	Highest Cost	Average Cost	Shortest Path Spanning Tree
70	$3,923,240	$5,701,500	$4,379,120	$3,978,690

Figure 6.9: Sewer plan costs for Neighborhood C

The quality of the shortest spanning tree sewer plan was also evaluated for real Neighborhoods B and C. For these neighborhoods, the entire set of spanning trees is too large to fully enumerate. Instead, for each neighborhood the shortest path spanning tree design is compared to the designs of 20,000 randomly generated spanning trees. For each neighborhood, tests were conducted with a maximum pipe length of 70. For both neighborhoods, the outfall was randomly placed at an entrance and fixed for the duration of the tests. The global layout was calculated using the greedy sequential technique, while the frontier method was used for local placement. Results for these tests are presented in Figures 6.8 and 6.9.

In these tests, the shortest path spanning tree generates good quality solutions. For Neighborhood B, the shortest path spanning tree plan is cheaper than 99.97% of the randomly sampled spanning trees. Similarly, for Neighborhood C, this plan is cheaper than 99.98% of the random samples. In addition, the shortest path spanning tree costs are within 2.2% and 1.4% from the cost of the best observed solutions for Neighborhoods B and C respectively. The shortest path spanning tree plan is observed to be 17.8% better than the average solution cost for Neighborhood B and 9.1% better than the average solution cost for Neighborhood C.

Results from these three test neighborhoods indicate that the shortest path spanning tree approach works well for edge selection. In every test case, the cost of the shortest path spanning tree was within 2.5% of the cheapest observed solution. For even medium-sized neighborhoods, such as Neighborhood B, the cost to enumerate and generate a design for every spanning tree is not computationally feasible. However, these results suggest that the minimum path spanning tree design has cost close to the optimal solution.

6.5 Discussion

The results presented in this section suggest that the local and global placement algorithms from Chapters 4 and 5 may be used as part of a complete sewer planning system. Several observations can be made from the results observed. First, note that the frontier placement technique has been used for each test case. From this it can be concluded that the use of sophisticated local placement techniques, such as the frontier placement algorithm, are feasible within the sewer planning process.

However, the use of the branch-and-bound global placement algorithm is not feasible for larger neighborhoods. As seen in Chapter 5, the application of the enhanced branch-and-bound algorithm for larger neighborhoods, such as Neighborhood C, was not computationally feasible. However, the use of the greedy sequential algorithm is still expected to generate good quality global layouts. Therefore, the greedy sequential algorithm should be suitable for use in the complete sewer planning system.

The shortest path spanning tree edge selection approach presented is simple, yet effective. The cost of sewer plans generated using this approach was within 2.5% of the cost of the best observed solution in every test case. Using this approach, a design only needs to be generated for a single spanning tree. Furthermore, this spanning tree can be generated quickly. For example, Dijkstra's Algorithm can be used to find the shortest path spanning tree with a worst case time complexity of $O(n^2)$, where n is the number of nodes in the expanded graph structure[9].

The key missing element from this system are realistic flow values. If these values can be easily supplied as input to the planning process, then this architecture can be used to efficiently generate a high-quality complete sewer system plan. If these flow values need to be calculated based on the automatically generated layout, then this computation time will have to be reflected in the evaluation of this system.

6.6 Future Considerations

In this chapter, a primitive sewer planning prototype was explored. However, there are more issues that can be addressed to potentially improve the system.

In the architecture shown in Figure 6.1, an iterative improvement scheme is proposed. In general terms, this process iteratively perturbs the layout, solving for the design for each new layout. However, the proposed architecture separates the layout problem into two components, edge selection and manhole placement. Perturbing the high-level network topology or the precise position of manholes will both change the cost of the overall sewer plan. A specific planning system must define how to modify the layout. In addition, it may be reasonable to modify the high-level topology sometimes, and the low-level detail of manhole and pipe positions other times. In Figure 6.1, the unit labeled iteration selector chooses how the layout will be modified. The optimal policy here is likely a combination of changes on both levels, a decision which is made by the iteration selector unit. Additional research is necessary to determine the effect of changes to different levels of the layout.

The edge selection algorithm implemented for this prototype constrains cuts to occur at the extremities of a road. However, in real neighborhoods, the optimal position of a cut may lie elsewhere within the road corridor. Often, the elevation at each point in the road may influence where the cut should be located. Consider the task of placing the pipeline for a storm sewer in a road corridor, and let the highest point of elevation lie at a point p. In such a case, it may be natural to locate the cut

at this point p. To generate the pipeline for this road, the manhole placement solver must place two pipelines based on this cut position; one between the first intersection and p, and another between the second intersection and p. An industrial strength sewer planner needs to consider elevation information to optimize sewer plan quality.

The system presented here assumes that the flow into the system is known. However this quantity depends on the specific network layout of the system. A fully-automated system needs to include techniques to calculate this flow. Flow estimates can be used, but these will change each time the layout changes. This system can be augmented with the addition of a module that calculates the flow from the chosen layout, and drainage information for the neighborhood.

An industrial strength sewer planning system requires a design solver that generates sewer plans using commercial pipe sizes. The design module used to implement this prototype generates pipes within a continuous size range bounded by commercial pipe sizes. For an industrial system, it is necessary to transform this design into one that uses commercial pipe sizes and obeys the constraints of the mathematical program. Since this solver generates designs quickly, it would be desirable to use it in an industrial strength sewer planning system. However, to be able to use this module, an efficient algorithm for converting these designs into valid real-world sewer designs is necessary. One area of future work involves the development of such a conversion algorithm. If no algorithm can be found, then an alternate design approach must be developed.

A shortest path spanning tree approach based on the cost estimates for each pipe was used to implement an edge selection algorithm for the test system. Results show that this approach produces good quality solutions, but not necessarily the best quality solutions. Any of the edge selection algorithms covered in the Chapter 2 survey could be used to implement edge selection. An area for further research may involve the computation of optimal spanning trees for the design phase.

6.7 Conclusion

This chapter explores the implementation of a complete planning system based on the architecture proposed in Chapter 2. Manhole placement was conducted using the local and global placement techniques introduced in this thesis. These algorithms were used in cooperation with shortest path spanning tree based edge selection, and a third party design module. The result is a system capable of defining both the position of pipes and manholes as well as the depth, diameter and slope of each pipe.

Tests on synthetic data indicate this architecture can be used to effectively automate the planning stages for sewer systems. In addition to the global and local techniques introduced earlier, this chapter proposes a simple edge selection technique. This technique simply fixes the backbone network as the shortest path spanning tree, where the cost of each edge is simply the length of the road. To evaluate the effectiveness of this technique, a full enumeration of every spanning tree for a test neighborhood was done. These experiments have shown that the quality of the shortest path

spanning tree performs well.

The tests presented here indicate that this architecture provides an efficient technique for planning sewer systems. These techniques need to be more thoroughly tested using real neighborhood data. The real data available for testing in this thesis was limited, and often incomplete. While the tests presented here still show the fitness of the proposed techniques, real-world data may provide insights that can be used to improve these results.

A better technique for dealing with the flow rate through the system is required. For the design module to produce the best solution, accurate flow information is required. This flow is based on the infiltration to the system, which is based on the sewage properties of the neighborhood, and the specific layout produced during the manhole placement phase. This can be developed as a separate module, and integrated into the proposed architecture.

The techniques presented here provide a solid basis for a complete sewer system planner. The prototype is not an industrial strength planner, but it does address some key issues for such a system.

Chapter 7

Conclusion and Limitations

This thesis explores the sewer system planning problem from a computing science viewpoint. Several sewer planning sub-problems are modeled as computation problems, and algorithms to solve these sub-problems are explored. The goal is to provide the engineering research community with a suite of algorithms to be used in a real-world sewer planning system.

The sewer planning process accepts as input the neighborhood road map, a set of available pipe sizes and hydrological data. The goal is to generate a sewer plan that can successfully convey waste from a set of pre-defined clients to an outfall point. Such a plan defines the position of pipes and manholes, in addition to the diameter and upstream and downstream heights for each manhole in the system. For this thesis, it is assumed that potential clients may lie at any point adjacent to a road. To provide full service, pipelines are laid completely down the corridor of each road. In addition, there may be only one path from each manhole to the outfall.

There are two important benefits to an automated planning system. First, automating the planning process relieves the engineer of the tedious manual planning task. This allows the highly-skilled expertise of the engineer to be applied to other planning problems. Second, this system can be used to reduce the cost of the sewer system plan. This will allow contractors to tender a lower cost estimate, increasing the chances of a successful bid.

Sewer system planning is a complex optimization problem. Standard practice separates the planning process into two parts; design and layout. The design defines the properties of the pipes, including the pipe diameter and upstream and downstream invert pipe elevations. A survey of design techniques is presented in Chapter 2. Sewer system layout is further divided into two parts, edge selection and manhole placement. Edge selection is concerned with the general topology of the sewer network. Edge selection begins with a graph, where the vertices define fixed points in the sewer network and the edges represent connections between them. Edge selection determines which edges in this graph belong to the core network, and which edges are secondary pipelines conveying only local flow. An overview of existing techniques for this problem is presented in Chapter 2. Each edge in the aforementioned graph specifies a complete pipeline composed of individual pipes and manholes. Manhole placement determines the position of each pipe and manhole. This level of

detail is often overlooked in the literature, and is the key focus of the research presented within this thesis.

Manhole placement is addressed by two categories of algorithms. Local placement specifies the position of pipes and manholes between two fixed points. Global placement algorithms reduce the global cost of pipelines by varying the position of fixed manholes. Each global placement algorithm utilizes local placement techniques to generate layouts between fixed points.

To minimize cost, local placement algorithms reduce the number of manholes in the pipeline. Three placement techniques were examined. The centerline and curbpoint approaches guide placement using a set of control points. Pipes are placed in an iterative fashion, where the furthest reachable control point from the last placed manhole guides placement for the current iteration. The centerline approach identifies control points in the center of the road, guiding the search toward the center of the road corridor. The curbpoint approach places control points directly on the curbs of the road. The goal of this approach is to have the pipeline cut corners, further reducing the number of manholes. The final local placement approach uses a reachability frontier to find the layout with the minimum number of manholes. This algorithm defines an iterative process such that each iteration generates a frontier defining the furthest reachable points. When the frontier passes the goal point, a layout can be traced backward through each frontier. The implementation of this frontier method uses a discrete approximation of a continuous approach proved to generate the minimum number of manholes between two points. Each local technique was tested on real and synthetic roads. The results show that the frontier approach is the most effective at reducing the number of manholes, dominating each of the other placement techniques. In addition, the frontier approach exhibits reasonable computation time, requiring less than one second of computation for pipelines with up to 24 manholes. To optimize the placement of pipelines between two points, the frontier approach should always be chosen.

To further minimize layout costs, global placement algorithms are presented. These algorithms assume the existence of a fixed point within each intersection of the neighborhood graph. Then, for each intersection a discrete sized grid of candidate manholes is defined. By varying the position of these intersection manholes, these algorithms minimize the number of manholes over each pipeline in the neighborhood. Two global placement techniques were introduced. The greedy sequential algorithm uses a local optimization heuristic to approximate the optimal set of fixed manholes. This approach assigns a fixed manhole to an intersection based on a cost heuristic defined as the minimum number of manholes for each directly adjacent road. This simple heuristic is often able to identity the lowest cost solution for the set of candidate manholes.

A branch-and-bound search is employed to find the best set of fixed manholes. This algorithm traverses the solution space, only expanding solutions that can decrease the cost. Each node in the search corresponds to fixing a single manhole for an intersection. For partial solutions, costs for individual roads are estimated as the lowest number of manholes in the pipeline, based on the potential

endpoints of the pipeline. The search space for this problem grows quickly as either the size of the candidate manhole set or the number of intersections increases. Therefore, several search enhancements have been proposed. The first employs a domination measure to eliminate provably inferior candidate manholes. The second identifies roads where the cost is not affected by the candidate manhole position, removing these roads from the optimization problem.

For small to medium neighborhoods, branch-and-bound is the algorithm of choice. However, as the size of neighborhoods increases, the size of the search space increases quickly and the branch-and-bound approach becomes intractable. In these cases the greedy sequential ordering approach can be used to find an estimate for the optimal global placement. Results indicate that the quality of layout for the greedy sequential approach is near optimal, as the number of manholes in the resulting layout is at most a few manholes from the branch-and-bound solution.

In the literature, the standard planning approach is to alternately apply design and layout techniques, iteratively improving the quality of the sewer plan. A new architecture that incorporates the more detailed manhole placement phase is proposed in this thesis. Chapter 6 describes the implementation of a simple planning system incorporating the manhole placement techniques introduced in this thesis. This system automates the planning process, using only the neighborhood rendering data, expected sewage loads, and a set of commercially available pipes sizes.

To adapt the work presented in this thesis to the real-world, there are several limitations to be addressed. One limitation involves the maximum distance between manholes for local placement. Local placement assumes a single maximum pipe length. In reality the maximum distance depends on the diameter of the pipe. In addition, there are sometimes instances where curved pipes may be used. To generate the lowest cost pipeline, these algorithms need to consider these additional pipe properties.

The edge selection and manhole placement algorithms do not consider ground elevation. The addition of elevation information may be exploited to further reduce the cost of a sewer plan. For example, the presence of a high-point within a road may affect the position and size of a cut for a chord edge. The algorithms presented here need to be improved to exploit elevation information.

The edge selection algorithm presented in this thesis assumes that cuts within the chord edges will occur at either extremity of the edge. However, this may not produce the best network topology.

For the algorithms presented in this thesis, the only costs considered were infrastructure and construction costs. However, another important category of costs to consider are maintenance costs. A sewer plan that is inexpensive with respect to infrastructure and construction may be undesirable because of high maintenance costs. A robust sewer system planner should address maintenance costs in the edge selection, manhole placement and design solvers.

This thesis makes several contributions to the planning problem. First, this work explores how data used to render the neighborhood can be grouped as intersection and neighborhood components and arranged as a neighborhood graph. This neighborhood graph can be used to automatically

generate the graph used in the edge selection phase.

Second this thesis considers a finer level of optimization by specifying the exact position of pipes and manholes within a pipeline. Local placement algorithms can be used to minimize the number of manholes between two fixed points, while global placement reduces the number of manholes over an entire neighborhood. While these techniques were presented with sewer systems in mind, it is expected that many of the ideas discussed here are applicable to other types of pipe networks, such as water distribution or irrigation networks.

Finally, a modified sewer system solver architecture using the algorithms presented in this thesis is proposed. This architecture is validated by the implementation of a simple prototype system. While there is more work to be done for the implementation of a fully-automated commercial strength sewer planning system, this thesis provides a number of key algorithms that may be used in such a system.

Bibliography

[1] http://www.haestadmethods.com/software/pondpack/.

[2] http://www.haestad.com/software/stmcstandalone/.

[3] http://www.pizer.com/hydra.html.

[4] http://www.vrand.com.

[5] http://www.ilog.com/products/cplex/.

[6] M. H. Afshar, M. Akbari, and M. A. Marino Hon.M.ASCE. Simultaneous layout and size optimization of water distribution networks: Engineering approach. *Journal of Infrastructure Systems*, 11(4):221–230, December 2005.

[7] C. Charalambous and A. A. Elimam. Heuristic design of sewer networks. *Journal of Environmental Engineering*, 116(6):1181–1199, November/December 1990.

[8] K. Chau and C.S. Cheung. Knowledge representation on design of storm drainage system. In *Innovations in Applied Artificial Intelligence*, number 3029 in Lecture Notes in Computer Science, pages 886–894. Springer-Verlag, 2004.

[9] T. H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[10] J. S. Dajani, Y. Hasit, and S. D. McCullers. Mathematical programming in sewer network design. *Engineering Optimization*, 3:27–35, 1977.

[11] A. F. Diogo and V. M. Graveto. Optimal layout of sewer systems: A deterministic versus a stochastic model. *Journal of Hydraulic Engineering*, 132(9):927–943, September 2006.

[12] A. F. Diogo, G. A. Walters, E. Ribeiro de Sousa, and V. M. Graveto. Three-dimensional optimization of urban drainage systems. *Computer-Aided Civil and Infrastructure Engineering*, 15:409–42, 2000.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[14] Z. W. Geem, T. G. Kim, and J. H. Kim. Optimal layout of pipe networks using harmony search. In *Proceedings of the Fourth International Conference on HydroScience and Engineering*, 2000.

[15] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Publishing Company Inc, 1989.

[16] A. M. Hassanli and G. C. Dandy. Optimal layout and hydraulic design of branched networks using genetic algorithms. *Applied Engineering in Agriculture*, 21(1):55–62, 2005.

[17] M. Heidari, V.T. Chow, P. V. Kokotovic, and D. D. Meredith. Discrete differential dynamic programming approach to water resources systems optimization. *Water Resources Research*, 7(2):273–283, 1971.

[18] B. Kolman and R. E. Beck. *Elementary Linear Programming with Applications*. Academic Press, 1995.

[19] R. P. Lejano. Optimizing the layout and design of branched pipeline water distributions systems. *Irrigation and Drainage Systems*, 20:125–137, 2006.

[20] G. Li and R. G. S. Matthew. New approach for optimization of urban drainage systems. *Journal of Environmental Engineering*, 116(5):927–944, September/October 1990.

[21] G. Y. Li. The optimal design of sewer networks by DDDP. *China Water Supply and Sewerage*, 1986.

[22] L.B. Merritt and R.H. Brogan. Computer-based optimal design of sewer systems. *Journal Of The Environmental Engineering Division*, 99(EE1):35–53, February 1973.

[23] Haestad Methods and S. R. Durrans. *Stormwater Conveyance Modeling and Design*. Haestad Press, first edition, 2003.

[24] D. W. Patterson. *Artificial Intelligence and Expert Systems*, chapter 9, pages 181–182. Prentice Hall, 1990.

[25] T. D. Prasad and N. Park. Multiobjective genetic algorithms for design of water distribution networks. *Journal of Water Resources Planning and Management*, pages 73–82, January/February 2004.

[26] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[27] D. A. Savic and G. A. Walters. Genetic operators and constraint handling for pipe network optimization. In T. C. Fogarty, editor, *Evolutionary Computing*, number 933 in Lecture Notes in Computer Science, pages 154–165. Springer Verlag, 1995.

[28] A. R. Simpson, G. C. Dandy, and L. J. Murphy. Genetic algorithms compared to other techniques for pipe optimization. *Journal of Water Resources Planning Management*, 120(4):423–443, July/August 1994.

[29] D. K. Smith and G. A. Walters. An evolutionary approach for finding optimal trees in undirected networks. *European Journal of Operations Research*, 120:593–602, 2000.

[30] P. R. Swamee. Design of a sewer line. *Journal of Environmental Engineering*, pages 776–781, September 2001.

[31] S. A. Taher and J.W. Labadie. Optimal design of water-distribution networks using GIS. *Journal of Water Resources Management and Planning*, 122(4):301–311, July/August 1996.

[32] S. Tekeli. Computerized design of sewer networks. *Journal of Environmental News*, 9:7–17, December 1981.

[33] S. Tekeli and H. Belkaya. Computerized layout generation for sanitary sewers. *Journal of Water Resources and Management*, 112(4):500–515, October 1986.

[34] E. Todini. Looped water distribution networks design using a resilience index based heuristic approach. *Urban Water*, 2:115–122, 2000.

[35] K. Vairavamoorthy and M. Ali. Optimal design of water distribution systems using genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*, 15:374–382, 2000.

[36] J. P. Velon. Sewer cost estimation model - an application. Master's thesis, Northwestern University, 1971.

[37] G. A. Walters. The design of the optimal layout for a sewer network. *Engineering Optimization*, 9:37–50, 1985.

[38] G. A. Walters and T. Lohbeck. Optimal layout of tree networks using genetic algorithms. *Engineering Optimization*, 22:27–48, 1993.

[39] Water Pollution Control Federation. *Design and Construction of Sanitary and Storm Sewers*, 1970.

Appendix A

Frontier Placement Optimality

In this section, the mathematical definitions of the reachability frontier are introduced. The term frontier is used to define the furthest reachable points from a set of base points. The frontier manhole placement method uses the idea of the frontier to place pipelines between two points, propagating reachability frontiers until the endpoint can be reached. It is claimed that this method generates placements with the minimum number of manholes. A proof of this claim is provided in this section.

Consider a simple polygon[1], Q, with two points, p_0 and p_n, inside of Q. The goal is to generate a path between the two points using the minimum number of line segments, with the restriction that each line segment may be no longer than a maximum distance d. Since both p_0 and p_n lie within a simple polygon, such a path exists. Denote $path^*$ as a path with the minimum number of line segments, which is represented by the sequence $path^* =< p_0, p_1, p_2, \ldots, p_n >$.

A point p_2 is d-reachable from p_1 if and only if $|\overline{p_1 p_2}| \leq d$ and $\overline{p_1 p_2}$ lies totally inside of Q, where $|\ldots|$ defines length. A point p_n is d-reachable from p_1 using n line segments if and only if the points may be connected by a path using n line segments, such that adjacent points in the path are d-reachable from one another.

Consider a set of points $\mathbf{P}$. The set of all points d-reachable from at least one $p \in \mathbf{P}$ is defined as the interior of $\mathbf{P}$, denoted $Int(\mathbf{P})$. The interior for a single point p, $Int(p)$, is generated in the following manner. Using p as an endpoint, sweep a line segment of length d around a full circle. The set of d-reachable points inside of the swept circle belong to the interior. The interior for a set of points is simply the union of the interior of each point in the set.

The set of the furthest reachable points from a set $\mathbf{P}$ will be defined as the Frontier of $\mathbf{P}$, denoted $Front(\mathbf{P})$. Once again, consider the generation of a frontier from a single point. To generate $Front(p)$, use p as an endpoint and sweep a line segment of length d around a full circle. For each angle in the circle the frontier point is the furthest d-reachable point from p along the line segment. Note that the frontier is a subset of the interior, defining the boundary of reachable points. The frontier for a set of points $\mathbf{P}$ can be defined in terms of the frontiers for each point in the set. The frontier of $\mathbf{P}$ will be defined as the union of the frontiers for each $p \in \mathbf{P}$, minus any points such that

[1] A simple polygon is a polygon where the interior and exterior are well defined.

$p \in Int(p')$ and $p \notin Front(p')$, where $p' \neq p, p \in \mathbf{P}$. In other words, if a point lies on the frontier for a point p, but is strictly in the interior for another point p', then it is not included in the frontier for the set.

The example shown in Figure A.1 illustrates $Int(p)$ and $Front(p)$ for point p inside of a simple polygon. In this figure, the line segment swept to generate $Int(p)$ and $Front(p)$ is represented by the dashed line. Sweeping this line a full 2π radians produces $Front(p)$ represented by the thick black lines Each of the points inside of thick black lines represent $Int(p)$.

The concepts for both the interior and frontier can be extended to n line segments as follows. The n^{th} interior of a point p, denoted $Int(p)^n$, is the set of points d-reachable using a minimum of n line segments. The n^{th} frontier is the furthest set of points d-reachable using n line segments. For both $Int(p)^n$ and $Front(p)^n$, the use of any fewer line segments would mean that one of the line segments has length greater than d. Using this definition, the original definitions of the interior and frontier of a point p are represented as $Int(p)^1$ and $Front(p)^1$ respectively. The $n+1^{th}$ interior can be defined in terms of the interior generation process previously described, where $Front(p)^n$ is the input for this generation. Similarly, the $n+1^{th}$ frontier can be generated in terms of the frontier generation process previously described, where $Front(p)^n$ is the input for this generation.

Figure A.2 shows an example where each intermediate vertex lies on a frontier. This example has four interiors and frontiers.

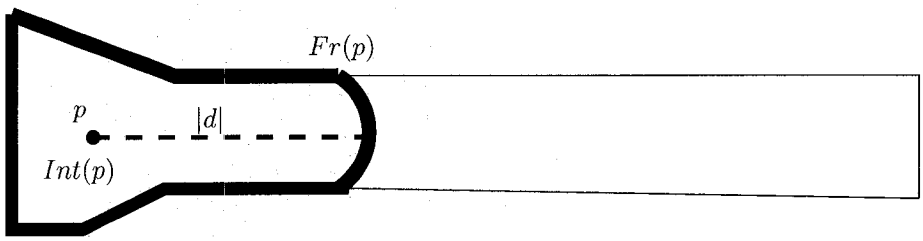

Figure A.1: Interiors and Frontiers between two points

Using the definitions for the interior and frontier, it will be proved that a path between two points can be constructed such that each intermediate point lies on a frontier and that such a path has the minimum number of line segments. This proof uses the following two lemmas.

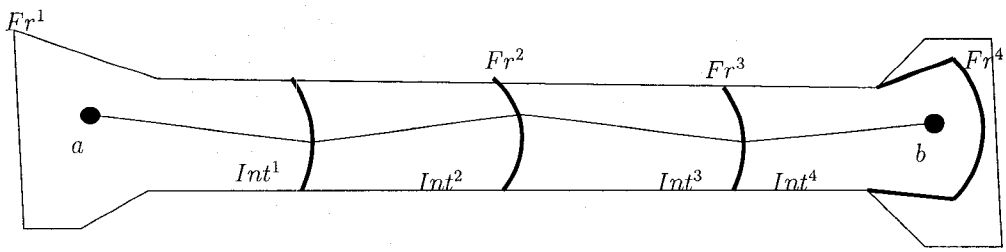

Figure A.2: Path between two points, where each intermediate point lies on a frontier

Lemma 1. *Consider an optimal path, $path^* =< p_0, p_1, p_2, \ldots, p_n >$, containing $n+1$ points. Each point $p_j \in path^*, 1 \leq j \leq n$, lies in the interior $Int(p_0)^j$.*

Proof. Consider the following two cases where this is not true.

Case 1: p_j lies in $Int(p_0)^k, k < j$

This means there will be a path from p_0 to p_j of length k, denoted $< p_0, q_1, \ldots, q_{k-1}, p_j >$. By definition the following path to p_n exists, $< p_0, q_1, \ldots, q_{k-1}, p_j, p_{j+1}, \ldots, p_n >$. This path will be of length $k + n - j < n$. Therefore, this is not an optimal path.

Case 2: p_j lies in $Int(p_0)^k, k > j$

Similarly there will be a path from p_0 to p_j of length k, denoted $< p_0, q_1, \ldots, q_{k-1}, p_j >$. Thus the following path to p_n exists, $< p_0, q_1, \ldots, q_{k-1}, p_j, p_{j+1}, \ldots, p_n >$. This path will be of length $k + n - j + 1 > n$. Therefore, this is not an optimal path.

Thus, each point $p_j, 1 \leq j < n$ in the optimal path $path^*$, lies in the Interior $Int(p_0)^j$.

□

Lemma 2. *Any line segment $\overline{p_i p_{i+1}}$ completely inside Q, such that $p_i \in Int(\mathbf{P})^i$ and $p_{i+1} \in Int(\mathbf{P})^{i+1}$ must contain a point p'_i such that $p'_i \in Front(\mathbf{P})^i$*

Proof. From the definition, the frontier represents the furthest set of d-reachable points. Since the line segment $\overline{p_i p_{i+1}}$ crosses the boundary of points d-reachable using i line segments, there must exist a point p' on $Front(\mathbf{P})^i$

□

Theorem 1 (Frontier Placement Theorem). *Consider two points p_0 and p_n inside a simple polygon Q. Let $path^* =< p_0, p_1, \ldots, p_{n-1}, p_n >$ represent an optimal path. Then there exists a path $path' =< p_0, p'_1, \ldots, p'_{n-1}, p_n >$ between p_0 and p_n such that each intermediate point $p'_i, i = 1 \ldots n-1$ lies on the corresponding frontier $Front(p_0)^i$.*

Proof. This proof will generate such a path backwards, starting from the goal point p_n. Consider the final line segment in $path'$, $\overline{p_{n-1} p_n}$. Using Lemma 1, we can infer that p_{n-1} lies in $Int(p_0)^{n-1}$ and p_n lies in $Int(p_0)^n$. Using Lemma 2, there exists a point p'_{n-1} on this line segment that lies on $Front(P)^{n-1}$. For the construction of the new path $path'$, add the line segment $\overline{p'_{n-1}, p_n}$. Now there exists a point p'_{n-2} on $Front(P)^{n-2}$ from which p'_{n-1} is d-reachable. This follows from the frontier generation process, as the set of points $Front(P)^{n-1}$ was used to generate $Front(P)^{n-2}$. And thus p'_{n-1} must be d-reachable from some point in $Front(P)^{n-2}$. Therefore, the line $\overline{p'_{n-2} p'_{n-1}}$ can be added to the path as well. This argument can be used inductively to trace a path back to a point p'_1 on $Front(p_0)^1$. Also by definition of the frontier, the point on $Front(p_0)^1$ is reachable from p_0. This gives the path $path' =< p_0, p'_1, \ldots, p'_{n-1}, p_n >$.

The following construction generates a path $path'$, where each intermediate point lies on a frontier. This path has $n+1$ points, $n-1$ lying on frontiers, plus the starting and ending points. By

definition, the path $path^*$ also has $n+1$ points. Therefore there exists a path $path'$ where each of the intermediate points in this path is a frontier point, and $path'$ has the minimum number of line segments.

□

Appendix B

Local Placement Test Roads Suite

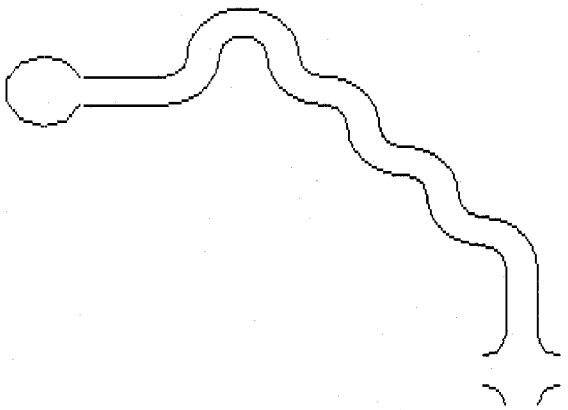

Figure B.1: Test Road A

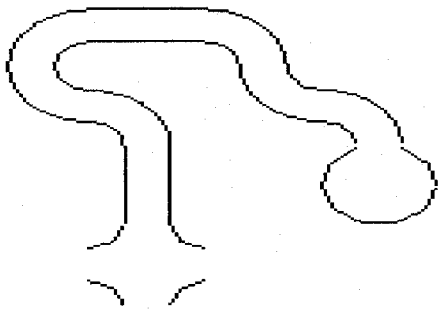

Figure B.2: Test Road B

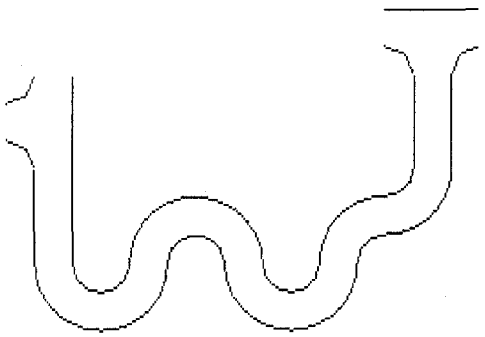

Figure B.3: Test Road C

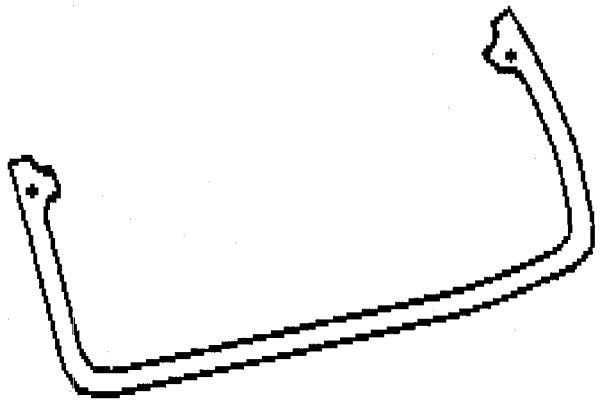

Figure B.4: Test Road D

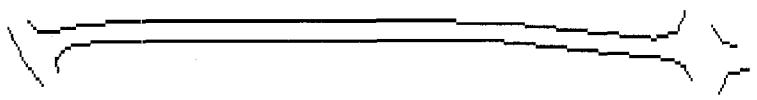

Figure B.5: Real neighborhood straight stretch

Appendix C

Local Placement Test Suite Results

Pipe Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
40	13	13	13	13
50	11	11	11	11
60	11	11	10	10
70	10	9	9	8
80	8	7	7	7
90	8	7	7	7
100	8	7	7	7
110	8	7	7	6
120	8	7	7	6
130	8	6	6	6
140	8	6	6	6
150	8	6	6	6

Figure C.1: Number of manholes in placements for Test Road A

Pipe Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
40	12	12	11	11
50	10	10	10	10
60	10	10	9	9
70	8	10	9	7
80	7	8	7	7
90	7	6	6	6
100	7	7	6	6
110	7	7	6	6
120	7	7	6	6
130	7	7	6	6
140	7	7	6	6
150	7	7	6	6

Figure C.2: Number of manholes in placements for Test Road B

Pipe Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
40	13	12	12	12
50	11	11	11	11
60	11	9	9	9
70	9	8	8	8
80	9	8	8	8
90	9	7	7	7
100	9	8	8	7
110	9	6	6	6
120	9	6	6	6
130	9	6	6	6
140	9	6	6	6
150	9	6	6	6

Figure C.3: Number of manholes in placements for Test Road C

Pipe Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
40	8	8	8	8
50	7	7	7	7
60	6	6	6	6
70	6	5	5	5
80	5	5	5	5
90	5	4	4	4
100	5	4	4	4
110	5	4	4	4
120	4	4	4	4
130	4	4	4	4
140	4	4	4	4
150	4	4	4	4

Figure C.4: Number of manholes in placements for Real Straight Road

Maximum Length	Centerline	Curbpoint Single	Curbpoint Doubling	Frontier
15	0.002	0.043	2,738	0.645
20	0.001	0.034	70.64	0.586
25	0.000	0.027	11.56	0.506
30	0.001	0.024	3.17	0.564
35	0.001	0.019	0.910	0.510
40	0.000	0.018	0.714	0.482
45	0.001	0.017	0.383	0.555
50	0.000	0.015	0.284	0.567
55	0.000	0.015	0.225	0.625
60	0.001	0.015	0.162	0.607

Figure C.5: Layout computation times in seconds for Test Road C

Appendix D

Commercial Pipe Sizes and Costs

0.150	23.004
0.200	31.368
0.250	47.568
0.300	59.928
0.375	73.248
0.450	75.288
0.525	79.368
0.600	112.368
0.675	165.048
0.750	216.168
0.825	250.323
0.900	299.598
0.975	328.713
1.050	375.948
1.200	470.298
1.350	575.568
1.500	702.678
1.650	840.828
1.800	1014.738
1.950	1176.288
2.250	1348.878
2.400	1532.268
2.550	1792.098

Figure D.1: Commercial pipe diameters with per unit cost